HTML & CSS

(Versión 1.5)

José María Morales Vázquez
josemaria@morales-vazquez.com

Este documento se encuentra bajo una LicenciaCreative Commons Atribución-CompartirIgual 3.0 Unported.

CONTENIDO

1. INTRODUCCIÓN....6
Historia de los lenguajes de marcas....6
Algunos lenguajes de marcas....7
Componentes de un lenguaje de marcas....8
Diferencias entre HTML y XHTML....9
Editores de texto enriquecido....10
Navegadores....11
HTML4 vs HTML5....11
2. HTML BÁSICO....14
2.1. ESQUELETO BÁSICO....14
El DOCTYPE....14
Los elementos Básicos....14
2.2. ELEMENTOS DE BLOQUE Y ELEMENTOS DE LÍNEA....15
2.3. LAS ETIQUETAS MÁS COMUNES....16
Listas....17
Las imágenes....18
Los hiperenlaces....18
Comentarios....20
Dirección del texto....20
Acrónimos y abreviaturas....21
Caracteres especiales....21
2.4. TABLAS....21
2.5. ALGUNAS ETIQUETAS Y ATRIBUTOS DESACONSEJADOS MÁS....24
3. CSS. SELECTORES BÁSICOS....25
3.1. INTEGRACIÓN DE HTML CON CSS....25
Comentarios en hojas de estilo CSS....26
3.2. SELECTORES BÁSICOS....26
Selectores de tipo o de etiqueta....26
Selector universal....27
Colores y propiedades básicas del texto....27
Selectores descendientes....30
Selectores de clase....31
Selectores de ID....32
3.3. COMBINANDO SELECTORES....32
4. UNIDADES DE MEDIDA Y OTRAS PROPIEDADES....34
Unidades de medida....34
Unidades de medida específicas de las tipografías....34
Unidades absolutas....34
Unidades relativas....35
Porcentajes....36
4.1. MÁS PROPIEDADES PARA EL TEXTO....36
4.2. APLICAR ESTILOS A LAS LISTAS....38
4.3. DOS NUEVAS ETIQUETAS HTML: div y span....39
4.4. HERENCIA....40

4.5. ESPECIFICIDAD....41
La clausula !important....43
5. EL MODELO DE CAJAS....44
5.1. EL CONCEPTO DE CAJA....44
div es la caja más versatil....45
5.2. PROPIEDADES DE LA CAJA....45
Las dimensiones del contenido....45
Márgenes interior y exterior....45
El borde de la caja....46
Imágenes de fondo en las cajas y sus propiedades....48
Cajas y jerarquía de elementos....49
5.3. POSICIONANDO LAS CAJAS (Y OTROS ELEMENTOS)....49
Las propiedades float y clear....50
Introduciendo la tercera dimensión en el documento....52
Visibilidad, desbordamientos y recortes....53
5.4. OTRAS PROPIEDADES....54
Cambiar el aspecto del puntero del ratón....54
Alterando si un elemento es de bloque o de línea....55
Modificando el aspecto de los bordes de las celdas de una tabla....55
6. SELECTORES AVANZADOS....56
Selector de hijos....56
Selector de hermanos adyacentes (o adyacente, sin más)....56
Selector de atributos....57
6.1. PSEUDO-CLASES Y PSEUDO-ELEMENTOS....58
Pseudo-clases....58
Pseudo-elementos....59
7. ALGUNOS DISEÑOS Y EFECTOS ÚTILES....61
7.1. DISEÑOS DE ANCHO FIJO, LÍQUIDOS (O FLUIDOS) Y ELÁSTICOS....61
Ejemplo 1 – Cajas....61
Ejemplo 2 – Tres columnas....65
Ejemplo 3 – Diseño de alto y ancho fijo....66
Ejemplos 4 y 5 – Diseños de ancho líquido....67
Ejemplo 6 – Diseño fijo, fluido, fijo....67
Ejemplo 7 – Separadores entre columnas transparentes....68
Ejemplo 8 – Sombras y diagonales....68
7.2. DISEÑO DE TABLAS....69
7.3 OTROS TRUCOS....70
Ejemplos 10 y 11 – Transpariencias....70
Ejemplo 12 – Box Punch....72
Ejemplos 13 y 14 – Tooltips....73
Ejemplos 15 y 16 – Menús....75
Ejemplo 17 – Esquinas redondeadas....77
Ejemplo 18 – Sprites....77
Ejemplo 19 – Solapas....78
Ejemplos 20 y 21 – Contorneos....79

8. FORMULARIOS Y JAVASCRIPT....81
8.1. FUNCIONES Y EMPLAZAMIENTO DEL CÓDIGO JAVASCRIPT....81
8.1.1 Funciones....81
8.2. FORMULARIOS....83
8.2.1. Caja de edición de texto (type="text")....84
8.2.2 Caja de edición de contraseñas (type="password")....85
8.2.3. Etiquetas (label)....85
8.2.4 Radio Buttons (type="radio")....85
8.2.5 Checkboxes (type="checkbox")....86
8.2.6. Agrupación de objetos....86
8.2.7 Caja de edición de texto de múltiples líneas....87
8.2.8 Caja de selección de ficheros....87
8.2.9 Cajas de selección (Comboboxes y Listboxes)....87
8.2.10 Botones....88
8.2.11. Nuevos elementos en HTML5....89
8.3. CSS EN LOS ELEMENTOS DE LOS FORMULARIOS....89
8.4. VALIDACIÓN DE FORMULARIOS MEDIANTE JAVASCRIPT....91
8.4.1. Métodos, eventos y propiedades....91
8.4.2. Eventos del elemento form....91
8.4.3. Eventos de los botones....91
8.4.4. DOM, Modelo de objetos del documento....92
8.4.5. Validación del formulario....93
8.4.6. Lista de eventos de Javascript disponibles en los formularios....95
8.5. JAVASCRIPT FUERA DE LOS FORMULARIOS....97
8.5.1. Alert y otras ventanas de confirmación o entrada de datos....97
8.5.2. Lista de eventos de Javascript disponibles de forma genérica....98
8.5.3. Los objetos document y window....99
9. HTML5....100
9.1. ESTRUCTURA DE UN DOCUMENTO HTML5 O XHTML5....100
9.2. CAMBIOS EN LAS ETIQUETAS....101
Etiquetas eliminadas....101
Nuevas etiquetas para la organización del documento....101
Solventando los problemas de versiones antiguas de Internet Explorer....101
9.3. MEJORAS EN LOS FORMULARIOS: NUEVOS TIPOS DE INPUT....102
9.4. MODERNIZR....104
9.5. ETIQUETAS MULTIMEDIA....107
Audio....107
Formatos de audio....108
Vídeo....108
Formatos de vídeo....109
Archivos de Flash....109
9.6. DIBUJO MEDIANTE LA ETIQUETA CANVAS....109
9.7. GEOLOCALIZACIÓN....113
9.8. MÁS COSAS....114
10. CSS3....116

10.1. LOS NUEVOS SELECTORES DE CSS3....116
Pseudo-elementos....116
Pseudo-clases....116
Nuevos selectores para uso con formularios....117
El selector :not()....117
Nuevos selectores de atributos....118
Selector general de elementos hermanos....119
9.2. LAS NUEVAS PROPIEDADES DE CSS3....119
Gradientes y una nueva paleta de colores....119
Fondos múltiples....120
Esquinas redondeadas y bordes decoradas....121
Sombras....122
Opacidad....123
Más control de la tipografía....123
Columnas más fáciles....124
Transformaciones y trancisiones....125
11.ENLACES, BIBLIOGRAFÍA Y REFERENCIAS....128

1. INTRODUCCIÓN

Un lenguaje de marcas es una forma de codificar un documento que, junto con el texto, incorpora etiquetas (o marcas) que contienen información adicional acerca de la estructura del texto o su presentación. Tal vez la forma más primitiva que hemos usado de lenguaje de marcas fuera un dictado en el que la persona que dicta nos va dando notas acerca de lo que tenemos que poner en negritas, cursivas, etc. En el mundo de los ordenadores llevamos mucho tiempo usando lenguajes de marcas. Wordstar, uno de los primeros procesadores de texto que existieron para el mundo de los PC o Latex, el programa de autoedición favorito para edición profesional de textos, son dos ejemplos de ello:

Wordstar
La lluvia en ^BSevilla^S es una ^Ymaravilla^S.

LATEX
La lluvia en \textbf{Sevilla} es una \textit{maravilla}.

Resultado:
La lluvia en **Sevilla** es una *maravilla*.

Es común en muchos ámbitos confundir un lenguaje de marcas con un lenguaje de programación. Pero no: se trata de cosas diferentes. Hay, fundamentalmente, tres carencias de los lenguajes de marcas que los distinguen:

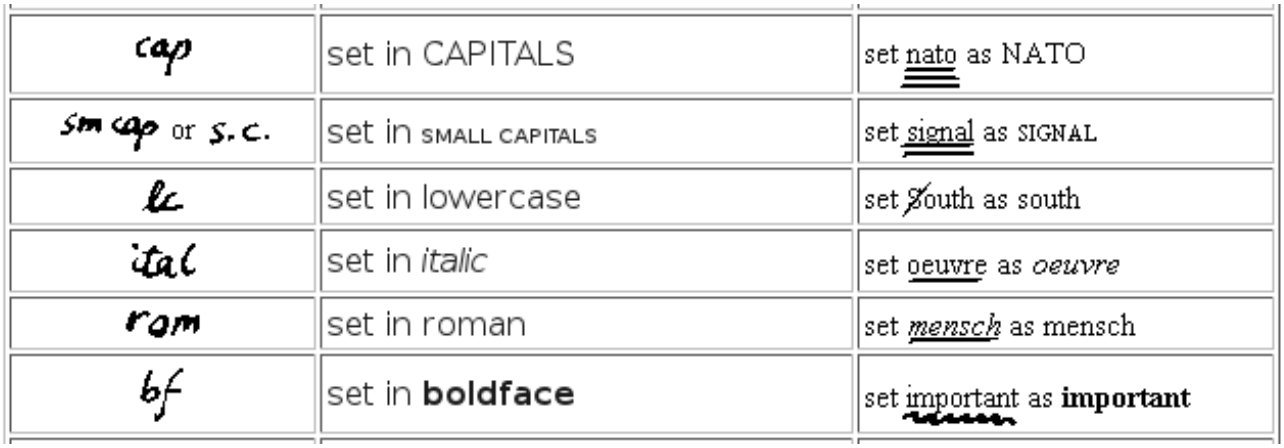
- No tienen funciones aritméticas
- No tienen variables
- No tienen estructuras de control

Las características principales de un lenguaje de marca son las siguientes:

- Se usan siempre sobre texto plano.
- Las marcas se entremezclan con el contenido del documento aunque, en general, es fácil distinguir unas del otro.
- Su procesamiento es muy sencillo.
- Son muy flexibles.

Historia de los lenguajes de marcas

La primera referencia que se tiene de un lenguaje de marcas como tal está aún alejado de la informática. Se trata de la práctica de los empleados de imprenta de anotar marcas con instrucciones en los márgenes de las pruebas de impresión:



En el mundo de la informática el "padre" de los lenguajes de marcas más importantes de la actualidad surge en los años 80. Se llama SGML (Standard Generalized Markup Language) y define unas reglas básicas para el etiquetado de documentos mediante marcas. SGML no es en realidad un lenguaje sino un metalenguaje, es decir, un lenguaje creado con la finalidad de definir otros lenguajes a partir de él. RTF (Rich Text Format), HTML (HyperText Markup Language) o XML (Extensible Markup Language) derivan de SGML.

Algunos lenguajes de marcas

HTML es, sin duda, el lenguaje de marcas más usado y la base de las páginas web. Su primera versión, que data de los años 90, describía sólo 22 elementos diferentes. La versión actual es la 4.01 aunque la versión 5 está en una fase muy avanzada y se empieza a utilizar cada vez más a pesar de que se trata de una norma que no está aún aprobada de forma definitiva.

XML deriva de SGML y se trata también de otro metalenguaje. Parte de un subconjunto de SGML y añade algunas restricciones nuevas de forma que los lenguajes derivados a partir de XML resulten más sencillos y fáciles de interpretar que los derivados directamente de SGML.

XHTML (Extensible HTML) es equivalente a HTML pero deriva de XML. La versión actual es la 1.1 y la versión 2.0 está en borrador. Las diferencias entre HTML 4 y XHTML 1 son nímias.

CSS (Cascading Style Sheets) es un lenguaje usado para definir la presentación de un documento en HTML o XHTML. No se trata en realidad de un lenguaje de marcas, pero se encuentra indisolublemente unido a estos dos. La versión actual es la 2 y la versión 3 se encuentra en fase de borrador. Al igual que ocurre con HTML5, CSS3 se usa cada vez más a pesar de no tratarse de una norma definitiva.

XSL (Extensible Stylesheet Language) describre la forma en que debería de mostrarse la información contenida en un documento con formato XML.

SGML está definido como una norma ISO mientras que el resto de los lenguajes mencionados están definidos por la W3C (Worl Wide Web Consortium).

Componentes de un lenguaje de marcas

Veamos un primer ejemplo muy simple de HTML y analicemos los diferentes componentes que pueden aparecer en un lenguaje de marcas y algunas de las características particulares de HTML:

```
<html>
     <head>
          <title>.:: PAGINA DE PRUEBAS ::.</title>
     </head>
     <body bgcolor="black" text="white">
          <center><h1>TITULAR</h1></center>
          Prueba de <strong>texto normal</strong><br />
          Segunda linea
          -Tambien en la segunda linea
     </body>
</html>
```

Se trata de un ejemplo reducido al mínimo y que, incluso, presenta algunos defectos o incorrecciones, pero nos vale como primer contacto. Veamos como se vería en un navegador:

- **Elementos:** Constan de una etiqueta de inicio, una de fin y todo lo que haya enmedio. Los elementos constan, a su vez, de tres elementos: etiquetas, atributos y contenido. Un ejemplo de elemento sería este:

```
<strong>texto normal</strong>
```

- **Etiquetas o tags:** Son las marcas propiamente dichas y habitualmente van entre corchetes quebrados <> Las hay de inicio y de fin aunque, en algunos casos, ambas pueden coincidir en una sóla partícula con una sintaxis especial cuando el elemento no tiene contenido:

```
<title> ... </title>
<br/>
```

- **Contenido:** Es el texto base informativo del documento. Por ejemplo, en el anterior elemento con las etiquetas strong el contenido sería el texto que luego aparecerá en negritas.
- **Atributos:** Es una pareja compuesta por un nombre y un valor que se encuentra dentro de una etiqueta de inicio e identifica las propiedades asociadas al elemento.

```
bgcolor="black"
```

  En realidad, en HTML convencional es posible encontrar atributos sin valor, pero en XHTML, que será nuestro preferido, esto no es válido como veremos a continuación.

Diferencias entre HTML y XHTML

Como hemos dicho hace un momento, XHTML introduce ciertas restricciones a HTML para hacer que el lenguaje resultante sea más sencillo y fácil de interpretar. Estas diferencias son las siguientes:

- Los elementos deben de cerrarse siempre. En HTML normal es perfectamente válido, por ejemplo, empezar un párrafo con la etiqueta y, sin poner la marca de fin de párrafo comenzar un segundo párrafo de nuevo con . Puesto que no podemos anidar dos párrafos uno dentro de otro el intérprete debe de reconocer que al empezar el segundo párrafo debería antes de terminar el primero. En XHTML hay que cerrar el primero explicitamente o en caso contrario tendremos un error de validación:

```
HTML: <p>Primer párrafo <p>Segundo párrafo
XHTML: <p>Primer párrafo</p><p>Segundo párrafo</p>
```

- Los elementos sin contenido deben de "cerrarse" siempre usando una etiqueta especial que realiza el autocierre en la misma etiqueta de inicio:

```
HTML: <br>
```

**XHTML:
**

- Los elementos anidados no deben solaparse. En HTML está permitido pero en XHTML es incorrecto:

  HTML: Texto

  XHTML: Texto

- Los valores de los atributos deben de ir siempre entre comillas simples o dobles:

  HTML: <body bgcolor=black text=white>

  XHTML: <body bgcolor="black" text='white'>

- Los nombres de etiquetas y atributos deben de ir siempre en minúsculas

  HTML: <body BGCOLOT=black text=WHITE>

  XHTML: <body bgcolor="black" text="white">

- No está permitido usar un atributo sin valor (minimización de atributos)

  HTML: <textarea readonly>

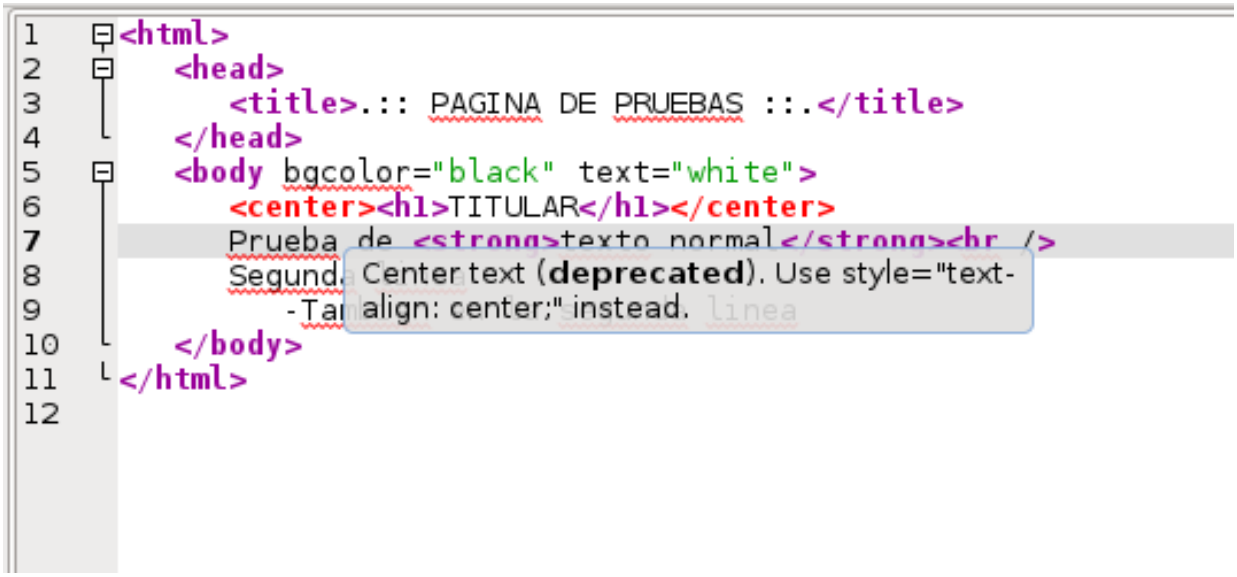
  XHTML: <textarea readonly="readonly">

- Los atributos y etiquetas desaprobados o desaconsejados (deprecated) en HTML 4 no son válidos en XHTML. Los veremos más adelante.

Una aclaración en todo esto: un fichero que valida correctamente como XHTML siempre validará como HTML pero no a la inversa, es decir, si escribimos de acuerdo a la norma marcada por XHTML también conseguimos ficheros válidos para HTML con una sintaxis más clara, legible y menos sujeta a errores de interpretación. Por eso será nuestra elección a lo largo de todo este manual.

Editores de texto enriquecido

A la hora de trabajar con lenguajes de marcas, la elección de un editor de texto enriquecido con ayuda contextual es muy importante:



Existen muchos editores que cumplen para esta labor, pero si te ves perdido a la hora de empezar puedes probar con uno de estos dos:

- Bluefish (multiplataforma): http://bluefish.openoffice.nl/
- Notepad++ (sólo Windows): http://notepad-plus-plus.org/

Navegadores

En el caso del HTML y el XHTML, el navegador web funciona como visor o intérprete del lenguaje de marcas y su respeto por los estándares es fundamental. Muchas veces en el pasado se ha utilizado la posición de supremacía de uno de ellos para desviarse del estándar y perjudicar a la competencia. Afortunadamente estas prácticas parecen abandonadas hoy en día. Tenemos varios recursos para comprobar la validez de un fichero escrito en HTML o la forma en que el navegador implementa los estándares:

Validadores de HTML y CSS:

- http://validator.w3.org/
- http://jigsaw.w3.org/css-validator/

Tests de cumplimiento de estándares:

- http://acid2.acidtests.org/
- http://acid3.acidtests.org/
- http://www.css3.info/selectors-test/

HTML4 vs HTML5

HTML5, aún está en estado de borrador, pero supone un enorme y esperado cambio. La última revisión de HTML, la 4.01, se hizo en 1999 y desde entonces Internet, nuestros hábitos y nuestras espectativas sobre lo que deberíamos de recibir al visitar una web han cambiado mucho.

En paralelo a HTML5 existirá una versión de XHTML5 más estricta que también se encuentra aún en estado de borrador. Las diferencias entre HTML5 y XHTML5 serán las mismas que las ya vistas para HTML4 y XHTML4: apenas unas restricciones extras para facilitar la interpretación del lenguaje.

HTML5, además, irá acompañado de CSS3 que mejora y simplifica el diseño web.

Las principales novedades de HTML5 y CSS3 son las siguientes:

- Ya no se habla de páginas web sino de aplicaciones web. Esto quiere reforzar el cambio en la filosofía que se persigue con esta nueva versión del lenguaje.
- La separación entre presentación y contenido se ve reforzada. En HTML5 todo lo relativo al diseño irá en los CSS. HTML5 sin CSS es en blanco y negro y no se debería siquiera elegir un tipo de letra diferente al que el navegador

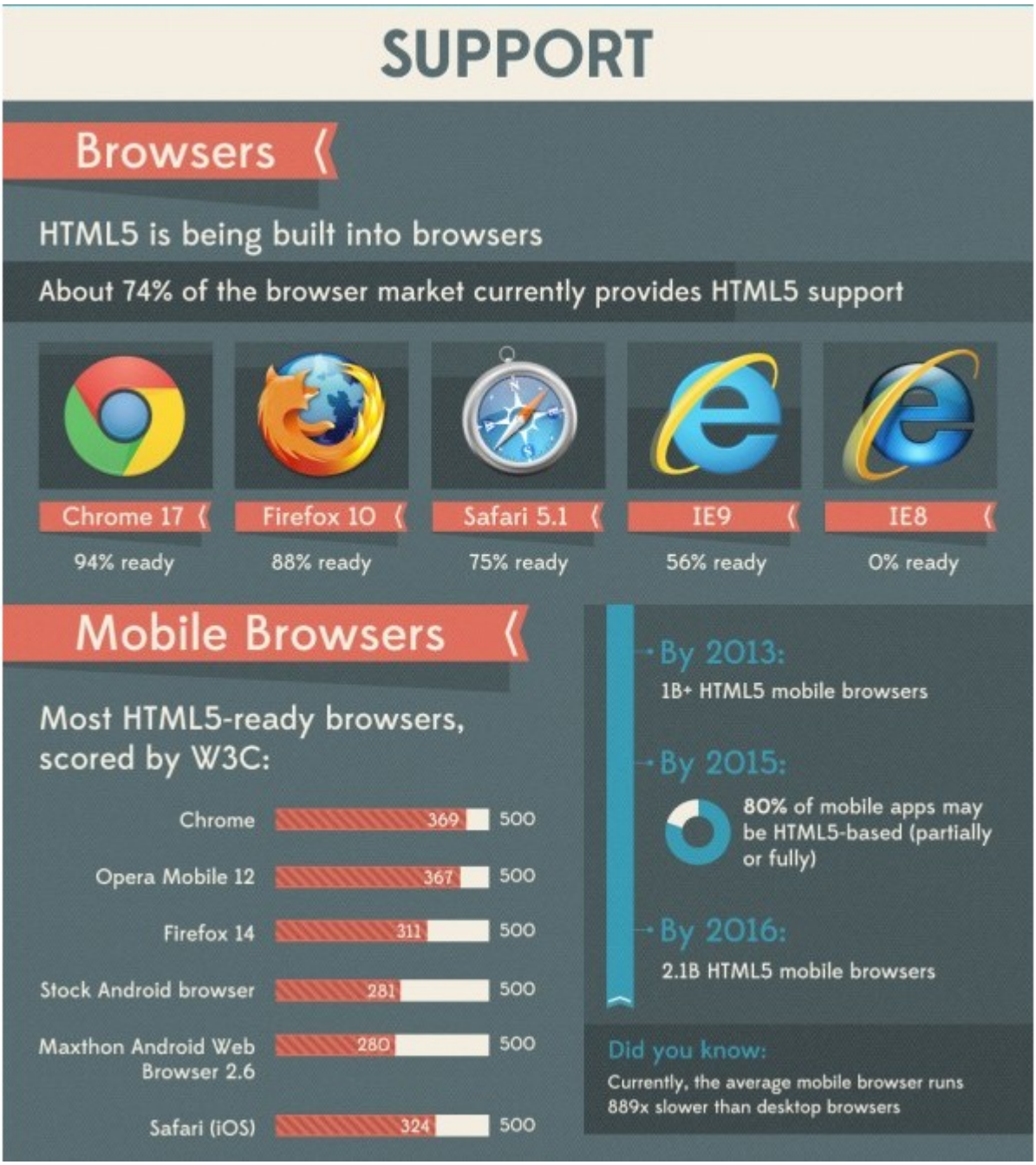
nos muestra por defecto.

- Existe una gran mejora en todo lo relativo al manejo de formularios
- HTML5 se encuentra plenamente integrado con Javascript. De hecho, se encuentra ligado de forma casi indisoluble a una amplia colección de API's de Javascript que le proporcionan soporte para diseño 2D y 3D, geolocalización, arrastrar y soltar, multimedia, etc.

Entonces, si existen todas estas ventajas y el borrador de HTML5 está ya prácticamente cerrado ¿por qué cuesta tanto trabajo encontrar webs realizadas en HTML5?¿Por qué no empezamos nosotros a trabajar directamente con él? La respuesta está en los navegadores, los intérpretes de todo esto. Veamos una fotografía de los navegadores más usados (en septiembre de 2011):

Las cifras impresas en rojo representan el porcentaje de HTML5 que ese navegador es capaz de interpretar correctamente. Como vemos, IE8, el navegador más usado con más de un cuarto del total, sólo interpreta el 20% de la norma HTML5.

En una infografía más reciente (septiembre 2012) se confirman estos porcentajes: uno de cada cuatro de los navegadores actuales no proporciona aún soporte para HTML5:



Hoy, febrero de 2014, las tornas han cambiado bastante desde que empezamos a escribir este manual y ya es perfectamente factible encontrar bastante a menudo aplicaciones web construidas completamente en torno a HTML5

2. HTML BÁSICO

En este segundo apartado trataremos los elementos más básicos de HTML siguiendo las siguientes normas:

- Nos centraremos en la versión 4.01, tratando de respetar la norma XHTML 1.0 más estricta (aunque siempre hablaremos de html por simplificar) y acercándonos cuanto podamos a cumplir con lo que establecerá HTML5 para facilitar una futura trancisión.
- Veremos en algunos casos algunas de las etiquetas y/o argumentos de uso desaconsejado indicándolo siempre a continuación. Puesto que en nuestro desempeño no sólo haremos páginas web sino que tendremos que modificar las que otros han creado necesitamos saber para quer sirven.
- Trataremos, también, de proporcionar siempre que podamos un contenido semántico a nuestras etiquetas
- Separaremos la parte de diseño (que en un futuro aprenderemos a aplicar mediante CSS) del contenido del documento. No te preocupes si, por el momento, tus páginas resultan demasiado simples.

2.1. ESQUELETO BÁSICO

El DOCTYPE

Todas las páginas HTML deberían de comenzar por un DOCTYPE o Declaración de tipo de documento que es la que asocia dicha página con un DTD (Document Type Definition) o Definición de Tipo de Documento e indica esto al navegador. En realidad, y para no hacerlo más complicado de lo que necesitamos saber, se trata meramente de una línea formal en la que indicamos al navegador la norma que sigue el lenguaje de marcas que debe de interpretar.

En la Wikipedia tenemos diferentes tipos de DOCTYPE:

http://es.wikipedia.org/wiki/Declaraci%C3%B3n_de_tipo_de_documento

La definición de HTML 4.01 Trancisional, la más habitual, es esta:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

El DOCTYPE debe de ocupar siempre la primera línea del documento html sin que exista ni un sólo espacio antes de ella o no tendremos garantías de que todos los navegadores la reconozcan.

Los elementos Básicos

Los elementos básicos en una página HTML son **html, head** y **body**. La expresión mínima formal de un documento HTML sería este:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html lang="es">
	<head>
		<meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
		<title>Lenguajes de Marcas</title>
	</head>
	<body>
		<p>¡Hola mundo!</p>
	</body>
</html>
```

El contenido del elemento **html** es toda nuestra página web (salvo el DOCTYPE). Lleva un atributo denominado **lang** que indica el idioma con que está hecha la web.
El contenido del elemento **head** está formado, a su vez, por una serie de elementos que no se visualizan directamente (salvo el título o **title** que aparecerá en la barra del navegador). El elemento **meta** puede aparecer con diferentes indicaciones dentro de esta sección de cabecera. La más importante es, tal vez, el juego de caractéres con que hemos escrito el documento HTML. Si nos equivocamos puede que algunos caracteres especiales (vocales acentuadas, signos de puntuación, etc.) no aparezcan correctamente. En Linux se suele usar **UTF-8** mientras que si usas windows suele ser el **ISO-8859-1.**
Por último, el elemento **body** contendrá toda la parte visible de la web. En este caso se trata de un único párrafo (etiqueta **p**) con un mensaje típico de bienvenida.

2.2. ELEMENTOS DE BLOQUE Y ELEMENTOS DE LÍNEA

Las páginas web están hechas siempre de elementos rectangulares que se colocan uno debajo de otro, uno junto a otro o uno encima de otro. No hay nada más. Cualquier cosa que nos parezca diferente se trata de un truco de diseño.

Dentro de esto, tenemos que distinguir entre los elementos de bloque y los de línea. Los de bloque podríamos asemejarlos con los párrafos de un texto: ocupan todo el ancho de la superficie útil del documento (o del área del documento al que han sido restringidos) y pueden tener otros elementos, iguales o diferentes, arriba, abajo o, en algunas ocasiones, en su interior (aunque esto último no siempre es correcto). Un párrafo, por ejemplo, no debería de contener a otro párrafo en su interior. También pueden contener elementos de línea.

Los elementos de línea se sitúan uno debajo de otro, uno al lado de otro (como, por ejemplo, un texto en negrita que puede ir seguido de otro en cursiva) o, incluso, uno dentro de otro (una cita puede tene una palabra en negrita en su interior). Los elementos de línea no pueden tener en su interior un elemento de bloque.

Otra característica importante de HTML es que por defecto los saltos de línea

consecutivas que hagamos manualmente en nuestro documento no se respetan. Por ejemplo, si yo escribiera lo siguiente:

```
<p>Hola a todos


los compañeros

del curso de HTML</p>
```

En el navegador se mostraría lo siguiente:

```
Hola a todos los compañeros del curso de HTML
```

Cuando aparecen múltiples espacios en blanco sólo se respeta el primero. Por ejemplo:

```
<p>Hola a todos                los compañeros del curso     de HTML</p>
```

Se vería en el navegador así:

```
Hola a todos los compañeros del curso de HTML
```

2.3. LAS ETIQUETAS MÁS COMUNES

```
<p></p>                     Define un párrafo
<br />                      Salto de línea
<b></b>                     Negritas
<strong></strong>           Negritas
<i></i>                     Cursivas
<em></em>                   Cursivas
<sup></sup>                 Superíndice
<sub></sub>                 Subíndice
<del></del>                 Texto tachado
<pre></pre>                 Preformateado
<h1></h1>                   Encabezado de nivel 1
<h2></h2>                   Encabezado de nivel 2
...
<h6></h6>                   Encabezado de nivel 6
<cite></cite>               Cita corta
<q></q>                     Cita corta
<blockquote></blockquote>   Cita larga
<hr />                      Línea horizontal
```

Todas las etiquetas llevan asociada una información de estilo por defecto, pero esta puede variar entre cada navegador. Más adelante, cuando veamos CSS, veremos como modificar dicho estilo y tratar de que sea lo más independiente posible del navegador que usemos para verlas.

En las etiquetas en las que podemos elegir, preferiremos siempre las que conllevan información semántica (**strong, em, cite**)frente a las que sólo nos dan información de estilo (**b, i, q**).

La etiqueta **pre** de texto "preformateado" define un área en el que se respeta de forma íntegra los saltos de línea y espacios en blanco que hayamos realizado al escribir el texto.

Las etiquetas **h1, h2, h3, h4, h5, h6, p, blockquote,** y **pre** son etiquetas que definen elementos de bloque, mientras que **strong, b, em, i, sup, sub**, **del, bdo, cite** o **q** son etiquetas de elementos de línea.

Listas

Las listas se definen mediante las etiquetas **ul, li** y **ol.**

```
<ul></ul>   Define una lista no ordenada (con "boliches")
<ol></ol>   Define una lista ordenada
<li></li>   Define cada item de la lista en cualquiera de los anteriores
```

Por ejemplo:

```
<ol>
      <li>Lunes</li>
      <li>Martes</li>
      <li>Miércoles</li>
      <li>Jueves</li>
</ol>
```

Las listas se pueden anidar, pero teniendo en cuenta que la nueva lista tiene que abrirse antes de cerrar el item anterior. Por ejemplo:

```
<ul>
      <li>Punto 1</li>
      <li>Punto 2
            <ul>
                  <li>Apartado A del punto 2</li>
                  <li>Apartado B del punto 2</li>
            </ul>
      </li>
      <li>Punto 3</li>
</ul>
```

El atributo **type** (**¡¡¡de uso desaconsejado!!!**) nos permitiría cambiár el tipo de símbolo en las listas desordenadas. Por ejemplo **<ul type="square">.** Otros valores válidos para este atributo son **"disc"** o **"circle"**.

Con la etiqueta **ol** tenemos dos atributos (**también de uso desaconsejado**) que nos permiten cambiar el comportamiento de la lista: **start** para variar el número por el que se inicia la lista ordenada (que por defecto es el 1) y **type** para cambiar el formato de enumeración que usamos. Ejemplos: **<ol start="50">** o **<ol type="A">**. Otros tipos válidos son **"1", "I", "i"** o **"a"**.

La etiqueta li se puede usar fuera de un bloque **ol** o **ul** y puede llevar un atributo denominado **type** con cualquiera de las opciones válidas en **ol** o **ul** (**también está desaconsejado su uso**).

Las imágenes

Para insertar imágenes se utiliza la etiqueta **img** Es una etiqueta única (sin contenido) por lo que tenemos que usar el método de auto-cerrado para cumplir las normas de XHTML. La sintaxis básica es esta:

```
<img src="imagen.jpg" alt="Texto alternativo" />
```

El atributo **src** es obligatorio e indica el lugar donde está la imagen. Se puede usar una ruta relativa, absoluta o indicar una URL web.:

```
<img src="pics/2012/imagen.jpg" alt="Texto alternativo" />
<img src="http://www.example.com/imagen.jpg" alt="Texto alternativo" />
```

El atributo **alt** también es obligatorio en XHTML. Algunos otros atributos útiles y habituales y muestra un texto alternativo que se mostrará cuando la imagen no pueda visualizarse por cualquier motivo:

```
<img src="imagen.jpg" alt="Texto alternativo" title="Texto resaltado al
pasar el ratón" height="200" width="400" />
```

El atributo **title** muestra un "tooltip" al pasar el ratón sobre la imagen. **width** y **height** nos permiten modificar el ancho y/o el alto de la imagen expresando el tamaño deseado en píxeles. Si no los usamos la imagen se mostrará con su tamaño original. Si modificamos sólo uno de ellos el otro quedará con su tamaño original y la imagen resultará anamórfica.

El atributo **border** muy útil para indicar que queremos una imagen con borde (sin él cuando lleva incorporado un hiperenlace) **está desaconsejado** en HTML4 y no existe en HTML5. Indica el número de píxeles que tendrá el borde de la imagen. Por defecto las imágenes se muestran sin borde salvo que pertenezcan a un hiperenlace (ver el punto siguiente)

```
<img src="pics/2012/imagen.jpg" alt="Texto alternativo" border="2"/>
```

La etiqueta **img** es una etiqueta de línea.

Los hiperenlaces

Los enlaces a otras páginas se realizan mediante la etiqueta **a**

```
<a  href="http://www.google.es" />Google</a>
```

El atributo **href** es obligatorio e indica la dirección destino del enlace. Puede ser una página web en el mismo disco, en el mismo servidor, en otro servidor, a un punto de la misma página, a una dirección de correo, etc.

```
<a href="album.html">Álbum de fotos</a>
<a href="mailto:josemaria@uponaday.net">Enviar un correo</a>
```

Otros atributos son **title** y **target**:

```
<a href="album.html" title="Texto resaltado al pasar el ratón"
target="_blank >Álbum de fotos</a>
```

El atributo **title** nos muestra un "tooltip" al pasar el ratón sobr el enlace, mientras que **target** nos permite indicar donde queremos que se abra la nueva página a la que hace referencia el enlace. El valor por defecto de **target** es **"_self"** e indica que se abrirá en la misma página donde lo pulsamos. Si usamos el valor **"_blank"** la página se abrirá en una solapa o ventana nueva.

Para hacer un enlace a un punto dentro de la misma página usamos la siguiente sintaxis:

```
<a href="#C4">Ver el capítulo 4</a>
...
<a id="C4">CAPÍTULO 4</a>
```

Como vemos, usamos un elemento **a** para crear el hiperenlace usando un término a nuestra elección precedido del carácter especial **#**. En el punto de destino del hiperenlace debemos de crear otro elemento **a** pero en este caso usaremos el atributo **id** y como contenido del mismo pondremos el mismo término que pusimos como valor del atributo **href** pero en este caso prescindiendo del carácter **#**

Para ir a un punto dentro de otra página usamos un formato similar:

```
<a href="otrapagina.html#C4">Ver el capítulo 4 de otrapagina.html</a>
```

En esa otra página deberíamos de tener un elemento **a** con un atributo **id** de forma similar al caso que hemos visto antes:

```
<a id="C4">CAPÍTULO 4</a>
```

Antes de HTML4 se usaba el atributo **name** en lugar de **id.** En HTML4 **name está desaconsejado** y en HTML5 no existe. El atributo **id** que nos sirve como referencia para el destino del hipervínculo no tiene porque ir en un elemento de tipo **a**. Podría ir en casi cualquier otro elemento. Por ejemplo:

```
<a href="#lista_1">Ver la lista de características</a>
...
```

```
<ul id="lista_1">
      <li>Procesador ARM</li>
      <li>128 Mbytes RAM</li>
      <li>16 Gbytes HDD Compact Flash</li>
</ul>
```

El valor del atributo id debe de ser único y no debe repetirse nunca en la misma página web.

Un ejemplo más completo para un enlace a email en el que rellenamos no solo la dirección de destino, sino también el asunto y el campo cc:

```
<a href="mailto:josemaria@uponaday.net?subject=De un alumno de
DAW&cc=departamento@reyfernandovi.com">Enviar correo</a>
```

NOTA IMPORTANTE: Para que valide correctamente como XHTML estricto debemos de sustituir el signo **&** por el código **&** No se ha hecho por claridad, pero lo veremos más adelante.

La etiqueta **a** es una etiqueta de línea.

Comentarios

Los comentarios en HTML se escriben entre los símbolos **<!--** y **-->**

```
<!-- Esto es un comentario y jamás se visualizará
     Un comentario puede tener varias líneas -->
```

Una nota respecto a los comentarios y, en general, sobre cualquier otra cosa que hagamos en las páginas web realizadas con HTML y CSS: Todo, absolutamente todo lo que escribamos en ellas llegará al navegador del usuario tal y como las hayamos escritos y será interpretado por este. Los comentarios y algún otro argumento puede que no resulten visibles, pero si el usuario quiere (y sabe hacerlo) siempre podrá verlos, por lo general con la combinación de teclas Ctrl+U o eligiendo la opción adecuada en el menú del navegador. Por tanto, tenemos que tener mucho cuidado con lo que hacemos en estos campos aparentemente ocultos. Como norma general lo que no queramos que vea ningún usuario no debería de escribirse.

Dirección del texto

Para indicar si el texto va de izquierda a derecha o de derecha a izquierda se usa la etiqueta **bdo** con el atributo **dir**:

```
<p><bdo dir="ltr">dábale arroz a la zorra el abad</bdo></p>
<p><bdo dir="rtl">dábale arroz a la zorra el abad</bdo></p>
```

ltr indica que el texto va de izquierda a derecha (left to right) y **rtl** de derecha a

izquierda. El valor por defecto es **ltr**.

Acrónimos y abreviaturas

Usamos las etiquetas **acronym** y **abbr**, respectivamente, con el atributo **title** que mostrará una nota explicativa o "tooltip" al pasar el ratón por encima.

```
Mi ordenador tiene 512 <abbr title="Megabytes">Mb</abbr> de memoria
<acronym title="Random Access Memory">RAM</acronym>
```

NOTA IMPORTANTE: acroym desaparece en HTML5, así que no conviene usarlo.

Caracteres especiales

Los caracteres especiales que se usan para las etiquetas no pueden escribirse tal cual en un documento HTML, así que hay que usar una codificación especial.

```
&lt;        <
&gt;        >
&quot;      "
&amp;       &
```

Existen también un caracter especial que pueden sernos úti:

```
       espacio irrompible y no eliminable
```

Si necesitamos algún otro caracter que no aparece en nuestro teclado (símbolos tipográficos, etc.) podemos recurrir a la siguiente tabla:

http://www.ascii.cl/es/codigos-html.htm

2.4. TABLAS

Las tablas se han usado durante mucho tiempo en HTML como el sustituto de otras alternativas de diseño. Nosotros no deberíamos de hacer esto jamás. Una tabla debe de usarse para lo que es: para presentar datos en un formato tabulado y no, por ejemplo, para presentar información en cuatro columnas separadas. Para esto ya veremos la forma correcta de hacerlo.

La sintaxis que muestra una tabla sencilla quedaría con dos filas y tres columnas sería así:

```
<table>
   <tr>
      <td>1</td><td>2</td><td>3</td>
   </tr>
   <tr>
      <td>4</td><td>5</td><td>6</td>
   </tr>
</table>
```

La etiqueta **<table></table>** es la que define y contiene al elemento tabla. Cada fila va encerrada como contenido de la etiqueta **<tr></tr>**. Cada celda va delimitada por la etiqueta **<td></td>.** El número de celdas que hay en una fila determina el número de columnas de la tabla.

Casi todos los atributos de la etiqueta table (**width, cellpadding, cellspacing, frame, rules** y **summary**) **han sido desaconsejados y/o han desaparecido** en HTML5. El atributo **border** (con un valor numérico que indica el número de píxeles de este) si puede seguir usándose. La etiquetas **tr** también **han visto desaparecer como desaconsejados sus atributos align** y **valign**. La etiqueta **td**, por último, **tampoco dispone de bendiciones para usar sus atributos width, height, align, valign, abbr, axis** y **scope.** Puedes ver para que sirven estos y otros atributos desaconsejados en las siguientes referencias:

http://www.w3schools.com/tags/tag_table.asp
http://www.w3schools.com/tags/tag_tr.asp
http://www.w3schools.com/tags/tag_td.asp

La etiqueta **td** continúa disponiendo de los atributos **colspan y rowspan** que sirven para fusionar celdas horizontal o verticalmente.

```
<table border="1">
   <tr>
      <td colspan="3">1, 2 y 3</td>
   </tr>
   <tr>
      <td>4</td><td>5</td><td>6</td>
   </tr>
</table>

<table border="1">
   <tr>
      <td rowspan="2">1 y 4</td><td>2</td><td>3</td>
   </tr>
   <tr>
      <td>5</td><td>6</td>
   </tr>
</table>
```

La etiqueta **<th>** se usa para definir una celda como de encabezado y puede usarse tanto en filas como en columnas:

```
<table border="1">
      <tr>
      <th></th><th>A</th><th>B</th><th>C</th>
   </tr>
   <tr>
      <th>D</th><td>1</td><td>2</td><td>3</td>
   </tr>
   <tr>
```

```
      <th>E</th><td>4</td><td>5</td><td>6</td>
    </tr>
</table>
```

La etiqueta **caption** se usa para dar título a la tabla. Debe de aparecer una sóla vez e inmediatamente despueś de la apertura de la etiqueta **table**:

```
<table border="1">
      <caption>Título de la tabla</caption>
      <tr>
      ...
</table>
```

Las etiquetas **thead**, **tbody** y **tfoot** nos permiten estructurar de forma lógica la información de la tabla, pudiendo incluso variar el orden de los mismos (definiendo el cuerpo por delante de la cabecera, por ejemplo) si así nos conviniera:

```
<table border="1">
  <thead>
    <tr>
      <th>Mes</th>
      <th>Ingresos</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Total</td>
      <td>2800,00 €</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>Enero</td>
      <td>1200,00 €</td>
    </tr>
    <tr>
      <td>Febrero</td>
      <td>1600,00 €</td>
    </tr>
  </tbody>
</table>
```

Por último, disponemos también de la etiqueta **colgroup** que nos permitirá agrupar de forma lógica algunas columnas de la tabla para aplicarles estilos de forma conjunta, pero no tiene sentido ver como funciona hasta que no trabajemos con CSS.

NOTA importante: Como hemos dicho en varias ocasiones, HTML5 está aún en fase de borrador. Los atributos que aquí aparecen como válidos o desaconsejados pueden variar de las especificaciones finales de la norma. Por

ejemplo, aún hay discrepancias acerca de que pasará finalmente con los atributos **border** y **summary** de la etiqueta **table**.

2.5. ALGUNAS ETIQUETAS Y ATRIBUTOS DESACONSEJADOS MÁS

font y **basefont** están desaconsejadas en HTML 4.01 y desaparecen en HTML5. Para cambiar las caracterísitcas del texto hay que usar hojas de estilo. Un ejemplo de uso:

```
<font face="arial narrow" color="green" size=”+4”>Esto no se hace</font>.
```

center también está desaconsejado. El atributo **align**, de la etiqueta p (que puede tomar valores de left, right, center o justify) también. Hay que usar hojas de estilo en su lugar.

```
<p align="right">Esto no debe de hacerse</p>
<center>Y esto tampoco</center>.
```

s y **strike** se desaconsejan, pero la etiqueta **del** que hace exactamente lo mismo sigue siendo válida.

```
Si queremos mostrar texto tachado deberíamos de <del>hacerlo así</del> y
nunca <s>así</s> o <strike>así</strike>.
```

La etiqueta **u** para subrayar texto también deja de estar soportada.

```
<u>Así se subrayaba antes</u>. Ahora ya no está permitido.
```

big y small también desaparecen en HTML5

```
<big>Esto no debe de hacerse</big>
<small>Y esto tampoco</small>
```

frame, frameset y noframes desaparecen en HTML5 y hay mucha controversia sobre la posibilidad de su uso en versiones anteriores. Sirven para dividir la superficie útil del navegador en diferentes zonas de forma que podamos cargar una página web diferente en cada una de ellas. Un ejemplo de su uso:

```
<html>
      <frameset cols="25%,*,25%">
            <frame name=”zona1” src="frame_1.htm" />
            <frame name=”zona2” src="frame_2.htm" />
            <frame name=”zona3” src="frame_3.htm" />
            <noframes>Tu navegador no soporta frames</noframes>
      </frameset>
</html>
```

3. CSS. SELECTORES BÁSICOS

Las hojas de estilo sirven para separar completamente el contenido del diseño de una página web, de tal forma que, si lo estructuramos correctamente, es posible cambiar totalmente el aspecto de nuestra web haciendo modificaciones en las hojas de estilo (css) sin tocar ni una sóla línea de su contenido (html). Para empezar, podemos hacer una pequeña demostración con la página de ejemplo que hemos creado en la anterior unidad. Si añadimos la siguiente línea en la sección de cabecera (**head**), el navegador dejará de usar la hoja de estilos por defecto y tomará una de las que pone a nuestra disposición la W3C como demostración:

```
<link href="http://www.w3.org/StyleSheets/Core/Traditional"
rel="stylesheet" type="text/css" />
```

Podemos sustituir la hoja **Traditional** por otra de las siguientes: **Midnight, Ultramarine, Chocolate, Oldstyle, Modernist, Steely** o **Swiss**. Cada una de ellas nos proporcionará un estilo diferente.

3.1. INTEGRACIÓN DE HTML CON CSS

Existen tres formas diferentes de integrar las hojas de estilo con HTML. La más rudimentaria consiste en usar el atributo **style** junto con cualquier etiqueta HTML. Por ejemplo:

```
<h1 style="color: red; font-family: Verdana, Arial;" >Título</h1>
```

La segunda forma sería introducir una nueva sección dentro de la cabecera (**head**) del documento HTML donde se insertarían los diferentes estilos:

```
<style type="text/css">
      h1 { color: red; font-family: Verdana, Arial; font-size: large; }
      p { color: gray; font-family: Times; font-size: medium; }
</style>
```

La tercera y ultima sería indicar, también en la sección **head** de nuestro documento HTML, la ruta a un archivo externo donde se incluirían los diferentes estilos. Ya hemos visto un ejemplo antes cuando tomábamos los ejemplos de la W3C. Uno propio podría ser así:

```
<link rel="stylesheet" type="text/css" href="estilos.css" media="all" />
```

En la sentencia anterior, en la etiqueta **link** podemos incluir un atributo denominado **media** que nos permitirá tener hojas de estilo diferentes según el medio donde se visualizará la página HTML. Los valores maś habituales de este atributo son **screen**, **print**, o **all,** que es el valor por defecto.

Dentro del archivo estilos.css escribiremos nuestros estilos con una sintaxis idéntica a la vista en el caso anterior:

```
h1 { color: red; font-family: Verdana, Arial; font-size: large; }
p { color: gray; font-family: Times; font-size: medium; }
```

Un documento HTML puede utilizar una de las tres formas anteriores, dos de ellas o las tres, incluso, a la hora de aplicar estilos. El navegador, en caso de conflicto, tomará siempre como bueno el estilo más cercano al contenido del documento. En caso de conflicto usando la misma forma de aplicar estilos el navegador tendrá en cuenta lo definido en la última ocurrencia aparecida.

Comentarios en hojas de estilo CSS

Los comentarios en las hojas de estilo (cuando estas se guardan en ficheros independientes) no se hacen de la misma forma que en HTML, sino que usamos los símbolos /* para abiri el comentario y */ para cerrarlo. Un comentario puede ocupar más de una línea. Un ejemplo:

```
/* Esto es un comentario
   y puede ocupar tantas líneas como deseemos
   hasta encontrar el símbolo de cierre de comentario */
```

3.2. SELECTORES BÁSICOS

Toda regla de formato en CSS está formadas por dos partes: selector y declaración. La declaración indica que formato hay que aplicar y el selector indica donde hay que aplicarlo. La declaración de la regla CSS va encerrada entre llaves y está formada por parejas **propiedad: valor;** terminadas siempre en punto y coma. Si observamos alguno de los ejemplos anteriores es fácil identificar cada uno de estos componentes y comprobar lo sencillo de la sintaxis.

Así como en HTML la palabra clave es la etiqueta, el corazón de CSS lo componen los selectores. La versión 2.1 de CSS tiene una docena de selectores diferentes pero es posible diseñar casi cualquier web con los cinco tipos básicos que veremos a continuación. Veremos los avanzados más adelante.

Selectores de tipo o de etiqueta

El selector de tipo o de etiqueta sirve para definir las propiedades de formato de todos los elementos de la página cuya etiqueta HTML coincida con el valor del selector. Los dos ejemplos que hemos visto en el punto 2 en los que se definía el formato para **h1** y **p** son ejemplos de selectores de este tipo. Podemos definir selectores de etiqueta para cualquier elemento HTML que esté dentro del cuerpo (**body**) de nuestra página HTML e, incluso, para el propio **body**:

```
body {font-family: Courier, Monospace; color: red; text-align: justify; }
```

Podemos aplicar estilos simultáneamente a más de una etiqueta separando los selectores con comas:

```
h1, h2, h3, h4, h5, h6 {color: blue; }
```

Observa que si cometes un error en la sintaxis, el navegador sencillamente no tendrá en cuenta la declaración de ese selector.

Selector universal

Existe un selector denominado universal que se simboliza con un asterisco. Se trata de una especie de comodín que permite seleccionar a cualquer elemento:

```
* {font-family: Courier, Monospace; color: red; text-align: justify; }
```

Cuidado: así aplicado sería similar, pero no igual, a aplicar la misma declaración sobre la etiqueta body, como hemos visto antes. Lo mejor para que aprecies las diferencias es que practiques sobre algunos ejemplos.

Colores y propiedades básicas del texto

Los colores en la declaración de cualquier selector de CSS se definen con dos propiedades: **color** (del texto o del elemento en si) y **background-color** (del fondo). Por ejemplo:

```
body {color: white; background-color: green; }
```

Los valores por defecto son **black** para el color del elemento y **transparent** para el fondo. Podemos aplicar estas propiedades en la declaración de cualquier selector de los ya vistos y los que veremos en adelante:

```
hr {color: red;}
```

¿Qué colores son válidos? Por un lado tenemos la posibilidad de usar una paleta de 17 colores básicos usando su nombre predefinido de entre los que aparecen en el siguiente mapa de colores:

Una segunda opción es indicar el color exacto que queremos mediante su valor

RGB, ya sea en hexadecimal, decimal o porcentual:

```
h1 { color: rgb(255, 20, 147); }
h2 { color: rgb(100%, 8%, 58%); }
h3 { color: #FF1493; }
```

Ciertos colores expresados en RGB hexadecimal se pueden presentar mediante una forma abreviada. Son aquellos en los que los tres componentes de color tienen sus cifras iguales, por ejemplo **#660099** puede expresarse también como **#609**. Un subconjunto formado por 216 de estos colores constituyen lo que se conoce como paleta de colores Web Safe y se usan cuando queremos garantizar que los colores que escojamos se verán exactamente igual en todos los dispositivos, incluidos aquellos que sólo pueden visualizar 256 colores (muchos ordenadores y, sobre todo, dispositivos móviles antiguos). Puedes consultar esta paleta en este enlace:

http://en.wikipedia.org/wiki/Web_colors#Web-safe_colors

Mediante la propiedad **font-family** elegimos la tipografía que queremos usar. Podemos indicar, el tipo de letra concreta que queremos o la familia tipográfica. Por ejemplo:

```
p {font-family: Arial; }
strong {font-family: serif; }
```

Si el tipo de letra consta de más de una palabra tenemos que indicarlo entre comillas dobles:

```
em {font-family: “Times New Roman”;}
```

Las familias tipográficas genéricas admitidas son **serif, sans-serif, monoespace, cursive,** o **fantasy**. Las tres primeros son las únicas que se usan habitualmente.

El tipo de letra así indicado de cualquiera de estas formas debe de estar instalado en el ordenador o ser accesible a través de Internet (y, en ese caso, indicar donde) para que se muestre correctamente, por eso es más seguro indicar una familia tipográfica o, mejor aún, una lista de preferencias que siempre acaba en una familia tipográfica. Por ejemplo:

```
h1 {font-family: Verdana, Helvetica, Arial sans-serif; }
p { font-family: Georgia, "Times New Roman", Times, serif; }
pre {font-family: "Courier New", Courier, monospace; }
```

Otra posibilidad es escoger una fuente disponible en Internet e indicar el lugar donde está disponible para que el navegador se la descargue y la use. El directorio más popular de fuentes es el que Google pone a nuestra disposición:

http://www.google.com/webfonts

El funcionamiento es sencillo: elegimos las fuentes que vamos a usar y Google nos proporciona la línea que tenemos que incluir en el head de nuestro HTML y la pareja propiedad:valor que debemos de usar en nuestras reglas CSS. Por ejemplo, para usar la fuente Gochi Hand debemos incluir esto en la sección head de nuestro HTML:

```
<link href="http://fonts.googleapis.com/css?family=Gochi+Hand"
rel="stylesheet" type="text/css">
```

Y nuestra regla CSS sería así:

```
h1 {font-family: "Gochi Hand", cursive;}
```

La propiedad **font-weight** nos permitirá cambiar el grosor del trazo de la fuente. Tenemos dos opciones para ello: especificar este mediante una palabra clave predefinida (**lighter, normal, bold** o **bolder**) o usar un valor numérico también de entre una lista de posibilidades (**100, 200, 300, 400, 500, 600, 700, 800** o **900**) donde el 400 se corresponde con normal y el 700 con **bold**. Sólo tenemos garantía de que funcionen **normal** y **bold.** El resto dependerá de la forma en que esté diseñada nuestra tipografía.

La propiedad **font-style** nos permite elegir entre **normal**, **italic** u **oblique.** La diferencia entre italic y oblique es que la **italic** debe de existir dentro del diseño tipografíco de la fuente (y si no no se muestra) mientras que la **oblique** se consigue forzando una inclinación de entre 8 y 12 grados la fuente regular. Siempre que exista esa posibildad obtendremos resultados de mejor calidad con **italic**.

font-variant (**desaparece en CSS3 y, por tanto, no es aconsejable su uso**) es una propiedad que nos permite dos únicas opciones: **normal** o **small-caps** (lo que en español conocemos como versales o versalitas).

```
p {font-weight: bold; font-style: italic; font-variant: small-caps; }
```

La justificación del texto la realizamos mediante la propiedad **text-align** cuyos valores posibles son **left, right, center** o **justify**. **text-align** con el valor **center** es el sustituto de la etiqueta **center** cuyo uso está desaconsejado.

```
h1 {text-align: center; }
```

Esta propiedad no sólo alinea el texto, sino que hace lo mismo con las imágenes que estén contenidas en un texto cuyo párrafo la tenga aplicada. Por ejemplo, si queremos una imagen alineada no podemos aplicarla sobre la etiqueta **img**, pero si sobre la etiqueta **p** del párrafo que contiene a esa imagen:

```
p {text-align: center; }
```

Y luego, meter la imagen dentro de un párrafo:

```
<p><img src="logotipo.jpg" alt="Logo de mi empresa" /></p>
```

Dos propiedades más antes de terminar este punto: **text-decoration** nos permite cinco efectos adicionales sobre nuestro texto: **none, underline, overline, line-throught** y **blink**. La propiedad **text-transform** tiene cuatro valores posibles: **none, capitalize, uppercase** y **lowercase**.

Existen más propiedades referentes al texto La propiedad **font-size,** por ejemplo**,** es la que nos permite cambiar el tamaño de las fuentes. También tenemos las relativas al interlineado y alguna más. Las veremos más adelante cuando hayamos hablado de las diferentes unidades de medida que podemos usar.

Selectores descendientes

Es una forma de seleccionar elementos que se encuentran dentro de otros elementos para aplicarles un estilo personalizado. Imaginemos, por ejemplo, que quiero aplicar un estilo particular a la etiqueta de cursivas (**em**) pero sólo cuando esta se aplica a un titular. Necesito usar un selector descendiente:

```
h1 em {color: red; }
```

El estilo se aplicará a todas las etiquetas **em** dentro de un elemento **h1** sin importar el nivel de profundidad. Es decir, en el siguiente fragmento HTML también se aplicarían:

```
<h1><a id="INICIO">Titular de la <em>pagina</em></a></h1>
```

El selector descendiente debe de estar formado por, al menos, dos selectores pero podría tener más si así lo precisamos:

```
p strong em {font-variant: small-caps; }
```

El estilo anterior se aplicaría a las cursivas (**em**), dentro de una etiqueta de negrita (**strong**) y, a su vez, dentro de un párrafo (**p**).

Podemos combinar este tipo de selectores con el selector universal para realizar restricciones. Por ejemplo, tengamos en cuenta las siguientes dos variantes de selectores y los dos fragmentos de código HTML a continuación:

```
p a { color: green; }
p * a { color: green; }

<p><a id="INICIO">Enlace</a></p>
<p><strong><a href="#">Enlace</a></strong></p>
```

Si usamos la primera regla CSS se colorearán en verde las dos líneas, mientras que si usamos la segunda regla, sólo se coloreará la segunda.

Selectores de clase

Los selectores que hemos visto hasta ahora fallan en una cosa: no son flexibles cuando queremos aplicar un estilo diferenciado a algunos párrafos, pero no a todos ellos. Una de las soluciones para esto es usar los selectores de clase. Para aplicarlos, lo primero que tenemos que hacer es usar el atributo **class** en las etiquetas HTML correspondientes a los párrafos cuyos estilos queremos diferenciar:

```
<p class="verde">Este párrafo debe de ir en verde.</p>
<p>Este párrafo va en color normal.</p>
```

La regla CSS se especifica anteponiendo un punto al valor indicado en el atributo class:

```
.verde {color: green; }
```

La regla CSS se aplicará a cualquier etiqueta HTML que incluya un atributo **class** cuyo valor sea verde, y no sólo a las etiquetas **p**. Es decir, sería perfectamente válido que, además de los párrafos anteriores, tuvieramos algo así en nuestro HTML:

```
<p> Si aplicamos el atributo class con valor verde a <strong
class="verde">estas negritas</strong> o a <em class="verde">estas
cursivas</em> también aparecen en verde.</p>
```

Si quisiéramos restringir el uso del selector de clase para que sólo sea válido en combinación de una determinada etiqueta, podemos combinarlo con el selector de tipo o etiqueta:

```
p.verde {color: green; }
```

Ojo, no debemos de confundir las tres diferentes formas de combinación de selectores que hemos visto hasta ahora:

```
p, .verde {color: green; }
p .verde  {color: green; }
p.verde   {color: green; }
```

En el primer caso la regla se aplica a cualquier etiqueta **p** o a cualquier etiqueta que tenga un atributo **class** con valor verde. En el segundo caso se aplica a cualquier etiqueta con un atributo **class** con valor verde que se encuentre dentro de un elemento de tipo **p**. En el último caso, sólo se aplica en las etiquetas de tipo **p** que tengan un atributo de tipo **class** con valor igual a verde.

Podemos aplicar también el formato de más de un selector de clase al mismo elemento HTML. Imaginemos las siguientes reglas CSS:

```
.codigo {font-family: monoespace; color: rojo; }
.versalita {font-variant: small-caps;}
```

Y en nuestra página HTML:

```
<p class="codigo versalita">Párrafo en rojo con fuentes monoespaciadas y
usando versalitas</p>
```

Por último, podemos usar también los selectores de clase múltiple. La regla CSS se expresa así:

```
.codigo.versalita {font-family: monoespace; color: blue; font-variant:
small-caps;}
```

Si incluimos esta regla en nuestro ejemplo anterior, tendrá preferencia sobre las reglas individuales y nuestro párrafo aparecerá en azul en lugar de en rojo.

Selectores de ID

El selector de **id** es muy similar al de clase. La principal diferencia es que se debería de usar sólo cuando es preciso aplicar estilos a un elemento único en la página. En ese caso usaremos el atributo **id** en lugar de **class** en la etiqueta HTML

```
<p id="destacado">Parrafo destacado en negritas y color azul.</p>
```

y el simbolo almohadilla (#) en lugar del punto en la definición de la regla CSS.

```
#destacado {font-weight: bold; color: blue; }
```

Una nota muy importante: la restricción de usar la etiqueta **id** en un único elemento es algo que nosotros debemos de imponernos pero que, si la obviamos y la ponemos en más de un sitio el navegador no se quejará e, incluso, aplicará correctamente los estilos indicados en todas las ocurrencias de la misma. No obstante, no podríamos considerar como correcto de forma estricta el código así escrito y, por tanto, deberíamos de evitarlo.

Casi todo lo demás dicho en referencia a los selectores de clase (la forma de restringir su uso, de combinarlo con otros selectores, etc) es aplicable a los selectores de id con una salvedad: no existen selectores de id múltiple y no podemos poner dos id diferentes a una misma etiqueta HTML al igual que hacíamos con el selector de clase.

3.3. COMBINANDO SELECTORES

Todos los selectores aquí vistos pueden combinarse entre si. Algunos ejemplos ya los hemos visto en este documento. Pero podemos complicarlo cuanto queramos o necesitemos. Veamos algunos ejemplos:

```
.aviso .especial { color: red; background-color: yellow; }
```

El anterior selector solamente selecciona aquellos elementos con un **`class="especial"`** que se encuentren dentro de cualquier elemento con un

class="aviso", es decir, estamos aplicando simultaneamente selectores de clase y selectores descendientes.

```
p.aviso strong.especial { ... }
```

Ahora, el selector solamente selecciona aquellos elementos de tipo **** con un atributo **class="especial"** que estén dentro de cualquier elemento de tipo **<p>** que tenga un atributo **class="aviso"**. O sea, estamos aplicando selectores descendientes junto con selectores de clase que tienen su uso restringido a ciertas etiquetas.

```
ul#menu li.destacado a#inicio { ... }
```

El anterior selector hace referencia al enlace con un atributo **id="inicio"** que se encuentra dentro de un elemento de tipo **<li>** con un atributo **class="destacado"**, que forma parte de una lista **<ul>** con un atributo **id="menu"**. De nuevo selectores descendientes pero en esta ocasión con tres niveles y en cada uno de ellos tenemos un selector de clase o de ID con uso restringido.

4. UNIDADES DE MEDIDA Y OTRAS PROPIEDADES

En este capítulo veremos algunos de los elementos básicos de CSS que aún nos restan. Estudiaremos las diferentes unidades de medida que podemos utilizar, algunas propiedades más relativas al texto, introduciremos las dos etiquetas **div** y **span** y veremos los conceptos de herencia y especificidad.

Unidades de medida

CSS tiene un amplio y flexible conjunto de formas de expresar tamaños y medidas. Manejarlas correctamente es necesario para expresar el tamaño de las fuentes que usamos y para otras características esenciales en un buen diseño: interlineado, márgenes, separaciones, etc.

Para experimentar con ellas presentaremos una nueva propiedad: **font-size** que nos permite definir el tamaño de la tipografía que estamos usando. Algunas de las formas de especificar medidas que veremos aquí son sólo válidas para los tipos de letras mientras que otras son más generales y nos servirán para cualquier otra cosa.

Unidades de medida específicas de las tipografías

La forma más sencilla es expresar esta medida como una palabra clave que exprese un valor fijo absoluto. Los valores permitidos son **xx-small, x-small, small, medium, large, x-large y xx-large.**

```
h1 {font-size: xx-large;}
p#piedepagina {font-size: x-small;}
```

La segunda forma de expresar el tamaño de la tipografía sería mediante una palabra clave que expresa un tamaño relativo. Dos son las palabras claves permitidas en este caso: **larger** y **smaller**. El primer valor elevará la fuente sobre lo que le correspondería por defecto mientras que el segundo lo reducirá.

```
p {font-size: medium;}
p.grande {font-size: larger;}
```

Unidades absolutas

Podemos expresar medidas absolutas en pulgadas (**in**), centímetros (**cm**), milímetros (**mm**), puntos (**pt**) o picas (**pc**). Estas dos últimas son medidas específicas de las imprentas y medios gráficos. Un punto equivale a 1/72 pulgadas (aproximadamente 0,35 milímetros) y una pica equivale a 12 puntos (alrededor de 4,23 milímetros). El punto también es conocido porque es la medida que suele usarse en los editores de texto para elegir el tamaño de una fuente.

Cuatro notas a tener en cuenta en todas ellas:

- Para separar las cifras decimales usaremos el punto que es la norma en las medidas anglosajonas.
- Si la parte entera de una medida es cero se puede suprimir.
- Si la medida es cero no se tiene que poner unidad de medida. Un cero es 0

y basta.

- Entre la magnitud y la unidad de medida no debe de haber nunca espacios en blanco.

Las siguientes reglas serían correctas:

```
h1 {font-size: .5in;}
p {width: 25cm; font-size: 8.5pt;}
p.destacado {width: 100mm; }
td {width: 10pc; }
```

Unidades relativas

Las unidades relativas son mucho más flexibles que las absolutas. Además, es altamente recomendable usarlas siempre que se pueda porque ante la gran diversidad de dispositivos que se pueden usar a la hora de ver una página web (portátiles, grandes televisores de plasma, tablets, teléfonos móviles...), estas son las que mejor se adaptan a cualquier situación.

Los píxeles son la medida relativa más conocida. Se trata de una medida relativa a las dimensiones del dispositivo donde estamos viendo la página web. Un ejemplo:

```
hr {width: 500px;}
```

em (ojo: no confundirlo con la etiqueta **em** de HTML) y **ex** son medidas relativas a las dimensiones de la tipografía que estamos usando. Son medidas muy conocidas por los profesionales de la tipografía. **1em** representa, aproximadamente, el ancho de la letra M mayúscula que estamos usando teniendo en cuenta tanto la tipografía como el tamaño de la misma: no es lo mismo una m si estamos usando una Arial Narrow que si estamos usando una Courier New. Igualmente, para cualquiera de ambas el tamaño varía si estamos usando una fuente a 12 puntos o a 18. **1ex** equivale a la altura de una letra x minúscula. Aunque no se trata de una regla exacta, **1em** suele aproximarse como el tamaño de la fuente que estamos usando (es decir, si estamos usando una fuente a 12 puntos, **1em** equivale a 12 puntos).

```
body {font-size: .9em;}
```

Si consideramos cierta la aproximación que hemos dicho antes, podemos decir que una fuente con un tamaño de **.9em** es, aproximadamente, el 90% de la fuente normal que debería de tener esa etiqueta. Una con **1.5em** sería del 150%

```
p {font-size: 1.5ex;}
```

También se suele aproximar, aunque esto es bastante más irregular, que **1ex** equivale aproximadamente a **0.5 em**

Aunque a primera vista son medidas que parece que sólo tienen sentido aplicadas a tipos de letra, podemos usarlas sobre cualquier elemento de nuestra web:

```
hr {width: 1em;}
```

Y una nota importante: la "referencia" sobre la que se calcula el tamaño relativo de un elemento cuando usamos estas medidas es siempre es el tamaño de letra de su elemento "padre", es decir, del elemento en el que se encuentra. Si el elemento no se encuentra dentro de ningún otro elemento, la referencia es el tamaño de letra del elemento **body**. Si no se indica de forma explícita un valor para el tamaño de letra del elemento **body**, la referencia es el tamaño de letra por defecto del navegador. Lo podemos ver con este ejemplo:

```
body { font-size: 10px; }
h1 { font-size: 1em; font-weight: normal;}
```

Al aplicar esto a nuestra web, definimos el tipo de letra base a 10 píxeles y el tamaño de los titulares de primer orden a **1em** que, como hemos dicho, equivale aproximadamente al tamaño de la letra base, es decir, que no habría diferencia entre los titulares de primer orden y el texto normal. Otro ejemplo:

```
body { font-size: 12px; }
strong { font-size: 1.2em;}
```

Con esto, todas las negritas serían, aproximadamente, un 20% más grandes sobre el tamaño del texto en el que se colocan.

Porcentajes

El porcentaje es otra forma de expresar una medida usada ampliamente en CSS que, además, también se considera una unidad relativa. Se pueden usar para expresar el tamaño de una fuente:

```
p {font-size: medium;}
p.grande {font-size: 200%;}
```

Cuando aplicamos un porcentaje como una unidad de medida general, este se refiere al espacio que el objeto debería de ocupar. Por ejemplo, una linea horizontal ocupa normalmente el ancho de la pantalla completa. Si aplicamos la siguiente regla CSS ocupará solamente la mitad:

```
hr {width: 50%;}
```

La W3C recomienda encarecidamente usar píxeles y porcentajes como unidades de medida para especificar el tamaño de los diferentes objetos y **em** como unidad para especificar el tamaño de las fuentes. Cuando se prepara un documento HTML cuya finalidad es tener salida impresa en papel y no en un dispositivo óptico, es mejor el uso de puntos y picas.

4.1. MÁS PROPIEDADES PARA EL TEXTO

Veremos a continuación algunas propiedades aplicables al texto de nuestros documentos que aún no conocemos.

La propiedad **line-height** ajusta el interlineado del texto. El valor **normal** es el que

debería de tener por defecto. Podemos variar este poniendo un porcentaje, un número o una medida en valor absoluto o relativo. Las siguientes reglas CSS serían prácticamente equivalentes:

```
p { line-height: 1.5;}
p { line-height: 1.5em;}
p { line-height: 150%;}
```

La propiedad **text-indent** nos permite que la primera línea de cada párrafo aparezca desplazada hacía la derecha respecto al resto del párrafo. Se puede usar una medida (po ejemplo, 2.5em) o un porcentaje relativo al ancho total de la línea del párrafo (un 50% haría que la primera línea empezara justo en su mitad). Admite valores negativos que harían que la primera línea del párrafo empiece más a la izquierda del mismo.

letter-spacing y **word-spacing** son dos propiedades que nos sirven para controlar el espacio en blanco que separa cada dos letras o cada dos palabras, respectivamente. Los posibles valores son **normal** o una medida, representando esta el espacio adicional que se suma o resta (admite valores negativos) al espaciado por defecto.

```
h1.juntito { letter-spacing: -.2em; }
p.separado { word-spacing: .5em; }
```

La propiedad **white-space** controla la forma en que se trataran los espacios en blanco, los saltos de línea y el ajuste de las mismas. Los posibles valores son **normal, pre, nowrap, pre-wrap** y **pre-line**. **normal** y **pre** son las más usadas. La primera representa el comportamiento normal del navegador (sólo se respeta el primer espacio en blanco y ningún salto de línea pero estas se ajustan) y **pre** el que obtendríamos usando la etique **pre** de HTML (se respetan todos los espacios en blanco y saltos de línea que hagamos pero las líneas no se ajustan y si son muy largas desbordan el espacio asignado). Las tres restantes son variantes de estas tres posibilidades. **no-wrap** seria como normal pero sin ajustar las líneas, **pre-wrap** sería como **pre** pero ajustando las líneas y, por último, **pre-line** sería como **pre** pero sin respetar los espacios en blanco.

La propiedad **vertical-align** nos ayuda a alinear elementos de diferentes alturas situados en la misma línea. La mejor forma de verlo es colocando una pequeña imagen junto a una línea de texto y hacer pruebas con los diferentes valores posibles. Supongamos la siguiente línea de HTML:

```
<p><img src="http://moodle.uponaday.net/logo.jpg" alt="FP" /> IES Rey
Fernando IV</p>
```

Como observamos, texto e imagen se alinean por la base que es el valor por defecto. Sería equivalente a usar el valor **baseline** para esta propiedad. Creemos ahora la siguientes reglas CSS:

```
img {border:1px solid; vertical-align: text-bottom;}
```

La diferencia entre el valor **text-bottom** y **baseline** es prácticamente inapreciable: apenas veremos que la línea de texto asciende un poco. El primero alinea la parte inferior de la imagen con la parte más baja de las letras, mientras que **baseline** realiza la alineación con la línea base de escritura de estas. **middle** es uno de los valores más corrientes y alinea el centro de la imagen con el centro de la línea de texto, mientras que **text-top** alinearía la parte superior de ambos elementos. **top** y **bottom** son dos atributos similares a, respectivamente, **text-top** y **text-bottom** pero más orientados a cuando queremos alinear diversas imágenes de diferentes alturas. **sub** y **super** son dos valores raramente usados. Alínearian la base de la imagen con la línea base de los subíndices o de los superíndices del texto, respectivamente. Por último, podemos poner una medida o un porcentaje. La medida puede ser positiva o negativa y representa la distancia desde la base de la imagen con la base de la línea de texto (medidas negativas harían descender la línea de texto). El porcentaje tiene el mismo efecto correspondiendo el 100% con el alto de la línea de texto . Con una tipografía de 12 puntos, un desplazamiento de -100% o de -12pt debería de ser prácticamente equivalente.

La propiedad **text-decoration** permite los valores **none** (por defecto), **`underline`, `overline`, `line-through`** o **`blink`.**

La propiedad **text-transform** admite los valores **none** (por defecto), **capitalize, uppercase** y **lowercase.**

4.2. APLICAR ESTILOS A LAS LISTAS

Los estilos se aplican a las listas mediante tres propiedades: **list-style-type, list-style-image** y **list-style-position.**

list-style-type define el tipo de "encabezado" de cada uno de los items de la lista. **circle**, **disc**, **square** o **none** son algunos de los más usados, pero también existen otros muchos que puedes consultar aquí:

http://www.w3schools.com/cssref/pr_list-style-type.asp

list-style-image nos permite usar una imagen de nuestra elección como encabezado de cada item en lugar de uno de los predefinidos con la propiedad anterior. Como siempre en estos casos hay que especificar la URL de acceso a la imagen. Si se especifica junto con **list-style-type**, esta segunda sólo se tendrá en cuenta si la imagen que especificamos no es accesible.

list-style-position es una de esas propiedades de "ajuste fino" del formato y se usa para especificiar si el marcador de principio del item debería de aparecer dentro (**inside**) o fuera a la izquierda (**outside**) de la caja en la que debería de aparecer cada item de la lista. El valor por defecto es **outside**.

Existe una propiedad abreviada llamada **list-style** que permite definir a las tres anteriores en una sóla regla. Las siguientes declaraciones, por tanto, serían

equivalentes:

```
ul.milista { list-style-type: square;
             list-style-image: url(redicon.jpg);
             list-style-position: inside; }

ul.milista2 { list-style: square url(redicon.jpg) inside; }
```

Existen muchas otras propiedades CSS que admiten reglas abreviadas que engloban a varias otras relacionadas. Algunas las iremos viendo más adelante y otras son fáciles de identificar en los ejemplos.

4.3. DOS NUEVAS ETIQUETAS HTML: div y span

div y **span** son dos nuevas etiquetas HTML cuyo objetivo es no hacer nada por defecto. Puedes hacer la prueba de colocarlas en un documento HTML y podrás comprobarlo por ti mismo. El objeto de ambas es dar estructura a nuestro documento. Se usan conjuntamente con las hojas de estilo y, particularmente, con los selectores de clase e ID.

div es una etiqueta de bloque ideada para estructurar de forma lógica nuestra página. Sería similar a un párrafo pero, mucho más flexible y, además, por defecto no conlleva la aplicación de ningún estilo.

La etiqueta **span** es muy similar pero se trata de una etiqueta de línea y se usa para agrupar elementos a los que aplicar estilos diferenciados.

Veamos el body del siguiente HTML:

```
<body>
   <div class="verde">
      <h1><a id="INICIO">Titular de la <strong>pagina</strong></a></h1>
      <p>Si meliora dies... bra, bra...</p>
      <hr />
      <p>Otro párrafo... bra, bra, bra...</p>
   </div>
   <p>Este bloque ya está fuera del efecto del anterior bloque div y, por
   tanto, su fondo será blanco.</p>
</body>
```

Y lo acompañamos de esta regla CSS:

```
.verde { background-color: green;}
```

Observamos que, efectivamente, mediante **div** podemos agrupar diferentes elementos de bloque y aplicar propiedades de forma conjunta a todos ellos. Podríamos conseguir el mismo efecto añadiendo **class="verde"** como atributo a todos los elementos de bloque incluidos en el elemento **div**, pero sería más pesado, menos claro y podría dar lugar a más errores e inconsistencias.

Imaginemos ahora que queremos marcar de forma especial con otro estilo y otro

tamaño, la primera letra del primer párrafo de un libro de cuentos que queremos publicar en la web. Usaremos para ello una fuente de Google Fonts, así que incluimos esto en el head de nuestro HTML:

```
<link href='http://fonts.googleapis.com/css?family=Rock+Salt'
rel='stylesheet' type='text/css'>
```

Luego, en nuestras hojas de estilo, definimos lo siguiente:

```
.primeraletra {font-family: 'Rock Salt', cursive; font-size: 500%; font-
weight: bolder; }
```

Y los párrafos de nuestra web irían así:

```
<p><span class="primeraletra">S</span>i meliora dies, ut vina</p>
<p><span class="primeraletra">V</span>Veteresne poetas, an quos et
praesens et postera respuat aetas?</p>
```

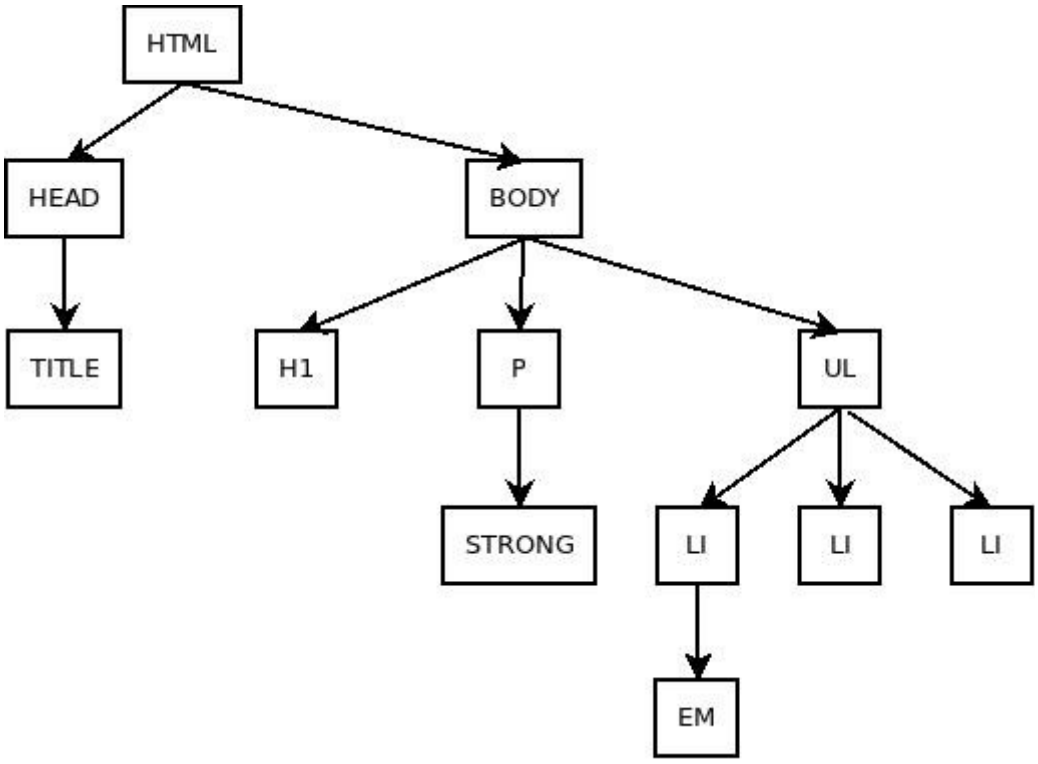
(**NOTA:** Más adelante veremos que existe otra forma más profesional de hacer esto mismo mediante los **pseudo-elementos**).

4.4. HERENCIA

Podemos expresar cualquier documento HTML mediante una estructura en forma de árbol. Por ejemplo, tengamos el siguiente HTML sencillo (en el que, intencionadamente, hemos suprimido algunas de las etiquetas y atributos para simplificarlo):

```
<html>
   <head>
      <title>Mi web</title>
   </head>
   <body>
      <h1>La web del profe de Lenguaje de Marcas</h1>
      <p>José María Morales es el <strong>profesor</strong> de este año
      de la asignatura y, entre otras cosas, vamos a ver con él:</p>
      <ul>
         <li>HTML y <em>XHTML</em></li>
         <li>CSS</li>
         <li>XML</li>
      </ul>
   </body>
</html>
```

Podríamos ver la estructura de nuestro documento HTML esquematizada como el siguiente árbol:



Pues bien, la herencia de CSS hace que las propiedades de los diferentes elementos se traslade siempre hacia abajo a través del árbol salvo que dichas propiedades sean reescritas, bien por los estilos por defecto, bien por una regla apropiada que hayamos introducido nosotros.

Por ejemplo, el elemento con la etiqueta **strong** aparecerá en negrita, no porque nosotros lo hayamos especificado así, sino porque así está definido por defecto y aunque definamos la propiedad f**ont-weight: normal** para el selector **body** seguirán siendo negritas.

Sin embargo, si definimos la propiedad **color: orange;** y **background-color: black** en el selector **body**, todo aparecerá con fondo negro y letras naranjas.

Si nuestra página web sencilla llevara un enlace (etiqueta **a**) el fondo si aparecería negro, pero el color sería azul porque esa etiqueta si tiene la propiedad color sobreescrita.

En definitiva: las propiedades de estilo se heredan siempre hacía abajo en el árbol salvo que dicha propiedad sea rescrita, bien esplicitamente, bien con una regla por defecto.

4.5. ESPECIFICIDAD

A nadie se le escapa a estas alturas que a medida que vayamos haciendo más complejas nuestras hojas de estilo, más dificil será saber en algunas ocasiones que regla prevalecerá sobre otra. Ya hemos dicho que, en igualdad de condiciones (por ejemplo, dos reglas que hacen referencia a estilos contradictorios para la

etiqueta **p**), se le dará preferencia a la que está más cercana al contenido si las reglas se encuentran en diferentes sitios (las más prioritarias serían las consignadas a través del atributo **style** en el propio HTML, las siguientes las que escribimos directamente en la sección **head** del mismo y, por último, aquellas que van en un fichero externo). Ante dos reglas que estén en el mismo sitio (por ejemplo, ambas en dos líneas diferentes de un mismo archivo externo CSS) se le dará preferencia a la que esté escrita en último lugar.

Estas reglas nos ayudan a solucionar los casos más simples de conflicto pero, ¿que ocurre cuando el problema lo plantean dos reglas totalmente diferentes que se deberían de aplicar al mismo elemento? Para eso surge lo que llamamos especificidad.

Imaginemos que nuestro CSS tiene las siguientes reglas:

```
ul ol li.alerta {color: red; }
.alerta{color: blue;}
li.alerta{color: yellow; background-color: yellow;}
ul .alerta{color: green;}
```

Y en nuestro HTML aparece algo como esto:

```
<ul>
   <li>Uno
      <ol>
         <li>Dos</li>
         <li class="alerta">TRES</li>
      </ol>
   </li>
   <li>Cuatro</li>
</ul>
```

¿Qué regla o reglas se aplicarían al elemento cuyo atributo **class** corresponde con **alerta**, correspondiendo este con las cuatro reglas que aparecen en el CSS? Puedes probar que cualquiera de ellas se corresponde con el elemento poniéndolas una a una. Si las pones todas juntas, independientemente del orden en que estén observarás que el texto aparece con color rojo y el fondo con color amarillo, es decir, se está aplicando la regla 1 y, parcialmente, la regla 3.

El hecho de que el fondo aparezca amarillo es bien sencillo: aparece así porque, no entra en conflicto con ninguna propiedad de otra regla. De hecho, podemos ver que si movemos esa propiedad a cualquiera de las otras reglas el fondo seguirá apareciendo amarillo. Esto es porque en realidad las estamos aplicando todas y sólo en el caso de que exista un conflicto entre propiedades en diferentes reglas tenemos que decidir cual es la que hay que usar. Esto se arbitra calculando la especificidad de la regla.

La forma más ampliamente aceptada de calcular la especificidad es mediante un conjunto de cuatro números separados por comas (por ejemplo 0,1,0,5). Los números a la izquierda tienen un rango mayor que a la derecha y, por tanto, definen a un selector más específico y que por tanto prevalece. Ante empate en un nivel, miramos en el siguiente. En caso de empate en todos los niveles

prevalece la regla escrita en última posición.

EJEMPLOS: entre una regla con valor 0,0,1,0 y otra con valor 0,0,0,13 prevalece la primera y entre una regla con valor 0,1,3,1 y otra con valor 0,1,3,2 prevalecería la segunda.

¿Cómo se calculan estos números? El primero es fácil y se usa cuando el estilo se aplica en el propio HTML mediante el atributo style. Este número sólo puede valer 0 o 1 (puesto que no podemos duplicar atributos en una misma etiqueta) y tiene siempre la máxima prioridad. Los otros tres se calculan sumando 1 según los tipos de selectores que aparezcan en la regla:

- El segundo número se calcula sumando uno por cada selector de id que aparezca en la regla
- El tercer número se calcula sumando 1 por cada selector de clase, selector de atributo o pseudo-clase que aparezca en la regla (los dos últimos los veremos en el capítulo sobre CSS avanzado)
- El cuarto número, el de menor prevalencia, se calcula sumando 1 por cada selector de etiqueta o pseudo-elemento (este último lo veremos también en CSS avanzado) que aparezca en la regla.

Así, si calculamos la especificidad de las cuatro reglas anteriores:

```
ul ol li#alerta {color: red; }                          /* 0,1,0,3 */
.alerta{color: blue;}                                   /* 0,0,1,0 */
li.alerta{color: yellow; background-color: yellow;}     /* 0,0,1,1 */
ul .alerta{color: green;}                               /* 0,0,1,1 */
```

Lo que nos confirma que, efectivamente, el color del texto debería de salir rojo independientemente del orden en que se situen las reglas. Si eliminamos esa regla, el texto saldrá en verde, pero puesto que hay un empate entre esta regla y la que dicta que debería de salir amarillo y prevalece la escrita en último lugar.

Los selectores de hijos y hermanos (también los veremos más adelante) no aportan una mayor especificidad a las reglas.

La clausula !important

La clausula **!important** se puede aplicar a cualquier regla aumentando así su nivel de prioridad de forma que prevalezca sobre definiciones posteriores del mismo selector e, incluso, sobre las valoraciones de especificidad. Ante las siguientes reglas, un texto etiquetado como **strong**, dentro de un párrafo (selector descendiente) y con el id minegrilla aparecería con color rojo:

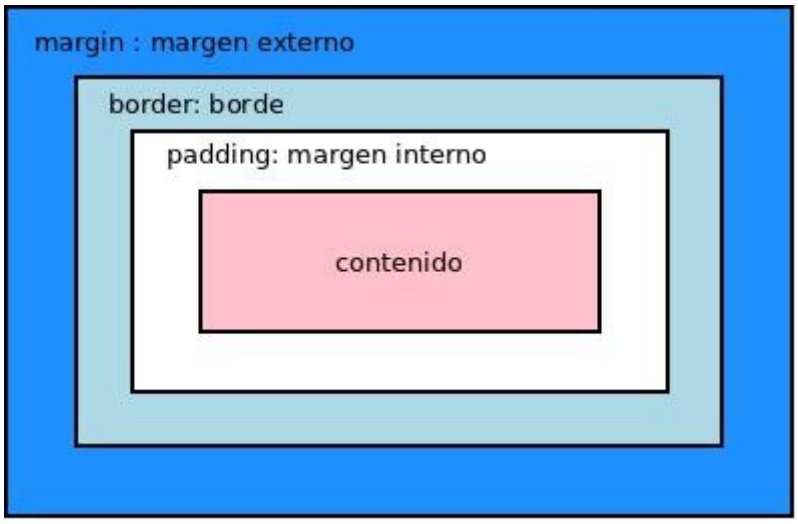
```
strong {color: red !important;}
p strong#minegrilla {color: blue; }
```

5. EL MODELO DE CAJAS

El modelo de cajas se trata, posiblemente, del concepto más importante relacionado con las hojas de estilo CSS. Como ya hemos dicho antes, todo en HTML está formado por cajas rectangulares que contienen a los diferentes elementos. Mediante las reglas que veremos a continuación podremos cambiar el aspecto y comportamiento de las mismas.

5.1. EL CONCEPTO DE CAJA

HTML y CSS definen una caja como una zona rectangular constituida por los siguientes elementos:



Tenemos, además, control total y de forma individual de cada una de las dimensiones de las tres capas externas. El contenido suele ser dinámico (en una o en sus dos dimensiones) o especificarse con dos valores: ancho y alto. Como unidades de medida tenemos todas las que ya conocemos

La principal confusión que suelen tener los "primerizos" es la forma de calcular las dimensiones totales de la caja. Supongamos una caja cuyo contenido tiene un ancho de 200 píxeles, el margen interior es de 20 píxeles, el borde de 5 y el margen externo de 25 píxeles ¿Cuál es el ancho total de la caja?

Haz las cuentas: son 300 píxeles ¿verdad?. Es mucho mejor verlo antes de echarle un vistazo a la sintaxis que es aún más engañosa:

```
.ejemplo { width: 200px;
    height: 100px;
    border: 5px solid blue;
    padding: 20px;
    margin: 25px;
    background-color: yellow; }
```

Y aquí, una nota por si alguien aún no se ha dado cuenta de la importancia del DOCTYPE: El internet explorer ha considerado durante mucho tiempo que el borde interior y los márgenes forman parte del contenido pese a que la

especificiación de la W3C jamás ha dicho esto. Aún con Explorer 9, si nuestro HTML no especifica un DOCTYPE, nos visualizará la página en modo de compatibilidad (quirk mode) y nos dibujará una caja con un tamaño de 250 píxeles, de forma que el contenido útil será de sólo 150 píxeles.

div es la caja más versatil

Aunque muchos elementos de bloque de HTML son susceptibles de ser usados como cajas, la que por regla general usaremos como tal es la etiqueta **div**. Es la más versatil ya que no tiene estilos propios salvo los que definamos nosotros.

5.2. PROPIEDADES DE LA CAJA

Las dimensiones del contenido

Las dimensiones del contenido de la caja se delimitan con las propiedades **width** y **height.** Ambas puede tener por valor un porcentaje, una medida o la palabra **auto** que haría que su tamaño se ajustara lo mejor posible al contenido real de la misma. Los valores por defecto de estas dos propiedades son **width: 100% y height: auto**;

Cuando llevan un porcentaje, este se aplica sobre el tamaño de la caja dentro de la que, a su vez, se encuentran (su elemento padre o contenedor). Por ejemplo, si una caja tiene por ancho el 50% y no está dentro de ninguna otra, esto se aplica sobre el tamaño total de la superficie útil del navegador. Si dentro de esta metemos otra también con un ancho del 50%, tendrá un ancho de la mitad de la caja donde se encuentra, es decir, un 25% de la superficie útil del navegador. Cambiemos el ancho de la regla **.ejemplo** vista antes por 50% y pongamos esto en nuestro HTML:

```
<div class="ejemplo">Hola caja 1
      <p class="ejemplo">Hola caja 2</div>
</div>
```

NOTA IMPORTANTE: La altura sólo puede valer un porcentaje si su padre o elemento contenedor tiene definida una altura con un valor concreto. En caso contrario se ignorará y tomará el valor **auto**.

Márgenes interior y exterior

Alternativamente tenemos las propiedades **max-height, min-height, max-width** y **min-width** para fijar las dimensiones máximas y mínimas. En este caso no podemos usar la palabra **auto** sino sólo porcentajes o medidas concretas.

Los márgenes interiores y exteriores se controlan con las propiedades **padding** y **marging** respectivamente y tenemos varias posibilidades. Podemos poner **auto** (para dejarlo al arbitrio de lo predefinido por el navegador), un valor exacto o un porcentaje que estaría referido al tamaño del elemento padre. Además, también podemos especificar un margen igual para cada una de las cuatro dimensiones de la caja o especificar que valor concreto queremos para ellas. Las propiedades

para esto son **margin-top, margin-right, margin-bottom** y **margin-left** para el margen externo y **padding-top, padding-right, padding-bottom** y **padding-left** para el interno. Una última posibilidad, más compacta, es usar la propiedad **margin** con uno, dos, tres o cuatro valores:

```
/* 20px para cada margen */
margin: 20px;

/* 20px para superior e inferior y 10px para derecho e izquierdo */
margin: 20px 10px;

/* 20px para el superior 10px para los laterales y 30 para el inferior */
margin: 20px 10px 30px;

/* los cuatro en el sentido de las agujas del reloj: 20px superior, 10px
derecho, 30px inferior y 10px izquierdo */
margin: 20px 10px 30px 10px;
```

Un truco habitual para mostrar una caja centrada con respecto al elemento donde está contenida es hacer que los márgenes a izquierda y derecha sean iguales y calcular el tamaño exacto que deben tener. Esto es relativamente fácil cuando trabajamos únicamente con porcentajes, pero no siempre es posible. El navegador lo hará solo estableciendo los márgenes laterales a **auto**.

Existe una peculiaridad con respecto a los márgenes exteriores de las cajas. Así como los márgenes derecho e izquierdo de cajas contiguas se suman, los superior e inferior de cajas adyacentes se solapan. Imaginemos, por ejemplo, dos cajas con margen exterior de 20 píxeles en cada una de sus cuatro dimensiones. Si las situamos una junto a otra tendremos 40 píxeles de separación pero si ponemos una encima de la otra sólo estarán separadas por 20 píxeles.

Para la propiedad **padding** la sintaxis es similar. Además, el margen exterior permite valores negativos (¡experimenta con los resultados!) mientras que el interior traducirá un valor negativo por cero.

Y un último detalle: el **body** de un documento HTML es también una caja y puede tener predefinidos valores para el **margin** y el **padding** de forma que nos quede un pequeño borde sin usar en la superficie del navegador. Si no queremos que esto sea así podemos usar esta regla:

```
body {margin: 0px; padding: 0px; }
```

El borde de la caja

El borde de la caja se define mediante tres propiedades: color, grosor y estilo: **border-color, border-width** y **border-style**, aunque luego veremos que tenemos más posibilidades tanto para simplificar la definición como para particularizarla con mucho más detalle como hemos hecho con los márgenes.

La forma abreviada de especificar las características del borde de la caja es esta:

```
border: red double 5px;
```

Donde el primer parámetro es el color (de cualquiera de las formas que hemos aprendido a definirlo), el segundo el estilo (siendo válidos los valores **solid, dashed, dotted, double, groove, ridge, inset, outset, hidden** o **none**) y el tercero es el ancho del borde que puede ser un valor de longitud (nunca un porcentaje) o las palabras clave **thin, medium** o **thick**.

Este formato abreviado sólo es válido cuando las cuatro dimensiones del borde son iguales. Los parámetros pueden ir en cualquier orden pero deberían de ir los tres (aunque la mayoría de los navegadores nos admitirían uno o dos solamente). Los valores por defecto si prescindimos de alguno son **black** para el color, **medium** para el grosor y **none** para el estilo.

La diferencia entre **none** y **hidden** es que aunque en ninguno de los casos el borde es visible, en el segundo influye respecto al lugar donde se dibujaría un elemento adyacente mientras que el primero se considera con un ancho de 0px.

Podemos especificar cada una de estas tres propiedades por separado con los atributos **border-color**, **border-style** y **border-width.** Asimismo, podemos cambiar los valores de cada una de las cuatro dimensiones en cada uno de ellos usando atributos específicos (**border-top-color, border-bottom-color, border-left-color** y **border-right-color**, por ejemplo) o con una lista de dos, tres o cuatro valores sobre el atributo principal igual que hemos hecho con los márgenes:

```
/* 5px para superior e inferior y 50px para derecho e izquierdo */
border-width: 5px 50px;

/*  dotted el superior, solid para los laterales y dashed el inferior */
border-style: dotted solid dashed;

/* los cuatro en el sentido de las agujas del reloj: red superior, blue
derecho, gree inferior y purple izquierdo */
border-color: red blue green purple;
```

UNA NOTA MUY IMPORTANTE: todos estos elementos son válidos para cualquier selector que apliquemos tanto a un elemento de línea como de bloque aunque, tienen mucho más sentido (sobre todo algunos de ellos) sobre uno de bloque.

Existe otra propiedad llamada **outline** mucho menos flexible para aplicar bordes a las cajas.

```
outline: 3px solid black;
```

outline no "roba" espacio a las dimensiones de la caja y, por tanto, no afecta a las dimensiones de esta. Es, por decirlo de alguna forma, como si usáramos un borde exterior al borde de la caja. Por contra es mucho menos flexible que **border** y, por ejemplo, no podemos especificar propiedades diferentes a sus diferentes segmentos.

Imágenes de fondo en las cajas y sus propiedades

Nuestras cajas no sólo pueden tener un color de fondo: también pueden tener una imagen. La sintaxis de la propiedad es la siguiente:

```
background-image: url(http://cort.as/1KYH);
```

Si la imagen está en el propio directorio donde se encuentra la imagen y, por ejemplo, se llama burbujas.jpg sería así: **url(burbujas.jpg)**. Al igual que ocurría con el valor del atributo **src** de la etiqueta **img**, si la imagen se encuentra en el propio servidor pero en otro directorio diferente podríamos especificar una ruta absoluta, relativa, etc.

Por defecto la imagen se coloca en formato de mosaico. Podemos controlar este valor mediante la propiedad **background-repeat** cuyos posibles valores son **repeat, repeat-x, repeat-y** o **no-repeat.** El valor por defecto es **repeat**.

Mediante la propiedad **background-position** podemos dar posición a nuestra imagen de fondo, tanto si estamos usando **no-repeat**, como si lo estamos haciendo (en cuyo caso posicionaría una de las imagenes y crearía el mosaico a partir de esta.) Tenemos tres posibilidades:

```
/* Posiciona la esquina superior izquierda de la imagen en la caja. Está
permitido el uso de valores negativos */
background-position: 50px 100px;

/* Idem al anterior pero usando porcentajes sobre las dimensiones de la
caja */
background-position: 20% 50%;

/* alinea la imagen en la caja admitiendo los valores left center o right
para la posición horizontal y top center o bottom para la vertical. El
orden es indiferente */
background-position: center center;
```

Otra propiedad interesante es **background-attachment**. Tiene dos posibles valores: **scroll** o **fixed.** El primer valor es el del comportamiento por defecto: la imagen se desplaza al mismo tiemp que se desplaza la página. El segundo permite que esta se fije en su posición y que lo único que se desplace sobre ella sea el contenido de la página.

Al igual que en otras de las opciones vistas, también tenemos un formato abreviado para definir todas estas propiedades de una sóla vez. Por ejemplo, si tenemos las siguientes reglas:

```
background-image: url(gotas.gif);
background-color: blue;
background-repeat: repeat-x;
background-position: center top;
background-attachment: fixed;
```

Podríamos sustituirlas por esta otra:

```
background: url(gotas.gif) blue repeat-x top center fixed;
```

El orden de los valores es indiferente y si omitimos alguno de ellos el navegador lo completará con la opción que tenga definida por defecto.

Cajas y jerarquía de elementos

Observa la forma en que aparece este código HTML en tu navegador usando la regla CSS correspondiente a la clase **.ejemplo** vista anteriormente:

```
<h1 class="ejemplo">Hola Mundo</h1>
<div class="ejemplo">Soy la caja número 1
      <div class="ejemplo">Soy la caja número 2</div>
</div>
<p>Hola <strong>mundo</strong> de las cajas. Esto es una prueba.</p>
```

Pon atención a dos cosas: en primer lugar la posición del segundo **div** se realiza de forma relativa al primero. Además, el párrafo final se superpone a la caja número 2 ¿verdad? Para entender la causa de ambas cosas piensa en el árbol formado por las etiquetas HTML. El segundo **div** es hijo del primero y por eso calcula su posición de forma relativa a la de su padre. Por el mismo motivo, como el párrafo final es hermano del primer **div** constituye su referencia para fijar su posición en la pantalla.

5.3. POSICIONANDO LAS CAJAS (Y OTROS ELEMENTOS)

Hasta ahora no nos hemos preocupado de controlar la posición de los elementos que colocamos en nuestro HTML y dejamos que estos se vayan situando uno debajo o al lado del anterior y que sea el navegador el que lo disponga. Esto se debe a que, por defecto, usamos un modo de posicionamiento que se denomina estático (**static**). La propiedad CSS que lo controla se llama **position.**

Aparte de **static**, esta propiedad dispone de tres posibles valores más: **relative, absolute** y **fixed**.

Si usamos el valor **relative**, la posición se indica respecto a la que debería de tener de forma normal dicho elemento. Se hace mediante las propiedades **top, bottom, left** y **rigth**. **left** y **top**, las más usadas, aplican un desplazamiento medido desde la esquina superior izquierda del elemento pudiendo ser este negativo si se expresa mediante un valor mientras que **bottom** y **right** miden ese desplazamiento desde la esquina inferior derecha del elemento. Lógicamente, no tiene mucho sentido usar conjuntamente **top** y **bottom** o **left** y **right**. Se pueden usar porcentajes siendo estos relativos al tamaño del elemento padre al que pertenecen.

```
div.caja1 {position: relative; left: 10%; top -20px;}
div.caja2 {position: relative;right: -40px; bottom: 40px;}
```

Si usamos el valor **absolute** el elemento se posiciona exactamente en las coordenadas que indiquemos. La forma de indicar la posición usa las mismas propiedades vistas en el caso del posicionamiento relativo pero en este caso están referidas a su elemento padre. Si dicho elemento padre es el **body** del

documento HTML estamos haciendo que la posición de este elemento sea independiente del resto del documento HTML de forma que nada que añadamos o quitemos a este puede alterar su posición. Si estás trabajando con cajas, no olvides que el margen exterior de la caja, es parte de la misma.

El último valor es **fixed**. Funciona de forma muy similar al anterior en cuanto a la forma de marcar el lugar donde lo posicionamos salvo que en este caso la posición del elemento queda fija incluso cuando nos desplazamos a través del documento HTML usando las barras de desplazamiento.

Las propiedades float y clear

La propiedad **float,** una de las más útiles en el mundo de las cajas, nos va a permitir modificar el posicionamiento de nuestros bloques y elementos de forma horizontal, colocándolas de forma flotante en el lugar que deseamos. Tiene tres posibles valores: **right, left** o **none** que es su valor por defecto. Vamos a verlo con un ejemplo. Estas serán nuestras reglas CSS:

```
div#contenedor{width: 90%;
               margin-left: 5%;
               margin-top: 30px; }

div#top{width: 100%;
        background-color: green;
        text-align: center;
        color: white;
        font-weight: bold;
        font-size: 3em;
        font-family: Verdana, Helvetica, Arial, Sans; }

div#izquierda{width: 80%;
              height: 50%;
              background-color: aqua;
              float: left;}

div.caja{width:90%;
         margin-left:5%;
         margin-top:10px;
         background-color: red;}

div#derecha{width: 20%;
            height: 50%;
            background-color: teal;
            float: right; }

div#pie{width: 100%;
        background-color: orange;
        font-size: .8em;
        text-align: right;
        font-family: Courier, Monoespace;}
```

Y ahora pondremos esto en el body de nuestro HTML:

```
<div id="contenedor">
      <div id="top">Cabecera de la página</div>
```

```
      <div id="izquierda">
            <div class="caja">Caja 1<br/>      En un lugar de la mancha de
cuyo
            nombre no quiero acordarme no hace mucho que vivía un
            hidalgo... </div>
            <div class="caja">Caja 2</div>
            <div class="caja">Caja 3</div>
      </div>
      <div id="derecha">Esto está en la columna de la derecha</div>
      <div id="pie">Todos los derechos reservados</div>
</div>
```

Observamos que las cajas que usan el selector de **id** derecha e izquierda se posicionan, respectivamente, a derecha e izquierda de la superficie útil de su elemento padre, que aquí marcamos como contenedor. El resultado ya empieza a parecerse a una web actual, pero aún nos queda mucho trabajo para que pueda considerase aceptable desde el punto de vista estético. Observa también que puesto que hemos usado porcentajes en las dimensiones clave de las reglas, la página se redimensiona cuando cambiamos la superficie útil de nuestro navegador. Y una última cosa: para que te des cuenta de lo importante de las medidas en CSS, trata de modificar con una cantidad irrisoria el ancho de la caja izquierda (pon, por ejemplo, 80.005%) o introduce un pequeño padding en las reglas correspondientes al estilo del selector **id** izquierda y verás como se descuadra todo el conjunto.

float es un parámetro que se usa no sólo para posicionar cajas sino, por ejemplo, para posicionar una imagen dentro de un párrafo. La regla CSS:

```
.aderecha {float: right; }
```

Y el HTML:

```
<img class="aderecha" src="cup.png" alt="copa" />
<p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non
risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec,
ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa,
varius a, semper congue, euismod non, mi.</p>
```

Por lógica, se trata de un atributo que no tiene ningún efecto cuando estamos usando posicionamiento **absolute** o **fixed**.

La propiedad **clear** nos permite anular los efectos de la propiedad **float**. Tiene cuatro valores posibles: **none** (por defecto), **right, left** o **both.** Los dos primeros anularían un **float right** o **left**, respectivamente. El valor **both** los anularía a ambos. Para ver el efecto, añadamos la siguiente regla a nuestro CSS anterior:

```
.sinfloat {clear: right;}
```

Y ahora añadamos al final del HTML este nuevo párrafo:

```
<p class="sinfloat"> Lorem ipsum dolor sit amet, consectetuer adipiscing
elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet,
adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam.
Maecenas ligula massa, varius a, semper congue, euismod non, mi.</p>
```

Como podemos ver, el nuevo párrafo salta a la parte inferior de la imagen como se habría comportado si esta no tuviera el atributo **float** mientras que el párrafo anterior sigue ateniéndose a su existencia.

Introduciendo la tercera dimensión en el documento

La propiedad **z-index** nos permitirá estructurar en tres dimensiones los elementos de nuestro documento HTML. Por defecto los elementos aparecen estratificados según su aparición en el documento, de forma que un elemento aparece siempre sobre todos los anteriores a él. La propiedad **z-index** nos permite alterar este orden indicando un número entero, positivo o negativo, que indica el nivel o la capa en la que situamos el elemento. Un ejemplo. Primero veamos las reglas CSS:

```
.caja1 { position: relative;
         background-color: yellow;
         border: 5px solid blue;
         z-index: 2; }

.caja2 { position: relative;
         background-color: purple;
         border: 5px solid blue;
         top: 1px;
         left: 80px;
         z-index: 1; }

.caja3 { position: relative;
         background-color: orange;
         border: 5px solid blue;
         top: -100px;
         left: -1px;
         z-index: -1; }
```

Y luego este fragmento de HTML:

```
<p class="caja1">Hola caja 1!</p>
<p class="caja2">Hola caja 2</p>
<p class="caja3">Hola caja 3</p>
```

Si prescindimos de la propiedad **z-index**, la caja 3 solapará a la 2 y esta a la 1. Con la propiedad **z-index** la caja 3 será la que vaya al fondo mientras que la 1 se encontrará en primer plano. Por defecto el valor de **z-index** es **auto** y el índice de capa base se asigna al 0. Por convención se suele usar el -1 como la capa de fondo y el 999 como la capa que queremos en primer plano, pero en realidad como hemos dicho, la propiedad admite cualquier valor entero negativo o positivo. Entenderás, además, que esta propiedad no tiene ningún efecto ante elementos que no se solapan.

Por último, como ves en la caja 1 se ha establecido la propiedad **position** como **relative** aunque no se efectua ningún desplazamiento sobre su posición normal. Esto es porque la propiedad **z-index** sólo tiene efecto sobre elementos en los que se han modificado las propiedades **position** o **float**.

Visibilidad, desbordamientos y recortes

La propiedad **visibility**, como su nombre indica, permite ocultar un elemento sin que este deje de ocupar su espacio en la página. Tiene dos posibles valores: **visible** o **hidden** y son autoexplicativos. Insisto: el elemento no se ve pero está ahí y si entramos en la opción de ver código del navegador (habitualmente Ctrl+U) podremos verlo, ¡así que cuidado con donde y para qué lo usamos!

La propiedad **overflow** le dice al navegador lo que tiene que hacer cuando un elemento no cabe completamente dentro de la caja o elemento que lo contiene. Su valor por defecto es **auto** y deja esta gestión al navegador. Otras posibilidades son **hidden** (corta y oculta lo que no cabe), **scroll** (coloca unas barras de desplazamiento para poder ver lo que no cabe) o **visible** (lo que no cabe desborda a su contenedor y se muestra en la página). Un pequeño código CSS para verlo:

```
.caja { width: 200px;
        padding: 5px;
        height: 100px;
        border: 1px dashed black;
        overflow: hidden;}
```

Y este sería el HTML de prueba:

```
<div class="caja">
      <p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed
      non risus. Suspendisse lectus tortor, dignissim sit amet,
      adipiscing nec, ultricies sed, dolor.</p>
</div>
```

Modifica el valor de **overflow** para ver las diferentes posibilidades. Y cuidado. Al igual que ocurría con la propiedad **visibility**, si aquí usamos el valor **hidden** esto no implica que no se pueda ver en la vista de código de la página. Existen las propiedades **overflow-x** e **overflow-y** por si queremos controlar individualmente las dos posibles barras de scroll posibles.

La propiedad **clip** nos permite fijar la parte visible de un elemento definiendo una zona rectangular y sólo es aplicable en elementos cuya posición se ha fijado como **absolute**. La sintaxis es así:

```
clip: rect(0 200px 100px 0);
```

Esto dejaría visible un rectángulo que, respecto al elemento en el que se aplique, empieza a 0px de la parte superior, llega hasta los 200px hacía la derecha, 100px hacia abajo y 0px desde la izquierda (o sea, cuatro coordenadas en el sentido de las agujas del reloj empezando desde arriba, como siempre)

Puedes ver un ejemplo completo con esta regla CSS:

```
.caja { width: 25%;
        padding: 5px;
        position: absolute;
        border: 1px dashed black;
        clip: rect(0 200px 100px 0);}
```

Y este código HTML que la usa:

```
<div class="caja">
      <p> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed
      non risus. Suspendisse lectus tortor, dignissim sit amet,
      adipiscing nec, ultricies sed, dolor.</p>
</div>
```

Y la misma advertencia de siempre: recuerda que la parte que no es visible se puede ver con la opción de Ver código del navegador!

5.4. OTRAS PROPIEDADES

Cambiar el aspecto del puntero del ratón

Podemos cambiar el aspecto del cursor cuando pasamos sobre un elemento con la propiedad **cursor**. Los valores posibles y su apariencia son:

- **auto.** El navegador determina por sí mismo el cursor según el contexto.
- **crosshair.** El cursor muestra una cruz.
- **default**. El cursor por defecto del sistema operativo, a menudo una flecha.
- **e-resize**. Cursor apuntando hacia el este.
- **ne-resize**. Cursor apuntando hacia el noreste.
- **nw-resize**. Cursor apuntando hacia el noroeste.
- **n-resize**. Cursor apuntando hacia el norte.
- **se-resize**. Cursor apuntando hacia el sureste.
- **sw-resize**. Cursor apuntando hacia el suroeste.
- **s-resize**. Cursor apuntando hacia el sur.
- **w-resize**. Cursor apuntando hacia el oeste.
- **help**. El cursor indica una ayuda. A menudo se muestra un signo de interrogación.
- **move**. El cursor indica un objeto que se puede desplazar.
- **pointer**. El cursor presenta un dedo que indica un enlace.
- **progress**. El cursor muestra una flecha con un reloj de arena.
- **text**. El cursor indica que es posible seleccionar el texto.
- **wait**. El cursor indica una progresión. A menudo se muestra un reloj de arena.
- **url.** Especifica un archivo donde se encuentra la imagen que se desea usar como cursor. El archivo de imagen especificado en la URL debe tener el formato cur (cursor) o ani (cursor animado).

Un ejemplo:

```
.caja { width: 50%;
        padding: 5px;
        border: 1px dashed black;
        cursor: crosshair;}
```

Alterando si un elemento es de bloque o de línea

La propiedad **display** nos permite redefinir el caracter de un elemento como de bloque o de línea. **block, inline** o **none.** son los valores más usados. Los dos primeros son autoexplicativos. El tercero hace que el elemento no se muestre pero, al contrario de lo que ocurría con la propiedad **visibility** y el valor **hidden**, ahora el espacio que ocuparía el elemento no se reserva. Este, sin embargo, sigue visible al dar a la opción de ver código.

Una utilidad habitual de esta propiedad es mostrar uno al lado del otro los elementos de una lista para, por ejemplo, construir un menú (ampliaremos este uso en un capítulo posterior):

```
li{ display: inline;
    list-style-type: none; }
```

En realidad la propiedad **display** es mucho más compleja y tiene otras opciones disponibles, pero no vamos a detenernos a verla. Existe una buena referencia acerca de ella aquí:

http://www.librosweb.es/referencia/css/display.html

Modificando el aspecto de los bordes de las celdas de una tabla

Las propiedades **border-collapse** y **border-spacing** controlan la forma en que vemos los bordes de una tabla. La primera tiene dos posibles valores: **collapse** y **separate**. En el primer caso los bordes adyacentes se unirán en uno sólo siempre que sea posible mientras que con el segundo valor la separación entre bordes se controlará mediante la propiedad **border-spacing**. El valor por defecto es **separate**.

border-spacing indica el espacio que habrá entre los bordes de celdas adyacentes, por ejemplo:

```
border-spacing: 2px 0;
```

Si se indican dos valores el primero hace referencia a la separación horizontal y el segundo a la vertical. En el caso de indicar un sólo valor se aplica a ambas distancias

Por último, la propiedad **empty-cells** tiene dos posibles valores (**show** o **hide**) controla que se vean o no los bordes y fondo de las celdas vacías de una tabla. El valor por defecto es **show**.

6. SELECTORES AVANZADOS

Aunque hemos dicho que prácticamente cualquier página web puede hacerse con los cinco selectores básicos que ya conocemos, existen algunos que se llaman avanzados que nos van a permitir, bien simplificar algunas tareas de aplicación de estilos o bien permitirnos algunos efectos imposibles de conseguir de ninguna otra forma como es el caso de la utilidad proporcionada por alguna de las pseudo-clases. Los presentaremos en las siguientes páginas.

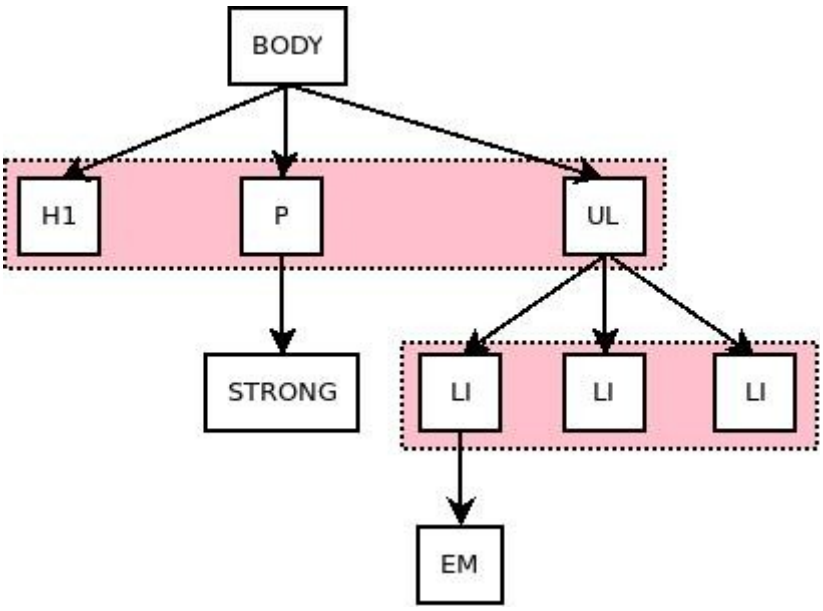
Selector de hijos

El selector de hijos es muy similar al selector descendiente pero es mucho más específico. Cuando uso un selector descendiente, por ejemplo **p .destacado** sólo me importa que el elemento perteneciente a la clase **destacado** se encuentre dentro del elemento **p**, sin importarme cuan "profundo" se encuentre dentro del árbol jerárquico de etiquetas del documento. Si uso el selector de hijo es porque quiero que el elemento con la clase **destacado** sea hijo directo del elemento **p** y no un descendiente cualquiera. Y se simboliza así:

```
p > .destacado {color: purple; }
```

Selector de hermanos adyacentes (o adyacente, sin más)

El selector de hermanos es aquel que nos permite seleccionar elementos que comparten el mismo padre y están adyacentes uno al otro. En el siguiente grafico podemos ver destacados los hermanos de un HTML sencillo ya visto en otros ejemplos:



Para aplicar un determinado estilo al primer párrafo después de un titular de nivel 1, usaríamos la siguiente sintaxis:

```
h1 + p {text-indent: 2.5em; }
```

El selector de hermanos es una forma muy útil de crear estilos para ciertas combinaciones de elementos muy utilizados: el primer párrafo de un capítulo, el primer párrafo posterior a una tabla, etc. También es útil para seleccionar todos los elementos menos el primero en una secuencia determinada. Por ejemplo:

```
p + p { text-indent: 1.5em; }
```

Selector de atributos

Los selectores de atributos nos permiten aplicar estilos a determinados elementos en función de los atributos que acompañen a la etiqueta HTML o de los valores que tengan dichos atributos. Si nos paramos a pensar, en realidad los selectores de clase o de ID son una especie de selectores de atributo porque aplican estilos en función del valor de un atributo (**class** o **ID**, respectivamente), así que no vamos a ver nada demasiado nuevo en este apartado.

Existen cuatro tipos diferentes de selectores de atributo. Los primeros seleccionan los elementos en función de un determinado atributo independientemente de su valor:

```
td[rowspan] {background-color: black; color: white; }
```

Los segundos seleccionan un elemento en función de un atributo con un valor determinado:

```
a[href="http://www.google.es/"] {color: red; text-weight: bold; }
```

La tercera clase de selectores de atributos realiza su selección en función del atributo y de la existencia de un determinado valor pero que no tiene porqué ser unico, sino simplemente aparecer en una lista separada por espacios:

```
div[class~=”alerta”] {color: red;}
```

Podemos correr el riesgo de pensar que simplemente es una forma de hacer más complicado lo que ya sabemos hacer con otros selectores más simples. Por ejemplo, estas definiciones son casi idénticas:

```
div.alerta {color: red;}
div[class=”alerta”] {color: red;}
div[class~=”alerta”] {color: red;}
```

Pero también podemos pensar en otras posibilidades más interesantes:

```
img[alt~=”ilustración”] {border: 2px solid;}
table[summary~=”2010”] {background-color: green;}
*[title~=”importante”] {color: red; cursor: crosshair; }
```

El cuarto tipo de selector de atributo apenas tiene utilidad. Permite seleccionar elementos con cierto atributo y que incluyen un determinado valor como el primer elemento de una lista separada por guiones.

```
*[lang|="es"] { color: blue; }
```

El anterior ejemplo seleccionaría a todos los elementos con un atributo **lang** que

empiece por es- (por ejemplo es-ES, es-MX, es-AR, etc.) De hecho esta es su principal (y casi única) utilidad: la selección de elementos según los códigos estándares de idiomas.

6.1. PSEUDO-CLASES Y PSEUDO-ELEMENTOS.

Existen ciertos estados o características de nuestro documento HTML a los que es útil y apropiado aplicarle un estilo diferenciado: un enlace ya visitado, la primera letra de un párrafo, etc. Esto se consigue mediante las pseudoclases. Para identificar que hablamos de una pseudo-clase en una regla CSS usamos los dos puntos como signo diferenciador y estas pueden aplicarse sobre selectores de tipo, de clase, etc. Veremos ejemplos enseguida.

Y una nota: en algunos textos se hace una separación entre pseudo-clases y pseudo-elementos mientras que en otras se engloban todas bajo la misma categoría. Aquí haremos la diferenciación clásica porque, si bien no existen grandes diferencias, si que es necesario hacer la distinción a la hora de calcular la especificidad de las reglas CSS.

Pseudo-clases

Las pseudo-clases son aquellas que se aplican fundamentalmente sobre los selectores de enlace (aunque alguna de ellas pude aplicarse también sobre otros elementos). Se trata de **:link, :visited, :hover, :active** y **:focus**.

La pseudo-clase **:link** hace referencia a un enlace que aún no ha sido visitado, mientras que **:visited** hace referencia a lo opuesto, es decir, los enlaces que ya han sido visitados por el usuario. Dos ejemplos:

```
a:link {color: green; background-color: yellow;}
a:visited {color: yellow; background-color: green; }
```

La pseudo-clase **:hover** nos dicta los estilos a aplicar al pasar el ratón por encima del elemento al que el selector hace referencia. Se suele usar fundamentalmente con enlaces pero es aplicable a cualquier otro elemento:

```
a:hover, p.destacado:hover{font-weight: bold; }
em:hover {color: blue;}
```

:active es una pseudo-clase cuyos estilos se aplican al hacer click sobre un elemento o activarlo mediante la tecla Intro cuando tiene el "foco":

```
h1:active {color: orange; background-color: black;}
```

Como imaginamos, muchas de estas propiedades pueden activarse simultaneamente: un enlace puede haber sido visitado y, podemos pasar el ratón por encima de él, etc. EL orden en que se colocan en la hoja de estilos es por ello fundamental para aplicarlos correctamente cuando queremos aplicar reglas distintivas a todos ellos:

```
a:link {color: blue;}
a:visited {color: green;}
a:hover {color: orange;}
```

```
a:active {color: red;}
```

Podemos aplicar también reglas a dos pseudo-clases simultaneamente. Por ejemplo, la siguiente regla se aplicaría sobre un enlace visitado sobre el que hacemos click:

```
a:visited:active {color: black; background-color: orange;}
```

Por último, **:focus** es una regla que se aplica cuando el elemento tiene el "foco", es decir, cuando se activaría si pulsamos la tecla Intro. Para poder usarla debemos de definir que elementos tienen el foco en nuestra página y el orden en el que lo reciben (cada vez que pulsamos secuencialmente el tabulador). Esto se hace mediante el atributo **tabindex** que podemos usar en las etiquetas a (enlaces) de nuestra web posibilitando así la navegación sin necesidad de ratón. Por ejemplo:

```
<p>Un enlace a <a href="http://www.google.es/" alt="Google"
tabindex="1">Google</a>, otro a <a href="http://www.bing.es" alt="Bing"
tabindex="2">Bing</a> y un tercero a <a href="http://www.yahoo.es"
alt="Yahoo" tabindex="3">Yahoo</a>.</p>
```

Y ahora ya si podemos ver los efectos de la pseudo-clase :focus

```
a:focus {background-color: black; color: white; font-size: 1.8em;}
```

La pseudo-clase **:first-child** nos permite seleccionar un determinado elemento siempre y cuando sea el primer hijo de su padre. Por ejemplo, veamos las reglas siguiente:

```
p:first-child {color: blue; }
p:first-child em {background: yellow; }
```

La primera selecciona y aplica estilo a un párrafo siempre y cuando sea el primer hijo de su padre, mientras que la segunda hace lo mismo sobre las cursivas de ese mismo tipo de párrafos.

Existe una última pseudo-clase de aplicación muy limitada llamada **:lang** que nos permite aplicar estilos diferenciados en función del idioma en que esté escrito el elemento. Por ejemplo así:

```
p:lang(es) {color: red; }
```

Aunque tienen un uso muy similar al visto en uno de los selectores de atributo, existe una ligera diferencia. El selector de atributo es aplicable sólo a los contenidos que usen el atributo lang, mientras que esta pseudo-clase sería aplicable también en otras circunstancias, como por ejemplo si el propio navegador fuese capaz de analizar el contenido y detectar elementos escritos en diferentes idiomas (cosa que, por el momento, no hacen demasiado bien...).

Pseudo-elementos

Los dos primeros pseudo-elementos que veremos son aplicables a la decoración

de párrafos de texto. Se llaman **:first-letter** y **:first-line** y sus nombres son autoexplicativos

```
p.decorado:first-letter {margin-right: 4px; font-size: 4em; color: red;
font-weight: bolder;}
p.decorado:first-line {font-variant: small-caps; font-weight: bold;}
```

Existen otros dos pseudo-elementos llamados **:before** y **:after** que usados junto con la propiedad **content** nos permiten insertar contenido (texto, imágenes, etc.) antes o despues de un elemento.

```
p#insert1:after{ content: " (TM – Marca Registrada) ";}
p.insert2:before{ content: url(icon_new.gif);}
```

En los anteriores ejemplos, el primero inserta el texto que aparece entre comillas al final de cada párrafo identificado con el selector de ID insert1, mientras que el segundo inserta el gif animado icon_new.gif delante de cada párrafo identificado como perteneciente a la clase insert2.

7. ALGUNOS DISEÑOS Y EFECTOS ÚTILES

Con HTML sólo pueden dibujarse cajas. Cajas y rectángulos absolutamente cuadrados. Esto es verdad al 99.00% (sobre todo hasta la aparición de HTML5) pero, sin embargo, no es eso lo que vemos en la web. Existen numerosos trucos y efectos de diseño para "engañar" al espectador y darle la sensación de que está viendo algo con un contenido mucho más elaborado de lo que en realidad tiene delante. En este capítulo aprenderemos a dibujar cajas mucho mejores de las que ya conocemos, pero también a decorarlas de forma que parezcan otras muchas cosas. Nos apoyaremos para ello en la amplia colección de ejemplos que acompañan al capítulo (y que, si aún no lo has hecho, puedes descargarte desde http://blog.unlugarenelmundo.es/hecho-en-casa/) y desde aquí los iremos analizando uno a uno e indicaremos cuales son las claves de diseño que usamos en los mismos.

7.1. DISEÑOS DE ANCHO FIJO, LÍQUIDOS (O FLUIDOS) Y ELÁSTICOS

Las tres estrategias "puras" de afrontar un diseño web son: diseños de ancho fijo, de ancho líquido (o fluido) o de ancho elástico.

Los diseños de ancho fijo configuran sus medidas mediante píxeles. Son fáciles de realizar y no tenemos que hacer demasiadas pruebas ni ajustes para asegurarnos que se verán bien. Puesto que ya casi no existen monitores con resolución inferior a 800 píxeles de ancho se suele tomar como estándar un ancho de 760 o 780 píxeles para el contenido.

Los diseños de ancho líquido son aquellos que trabajan mediante porcentajes. Aprovechan mucho mejor la superficie en las pantallas grandes pero nos exigen más pruebas para asegurarnos de que todo se ve correctamente en resoluciones bajas.

Los diseños elásticos son los más difíciles de implementar. Basan sus medidas en la unidad **em**, de forma que si incrementamos el tamaño del texto, también aumentamos proporcionalmente las dimensiones de nuestro diseño.

Nos encontraremos por ahí muchos modelos híbridos pero estos son los fundamentales. Por ejemplo, se habla de un diseño híbrido líquido-fijo cuando lo fundamental del diseño es líquido pero existe en él algún elemento de ancho fijo. Por ejemplo, podemos pensar en un diseño a dos columnas donde una de ellas es de ancho fijo (para emplazar menús, publicidad, etc.) mientras que la otra, destinada a los contenidos, tiene un diseño líquido.

Ejemplo 1 – Cajas

En el ejemplo número 1 nos encontramos cuatro diseños de cajas básicos donde aprendemos a solventar diferentes problemas. Hasta ahora las cajas con las que habíamos trabajado (en el capítulo 5) tenían el ancho, o bien fijo, o bien líquido y el alto fijo. En el mundo real necesitamos diseñar cajas de alto variable en función del contenido y partiendo de la premisa de que no conocemos el tamaño de ese contenido.

Si metemos dos columnas dentro de una caja contenedora, que suele ser la estrategia habítual, y no definimos alturas fijas las cajas desbordarán al contenedor descuadrando el diseño. Tenemos varias formas de resolver esto:

- Poner un pie al diseño. Este pié debería de llevar definida la propiedad clear para "romper" el efecto de los float. Si no queremos tener pié o no lo necesitamos podemos poner uno "invisible".

```
#pie{
      color: white;
      width: 760px;
      padding: 10px;
      }
```

- Una segunda solución es usar también la propiedad float en el contenedor.

```
#contenedor2 {
      border: 2px dotted black;
      margin: 5px auto;
      width: 780px;
      float: left;
      }
```

- Una tercera opción sería definir la propiedad overflow a auto o hidden en la caja contenedora:

```
#contenedor {
      border: 2px dotted black;
      margin: 5px auto;
      width: 780px;
      overflow: auto;
      }
```

Encontramos otro "problema" cuando queremos colocar tres o más columnas. Hasta ahora sólo hemos usado dos y colocamos una a la derecha con **float: right** y otra a la izquierda con **float: left**. En realidad, si ajustamos bien el espacio podríamos haber usado dos veces uno u otro porque los float se "apilarían" en horizontal llenando todo el espacio disponible. Esto nos da la idea de como hacerlo con, por ejemplo, tres columnas: podemos "apilar" la central a la derecha o a la izquierda pero, eso si, calculando siempre bien las dimensiones:

```
#izquierda2 {
      float: left;
      width: 31%;
      padding: 1%;
      }

#centro2{
      float: left;
      width: 32%;
      padding: 1%;
      }

#derecha2 {
      float: right
      width: 31%;
      padding: 1%;
```

```
}
```

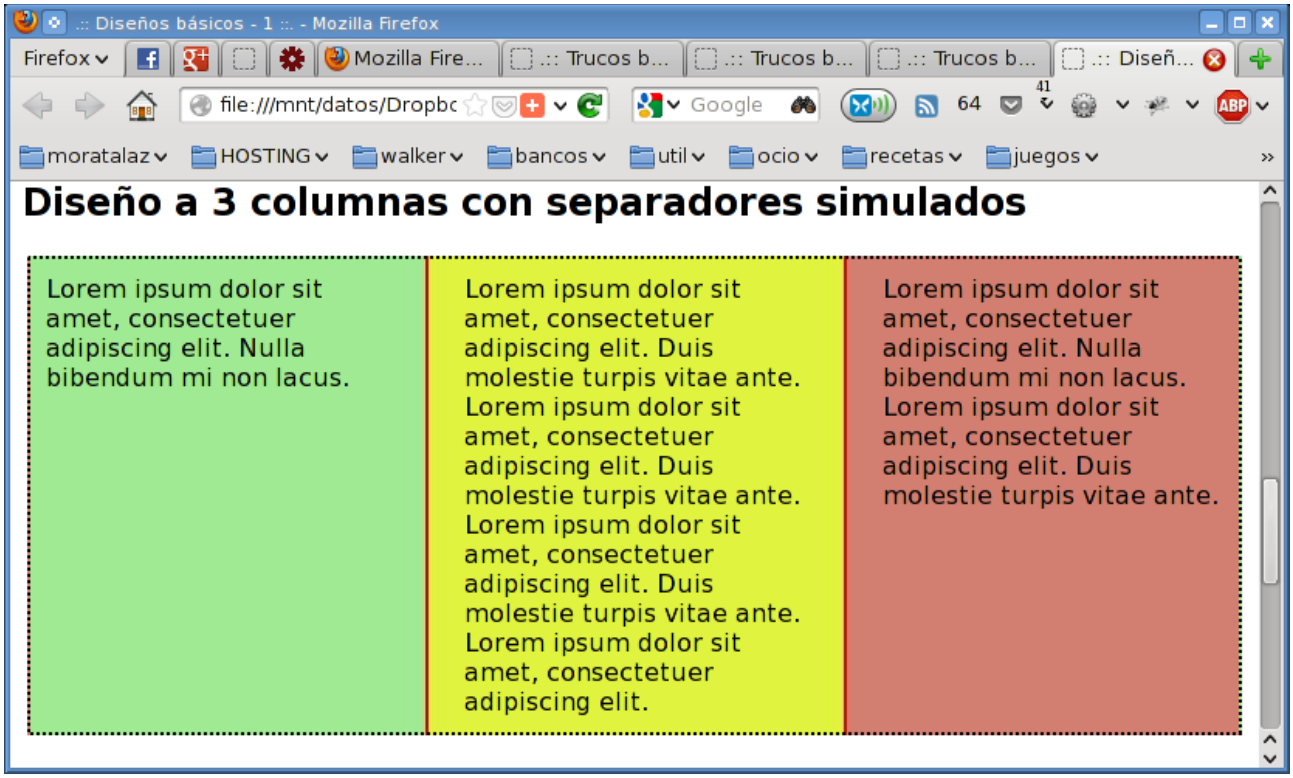
Observemos una cosa en el ejemplo anterior en el que, como vemos, el diseño del ancho es líquido. El ancho total se ha calculado para llenar completamente el contenedor ¿lo ves? El padding es el espacio interior entre el contenido de la caja y el borde de la misma y se deja tanto a derecha como a izquierda, es decir, un 2% por cada columna. En total hacen el 100% Si las columnas tuvieran bordes pasaría lo mismo: un borde del .5% ocuparía un 1% por cada columna. Es un error muy común hacer mal estos cálculos, tanto en diseños líquidos como fijos, y en ese caso las cajas se desbordarían, quedarían descuadradas, con márgenes antiestéticos, etc.

En los dos ejemplos anteriores hemos usado, intencionadamente, colores chillones para cada columna para que quede claro que la columna más alta, sea cual sea, es la que marca el alto total del diseño. Puedes hacer pruebas de esto sumando o quitando contenido en unas u otras. El problema que tenemos con estos diseños es que, o el fondo es blanco para todas, o si queremos usar imágenes o colores el resultado es bastante poco profesional. Idem si quisiéramos usar bordes separadores para cada columna. ¿Cómo lo solventamos? En la caja número tres de este ejemplo tenemos una forma rápida de hacerlo pero sólo válida para diseños de ancho fijo: usar una imagen de fondo para el contenedor de nuestras columnas donde van dibujados los fondos y separadores de cada una de ellas. Tan fácil como esto:

```
#contenedor3 {
      background-image: url(fondo3.jpg);
      border: 2px dotted black;
      margin: 5px auto;
      width: 780px;
      overflow: auto
      }
```

La imagen a insertar debaría de ser algo así:

Y nuestra tabla así:



Pero la solución "real" válida para cualquier tipo de situaciones no es esta. La tienes en el diseño número 4 de este ejemplo 1. El "truco" consiste en usar una estructura con dos contenedores siguiendo los siguientes puntos:

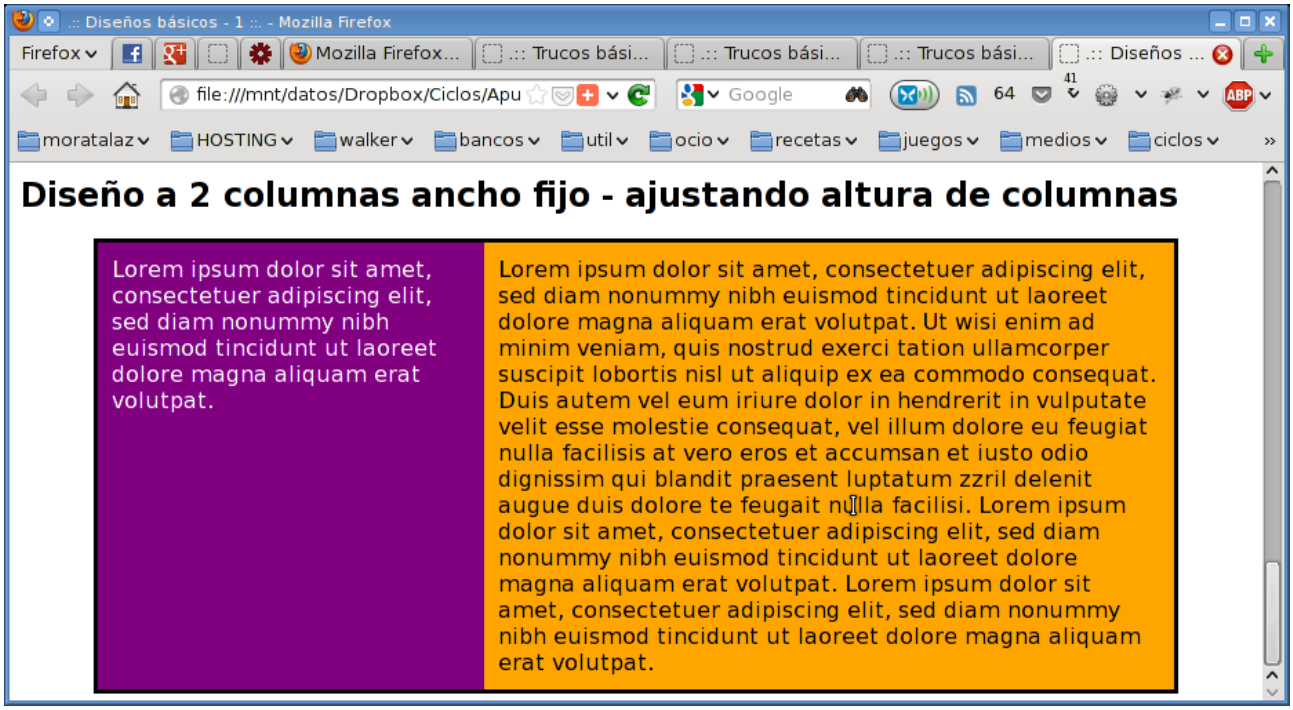
- El primer contenedor tendrá, como siempre, el ancho total del diseño y llevará como color de fondo el que queramos dar a la columna de la izquierda.
- El segundo contenedor ira dentro del primero con un **float: right;**, ocupando el ancho de la columna de la derecha y con el color de fondo que queramos que lleve esta.
- Dentro de este segundo contenedor metermos las dos columnas. La de la derecha ocupando el ancho de dicho contenedor. La de la izquierda con la propiedad **position: relative;** y un **margin: left;** con valor negativo para desplazar esa columna a la posición a la izquierda que le corresponde. El HTML simplificado al máximo quedaría así:

```
<div id="contenedor4">
      <div id="contenedor4-2">
            <div id="izquierda4">Lorem ipsum dolor sit amet</div>
            <div id="derecha4">Lorem ipsum</div>
      </div>
</div>
```

Y el CSS:

```
#contenedor4 {
```

```
        width: 780px;
        margin: 5px auto;
        color: white;
        background-color: purple;
        overflow: auto;
        outline: 3px solid black;
        }

#contenedor4-2 {
        width: 500px;
        float: right;
        background-color: orange;
        }

#derecha4 {
        width: 480px;
        float: right;
        padding: 10px;
        color: black;
        }

#izquierda4 {
        width: 260px;
        float: left;
        padding: 10px;
        position: relative;
        margin-left: -280px;
        }
```



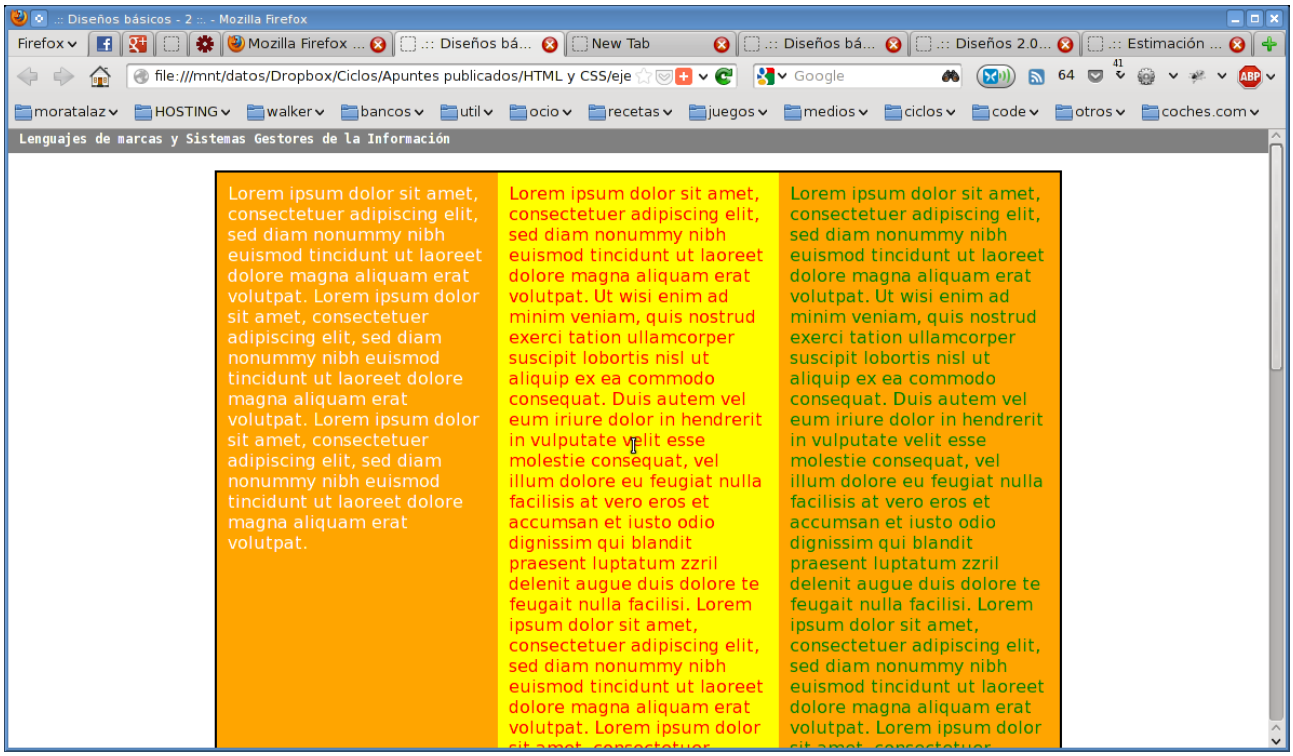
Ejemplo 2 – Tres columnas

¿Y si necesitamos tres columnas? El sistema sería muy similar al visto en el último caso con la salvedad de que ahora necesitaríamos un contenedor por cada una

de las columnas (para darle color al fondo de cada una de ellas de forma diferenciada) y todas las columnas "reales" con contenido irían dentro del último contenedor, a la derecha, con los desplazamientos a la izquierda que veamos oportunos. Y así podríamos seguir con cuatro, cinco, etc.

En el ejemplo 2 tienes el diseño típico a tres columnas del mismo ancho al que se ha añadido algo que ahora está de moda en todos los diseños web: un cintillo fijo en la parte superior de la página donde habitualmente se sitúan enlaces a menús, al perfil de usuario, cerrar sesión, etc. Esto se consigue de forma fácil con una caja en la que hemos definido dos propiedades: **position: absolute:** y **z-index : 99;**

```
#cintillo {
      width: 100%;
      height: 18px;
      padding: 2px 10px;
      position: fixed;
      top: 0;
      left: 0;
      z-index: 99;
      background-color: gray;
      color: white;
      font-weight: bolder;
      font-family: monospace;
      }
```



Ejemplo 3 – Diseño de alto y ancho fijo

Los diseños de alto y ancho fijos no tienen mayor complejidad y se incluye este

ejemplo sólo para completar la colección de casos posibles.

Ejemplos 4 y 5 – Diseños de ancho líquido

La complejidad del diseño de ancho líquido con múltiples columnas reside en los cálculos para ajustar estas sin que se desborden o descuadren al modificar el ancho del navegador. En el ejemplo 4 tienes un caso a tres columnas en el que se ha tratado de detallar en los comentarios la forma de realizar estos cálculos. El ejemplo 5 sería exactamente igual pero aquí ya usamos adicionalmente la estrategia de contenedores anidados que nos permite que todas las columnas tengan la misma altura independientemente de cual sea la que tenga más cantidad de contenido. En ambos casos usamos, además, dos columnas auxiliares para usarlas como separadores entre las columnas principales.

Ejemplo 6 – Diseño fijo, fluido, fijo

En este ejemplo presentamos una estructura en la que las columnas laterales, derecha e izquierda, son de ancho fijo y es la columna central la que modificará su ancho de forma fluida (porcentual) en función del ancho del navegador.

Los "trucos" de diseño son los siguientes:

- Suponemos que el contenido principal estará en la columna central la cual siempre tiene que exceder en longitud a las laterales. En caso contrario el diseño no podría ser tan simple como se presenta aquí.
- Las columnas laterales llevan tres propiedades: **position: absolute; top: 0;** y luego, según corresponda, **rigth:0;** para la columna de la derecha y **left:0;**

para la de la izquierda.

- El padding a derecha e izquierda de la columna principal central ha de ser fijo y suficiente como para "cubrir" el espacio que ocupan las columnas laterales.

Ejemplo 7 – Separadores entre columnas transparentes

En este ejemplo usamos una variante del método visto para que el separador entre las columnas sea transparente y deje ver el fondo de la web que, en este caso, es un degradado de color vertical. Para ello el contenedor principal ocupa el espacio de la columna de la derecha, el secundario, dentro de este, tiene los atributos de **position: relative;** y **margin-right: -260px;** para desplazarlo a la derecha, dejando un espacio vacío entre ambos que actuará como separador y dejará ver el fondo de la web y, por último, dentro de este segundo contenedor alojaremos las dos columnas, una de ellas, la izquierda, de nuevo con **position: relative;** y un **margin-left: -710;** para desplazarla a su posición en la izquierda.

Ejemplo 8 – Sombras y diagonales

El ejemplo 8 está extraido directamente de la web y es un caso sencillo de como introducir sombras, bordes redondeados y líneas diagonales en nuestro diseño de forma sencilla. El efecto se consigue mediante tres imágenes: una de cabecera que introduce unos bordes redondeados y una diagonal que cruza de parte a parte el diseño, una de pié con los bordes redondeados y la sombra inferior, y otra que se repite verticalmente a lo largo de todo el cuerpo del diseño para completar el efecto de sombreado. La elección de introducir la imagen de cabecera en el html y no en el css es discutible, pero he querido dejar el diseño

tal y como lo encontré en la web con los mínimos cambios para su estudio:

Cabecera del doumento en el HTML:

```
<div id="header">
      <img src="eldiario.png" width="883" height="260" alt="eldiario.es">
</div>
<div class="article">
      …
</div>
```

Repetición de la imagen en el cuerpo del diseño en el CSS:

```
div.article {
      padding: 0px;
      width: 883px;
      margin-top: 0px;
      margin-right: auto;
      margin-bottom: 20px;
      margin-left: auto;
      background-image: url(fondo-eldiario.png);
      background-repeat: repeat-y; }
```

Pié del documento, en el CSS:

```
#abajo {
      background-image: url(fondo-abajo-eldiario.png);
      background-repeat: no-repeat;
      height: 25px;
      margin: 0px;
      padding: 0px; }
```

7.2. DISEÑO DE TABLAS

El ejemplo 9 muestra como dar un aspecto profesional a una tabla con apenas unas pocas propiedades. El fondo del diseño tiene un degradado de color vertical conseguido mediante la repetición de una imagen de fondo. Las propiedades, como puedes ver, no tienen nada especial más que una adecuada utilización de tipografías, colores y espaciados. Tan solo hay una nota especial a tener en cuenta.L a tabla usa la propiead **border-spacing: 0 1px;** que define el espacio de separación entre celdas adjacentes. El primer valor es el que tendrá en cuenta para el espaciado horizontal y el segundo para el vertical (si hubiera sólo un valor aplicaría a ambas magnitudes). Con esto conseguimos anular el espaciado horizontal por defecto (de un pixel) que dejaría un punto en blanco entre celda y celda en las líneas horizontales que delimitan las filas. Por lo demás no existe nada más a reseñar.

Curso	Asignatura	Gr	Ss	Hr	Sem	Mes	Tema	Título	% tema	% tot.est.	% tot.real
DEBUG	DEBUG	-	22	18	1	0	2	2 semanas	11%	41%	
ESO3	Inicio prog. y leng.visuales	C	50	42	25	6	6	Cajas de comprobación. (CheckBox). Botones de selección. (RadioButton). Cajas de agrupación. (Panel y GroupBox)	21%	56%	
ESO4	Informática	A	75	63	25	6	7	Procedimientos de diseño gráfico	92%	56%	
		BD	75	63	25	6	7	Procedimientos de diseño gráfico	92%	56%	
		C	75	63	25	6	7	Procedimientos de diseño gráfico	92%	56%	
BACH1	Tecnologías de la información	ABC	100	83	25	6	4	Multimedia.	61%	45%	
SMR1	Aplicaciones ofimáticas	1º	178	148	22	6	8	Bases de datos. Conceptos	83%	55%	
	Sistemas operativos monopuesto	1º	133	111	22	6	7	Windows. Administración y configuración	66%	56%	
	Redes	1º	155	129	22	6	8	Servicios de red	38%	54%	
	Montaje y mantenimiento de equipos	1º	134	112	22	6	9	Ensamblaje de un ordenador	38%	56%	
SMR2	Servicios en red	2º	200	167	25	6	10	Servicios de voz transportada por servicios de red (VoIP)	58%	98%	
	Seguridad en redes	2º	100	83	25	6	9	Auditorias de seguridad	67%	98%	
	Sistemas Operativos en red	2º	200	167	25	6	11	Sistemas heterogeneos en red	47%	97%	

7.3 OTROS TRUCOS

Ejemplos 10 y 11 – Transpariencias

Los efectos con imágenes translúcidas y que parecen "fijarse" al fondo de la web mientras que el contenido se desplaza son muy utilizados y fáciles de implementar. En estos dos ejemplos podemos comprobarlo. Las bases de este tipo de diseño son las siguientes:

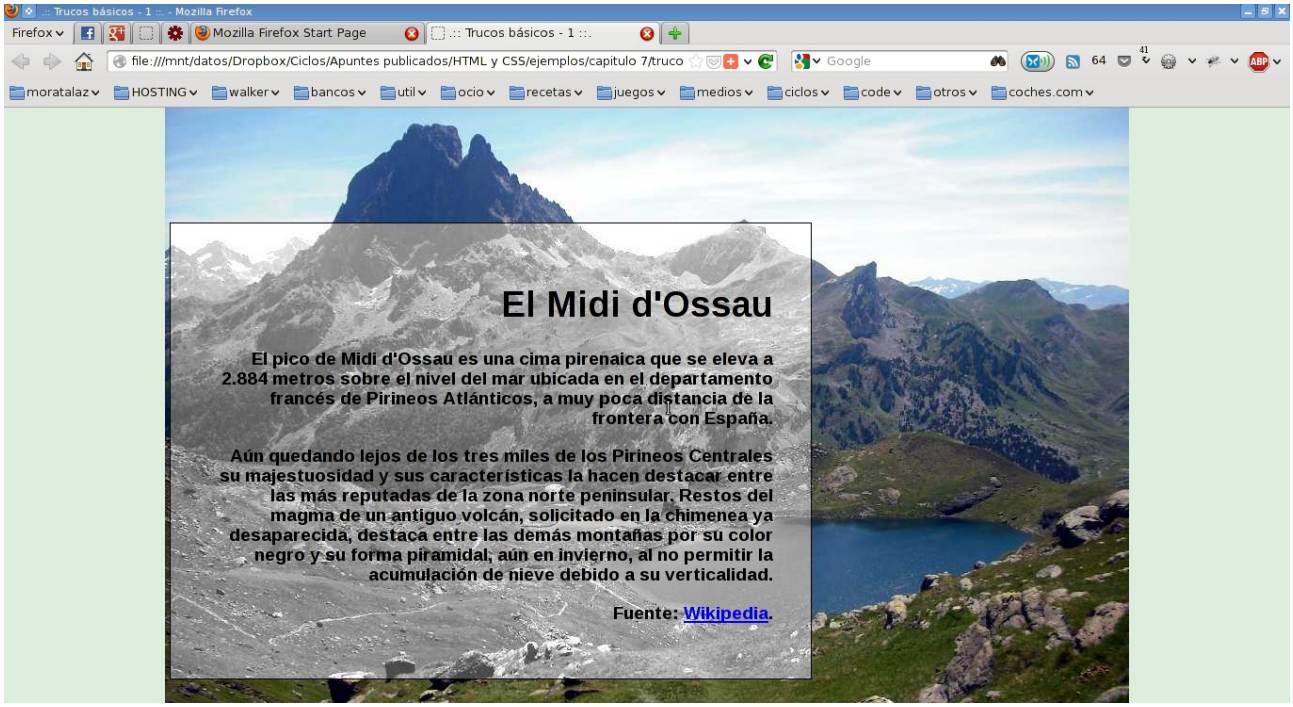
- Debemos de contar con un juego de imágenes especialmente preparadas. Todas ellas similares pero con algún "efecto diferenciador"

- Situaremos cada una de estas imágenes en una caja diferente de forma que el color de fondo de la caja (**background-color**) coincida con el color de fondo de la imagen

- Fijaremos todas las imágenes de la misma forma usando las propiedades **background-attachment** y **background-position**

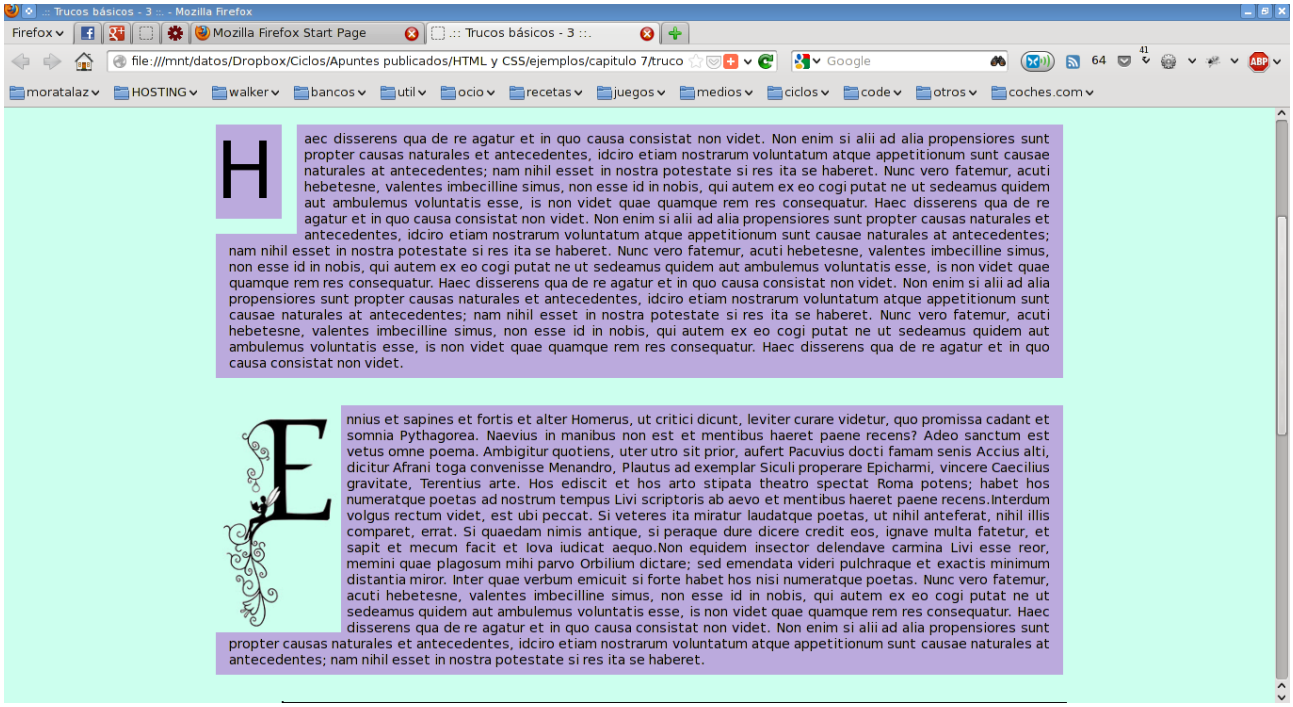
  En el siguiente CSS extraido del ejemplo 1 se muestran estas propiedades en formato comprimido salvo en la caja correspondiente al body del documento:

```
body {
      background-color: #ded;
      background-image: url(world-1.jpg);
      background-position: top left;
      background-repeat: no-repeat;
      background-attachment: fixed;
      padding: 1% 5%;
      }

div#contenedor {
      background: white url(world-2.jpg) top left no-repeat fixed;
      width: 800px;
      margin: 15px auto;
      border: 1px solid black;
      padding: 40px;
      }

div#contenedor h2 {
      background: #cbb196 url(world-3.jpg) top left no-repeat fixed;
      color: white;
      padding: 20px;
      margin: 20px auto;
      border: 1px solid black;
      }
```



Ejemplo 12 – Box Punch

El Box Punch es otro ejemplo muy socorrido para disimular el hecho que ya conocemos de que sólo podemos trabajar con cajas perfectamente cuadrangulares. Mediante la ilusión de combinar colores de fondo y bordes en diferentes cajas superpuestas se consigue el efecto de hacer creer que estas cajas son más irregulares de lo que en realidad lo son.

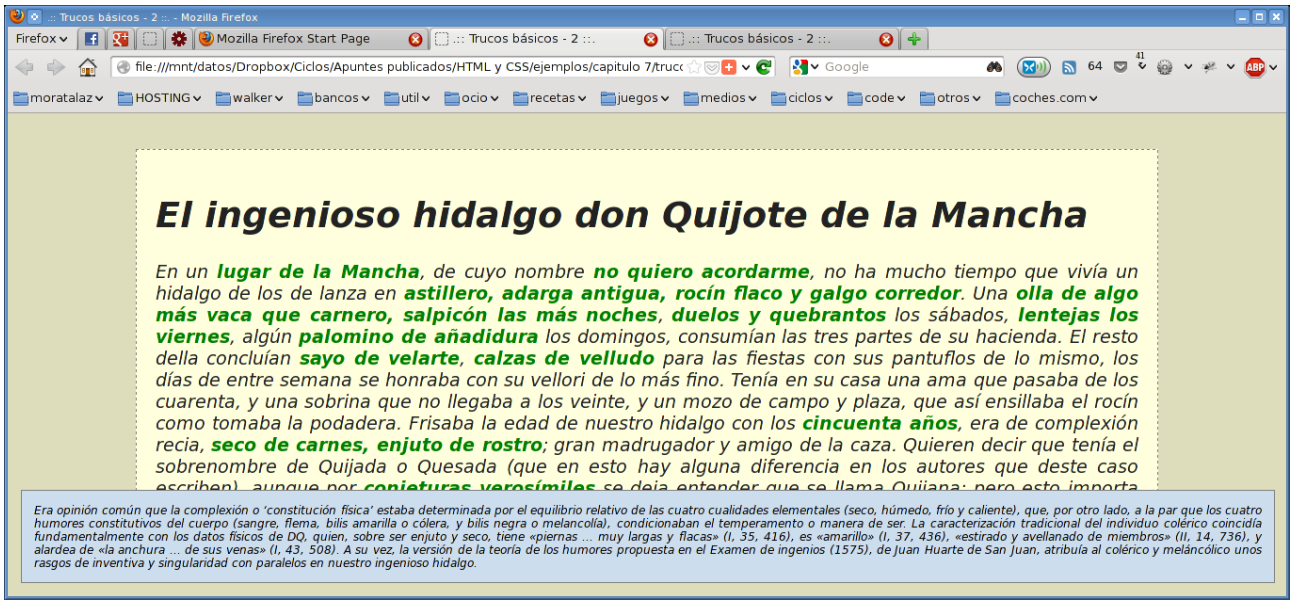


En el ejemplo 12 tenemos 4 variantes de este efecto aplicados a conseguir una letra capital destacada en un texto y un quinto caso para crear un texto destacado, pero con un poco de imaginación se nos pueden ocurrir muchas otras aplicaciones. Las claves de diseño son las siguientes:

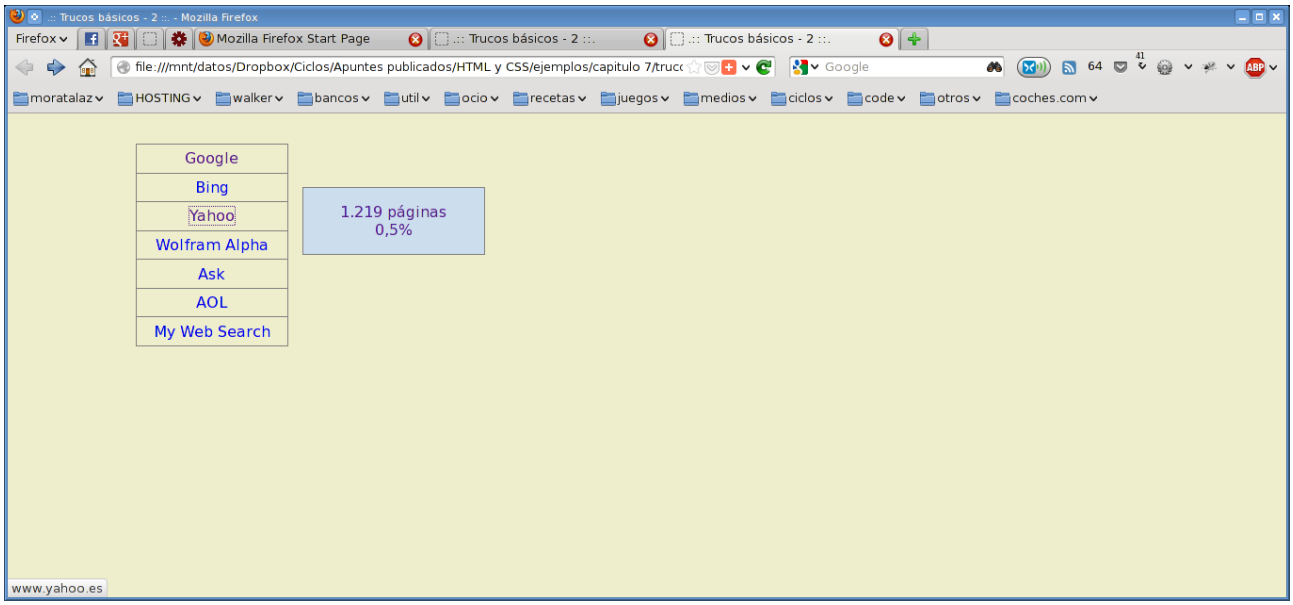
- Trabajamos siempre con dos cajas superpuestas y una de fondo. La caja en primer plano, la que pretende dar el efecto de "arrancar" un bocado a la que está en segundo plano debe de tener el mismo color que el fondo. En una de las variantes que vemos no es así: la caja en primer plano tiene el mismo color que la de segundo y es el borde de la de primer plano la que tiene el mismo color que el fondo.
- Usamos **float** en la caja de primer plano para situarla y **margin** y **padding** para establecer una separación con el texto principal.
- Cuando la caja donde está el texto principal lleva un borde usamos un margen de valor negativo en la caja de primer plano para superponerla a este y "borrarlo". El margen negativo tendrá el mismo tamaño que el borde.

Ejemplos 13 y 14 – Tooltips

El ejemplo 13 nos muestra como hacer tooltips, ventanas emergentes que nos muestran información al pasar el ratón por encima de algún punto definido de nuestra web. En el caso del ejemplo 13 estas ventanas se muestran siempre en el mismo punto, en el pié de la web.



En el ejemplo 14 se muestran en un punto dependiente del lugar por el que pasamos el ratón y que activa su visualización.



Las claves de este tipo de diseños son estas:

- Tanto el texto que "dispara" el efecto como el texto que se mostrará en el tooltip aparecerán en el HTML como contenidos de elementos diferenciados y anidados. En el ejemplo 13 usamos dos **span**:

```
<p>En un <span class="nota">lugar de la Mancha<span class="tip">lugar: no
con el valor de 'sitio o paraje', sino como 'localidad' y en especial
'pequeña entidad de población', en nuestro caso situada concretamente en
el Campo de Montiel (I, 2, 47, y 7,93), a caballo de las actuales
provincias de Ciudad Real y Albacete. Seguramente por azar, la frase
coincide con el verso de un romance nuevo.</span></span>, de cuyo nombre
```

  Y en el 14 un **span** dentro de un **a:**

```
<a href="http://www.yahoo.es/">Yahoo <span class="txtayuda">1.219
páginas<br/>0,5%</span></a>
```

- El texto que aparecerá en la nota se marca como invisible usando la propiedad **display:**

```
span.nota span.tip {
      display: none;
      font-size: .6em;
      font-weight: normal;
      color: black;
      }
```

- Al pasar el ratón sobre el elemento disparador (usaremos para ello la pseudo clase **hover**) aplicaremos un nuevo estilo a la nota. Fijaros bien en la construcción del selector porque es lo más importante de todo el efecto. En ese caso reescribimos la propiedad **display: block** para que se muestre la nota y usamos las propiedades adicionales que creamos oportunas según como y donde queramos mostrar dicha nota.

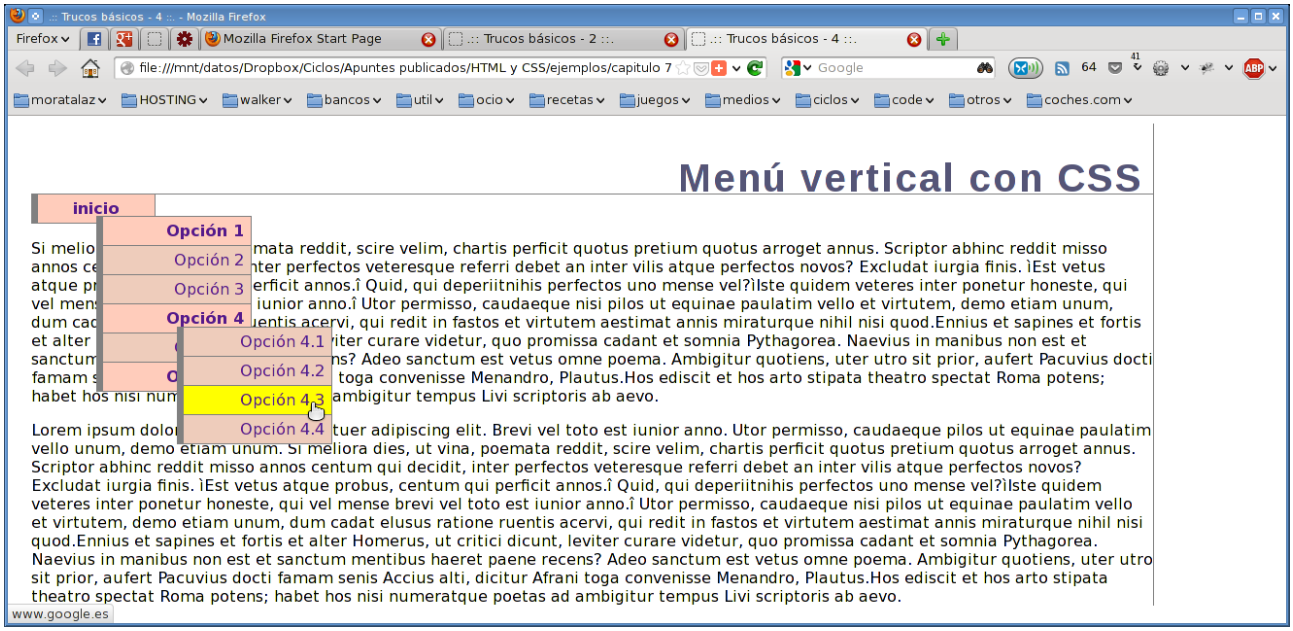
```
span.nota:hover span.tip{
	display: block;
	/* La nota irá siempre en la parte inferior de la pantalla */
	position: fixed;
	width: 96%;
	margin: 1%;
	padding: 1%;
	bottom: 0;
	left: 0;
	background: #cde;
	border: 1px solid gray;
	}
```

En el caso anterior la nota irá siempre en la parte inferior de la pantalla (ejemplo 13) y por eso usamos la propiedad **position: fixed. bottom** y **left** marcaran el lugar de aparición. En el ejemplo 14 que buscamos una posición diferente para cada nota usamos **position:absolute.**

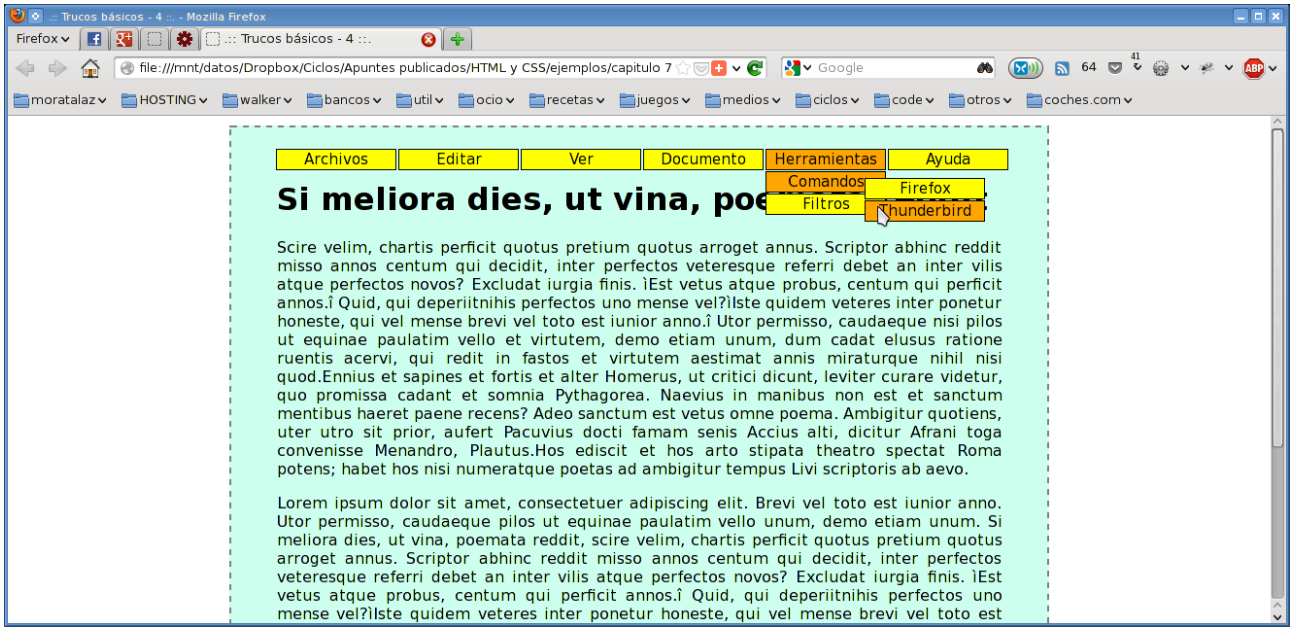
```
ul.menu li a:hover span.txtayuda {
	display: block;
	width: 10em;
	position: absolute;
	top: 0;
	left: 100%;
	margin: -1em 0 0 1em;
	padding: 1em;
	background: #cde;
	border: 1px solid gray;
	}
```

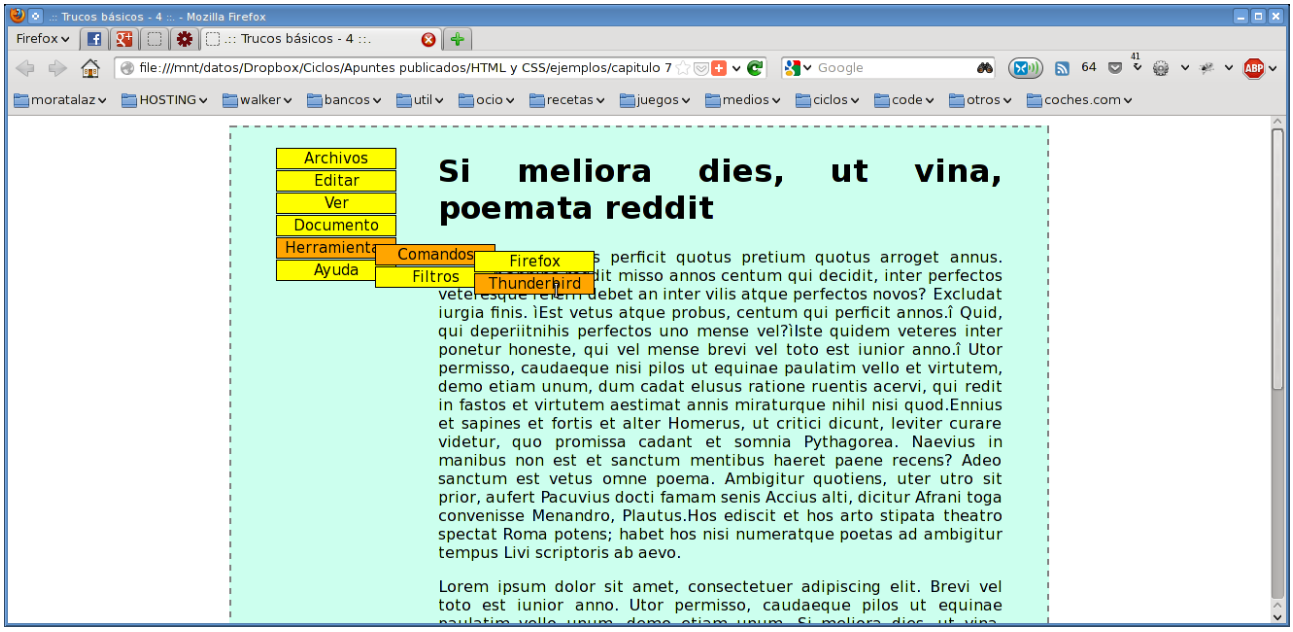
Ejemplos 15 y 16 – Menús

De los tooltips a los menús sólo hay un pequeño paso. De hecho el anterior ejemplo 14 podría considerarse ya un pequeño menú muy primitivo



El ejemplo 16 tiene además la peculiaridad de que apenenas cambiando unas líneas en el CSS podemos transformarlo en un menú horizontal o vertical a nuestro antojo. Se adjuntan las dos CSS necesarias para cambiar entre uno y otro modelo:

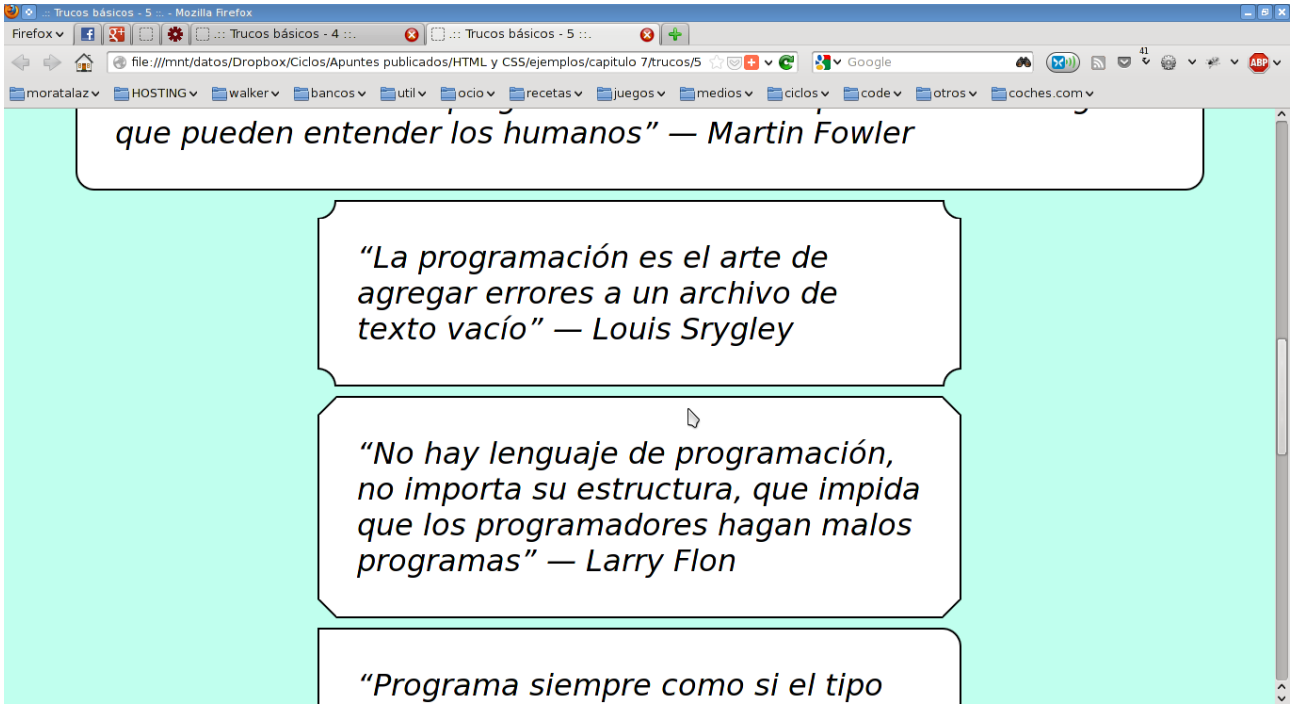




Ejemplo 17 – Esquinas redondeadas

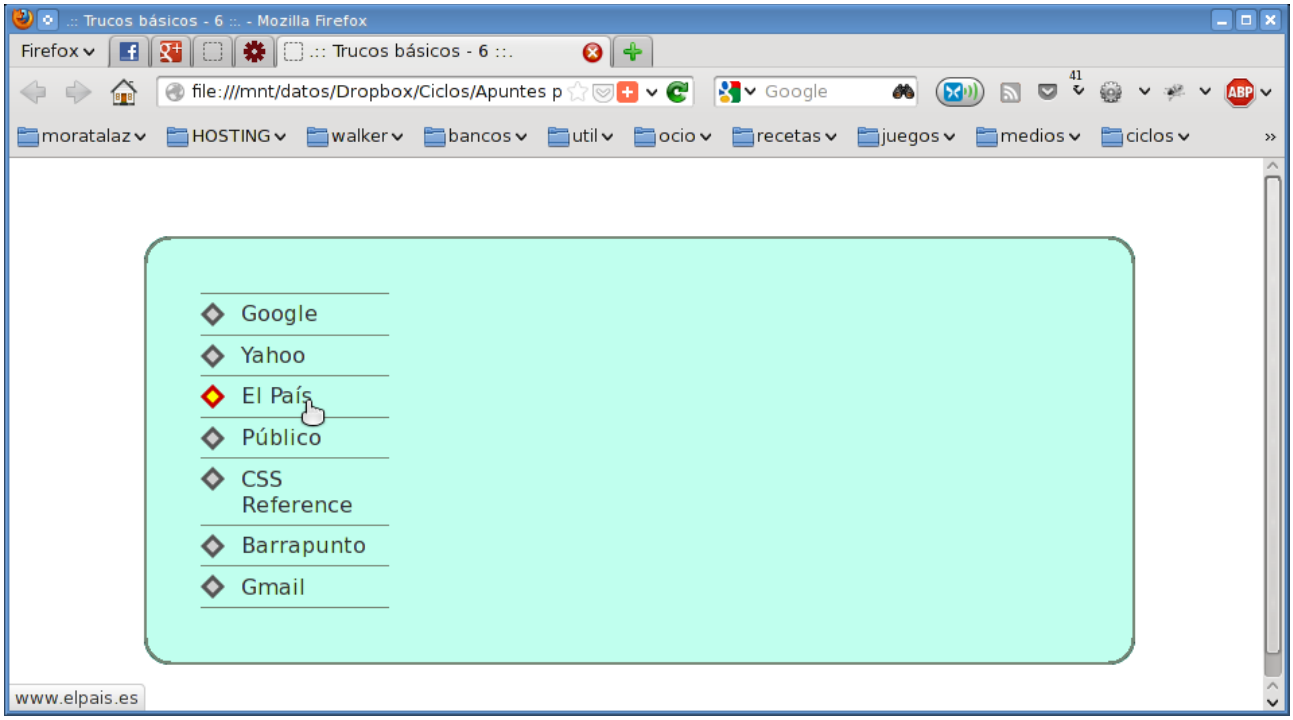
Existen muchos trucos para realizar esquinas redondeadas con HTML4 y CSS2. El más sencillo consiste en usar imágenes emplazadas dentro de una caja mediante la propiedad **position** con valor **absolute**. La caja debería de tener un borde sólido y la imagen de la esquina debería de estar especialmente preparada para integrarse con el color del fondo de la página y de la caja y, además, "morder" el borde de la caja en la que se encuentra para hacer desaparecer parcialmente las esquinas del borde cuadrado habitual de esta.

Con HTML5 tenemos otros mecanismos más fáciles para hacer bordes redondeados simples, pero si buscamos efectos más llamativos (como alguno de los que acompañan a este ejemplo) tendremos que seguir usando este método u otro similar:



Ejemplo 18 – Sprites

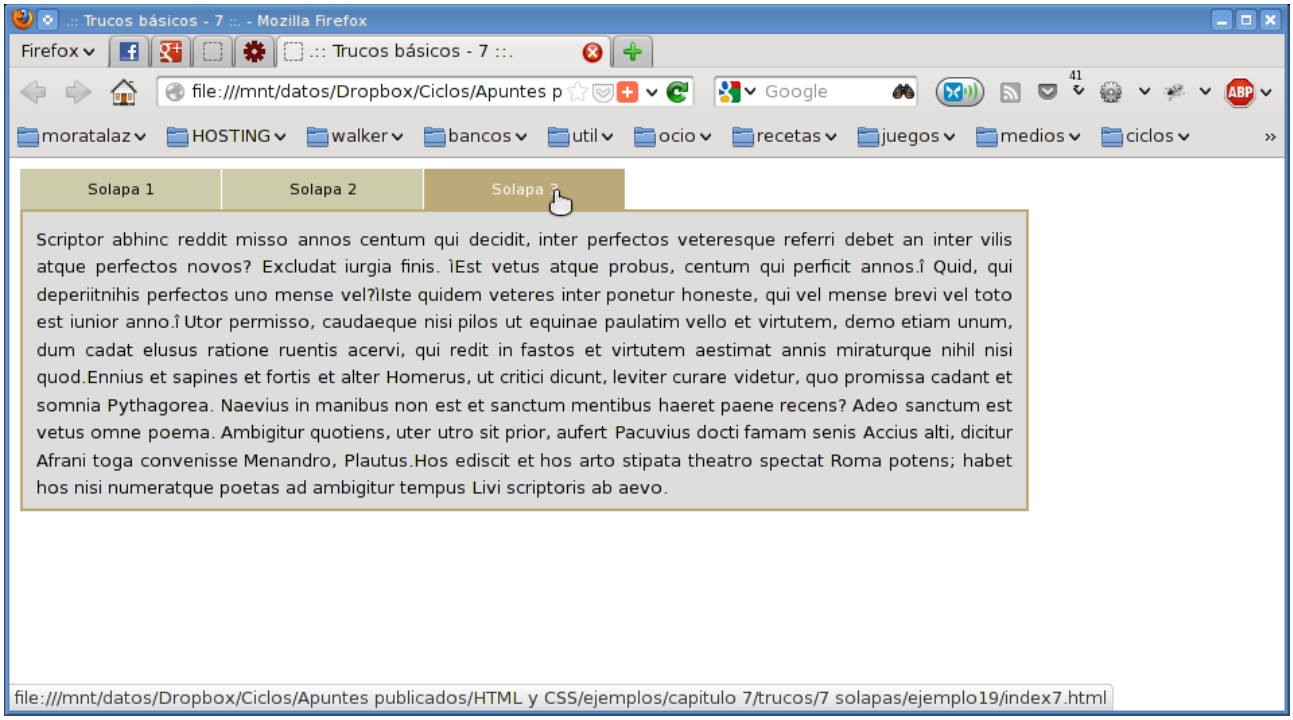
La carga de una página web implica que cada elemento individual que la compone se recuepera mediante una petición al servidor. En páginas con muchas imágenes pequeñas en las que queremos una carga excepcionalmente rápida se suele usar un truco para mejorar el rendimiento que consiste en agrupar varias de estas imágenes en una sóla y seleccionar en cada momento el trozo de imagen que queremos ver mediante la propiedad **background-position**. Se conoce a esta técnica como de "Sprites" y en el siguiente ejemplo podemos ver como usarla.



El ejemplo que se acompaña usa dos tipos de Sprites: uno con sólo tres imágenes y otro, más complejo, con seis en el que el desplazamiento de la propiedad **background-position** implica moverse tanto en el eje horizontal como en el vertical de la imagen.

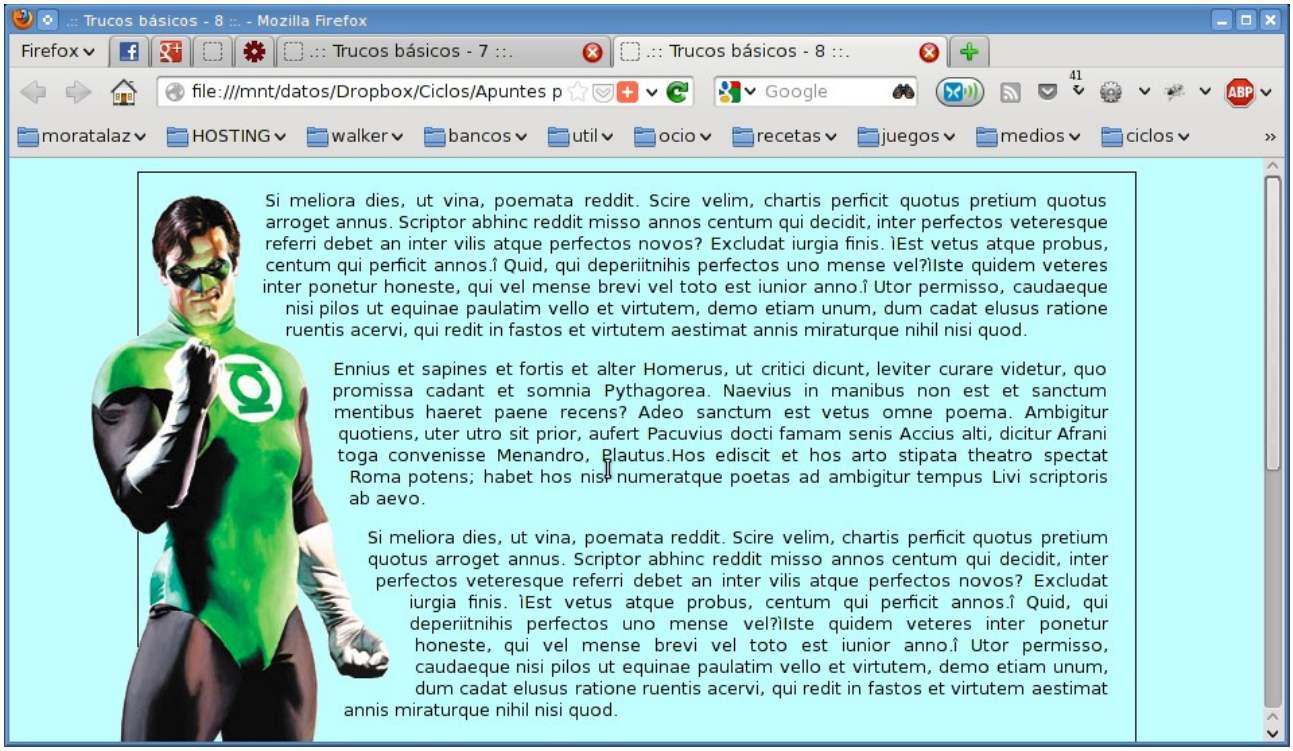
Ejemplo 19 – Solapas

El efecto de solapas, ampliamente usado en la web actual, es similar en su construcción inical al de un menú. El truco consiste en mantenemos una caja diferente por cada contenido y alternar entre su visualización o no, según corresponda, mediante la propiedad **display** y los valores de la misma **block** o **none**, respectivamente.



Ejemplos 20 y 21 – Contorneos

El contorneo de imágenes, algo típico de las revistas, es un efecto sencillo pero laborioso. La forma más sencilla de hacerlo consiste, básicamente, en "diseccionar" la imagen a contornear en rodajas de más o menos alto (según lo fino que queramos que sea el efecto) y ancho variable y, simplemente, dejar que las cajas de nuestro texto se apoyen lateralmente en dichas imágenes, bien a la izquierda, a la derecha o a ambos lados.


.:: Trucos básicos - 8 ::. - Mozilla Firefox
Firefox
.:: Trucos básicos - 7 ::.
.:: Trucos básicos - 8 ::.
file:///mnt/datos/Dropbox/Ciclos/Apuntes p
Google
moratalaz
HOSTING
walker
bancos
util
ocio
recetas
juegos
medios
ciclos
Si meliora dies, ut vina, poemata reddit. Scire velim, chartis perficit quotus pretium quotus arroget annus. Scriptor abhinc reddit misso annos centum qui decidit, inter perfectos veteresque referri debet an inter vilis atque perfectos novos? Excludat iurgia finis. ìEst vetus atque probus, centum qui perficit annos.î Quid, qui deperiitnihis perfectos uno mense vel?ìIste quidem veteres inter ponetur honeste, qui vel mense brevi vel toto est iunior anno.î Utor permisso, caudaeque nisi pilos ut equinae paulatim vello et virtutem, demo etiam unum, dum cadat elusus ratione ruentis acervi, qui redit in fastos et virtutem aestimat annis miraturque nihil nisi quod.
Ennius et sapines et fortis et alter Homerus, ut critici dicunt, leviter curare videtur, quo promissa cadant et somnia Pythagorea. Naevius in manibus non est et sanctum mentibus haeret paene recens? Adeo sanctum est vetus omne poema. Ambigitur quotiens, uter utro sit prior, aufert Pacuvius docti famam senis Accius alti, dicitur Afrani toga convenisse Menandro, Plautus.Hos ediscit et hos arto stipata theatro spectat Roma potens; habet hos nisi numeratque poetas ad ambigitur tempus Livi scriptoris ab aevo.
Si meliora dies, ut vina, poemata reddit. Scire velim, chartis perficit quotus pretium quotus arroget annus. Scriptor abhinc reddit misso annos centum qui decidit, inter perfectos veteresque referri debet an inter vilis atque perfectos novos? Excludat iurgia finis. ìEst vetus atque probus, centum qui perficit annos.î Quid, qui deperiitnihis perfectos uno mense vel?ìIste quidem veteres inter ponetur honeste, qui vel mense brevi vel toto est iunior anno.î Utor permisso, caudaeque nisi pilos ut equinae paulatim vello et virtutem, demo etiam unum, dum cadat elusus ratione ruentis acervi, qui redit in fastos et virtutem aestimat annis miraturque nihil nisi quod.

8. FORMULARIOS Y JAVASCRIPT

Javascript es un lenguaje de programación interpretado por el navegador y creado por Netscape. Su función original era mejorar la funcionalidad de los formularios en los documentos HTML. Hoy en día es el lenguaja más usado en el lado del cliente de una aplicación web para aportar a estas dinamismo e interactividad con el usuario. Continúa muy ligado al funcionamiento de los formularios pero ya no se limita a eso en exclusiva. Además, la nueva versión del estándard HTML (HTML5) incluye multiud de nuevas librerías en Javascript que amplian enormemente la funcionalidad de una página web.

El programa más sencillo escrito en Javascript sería la siguiente versión del ya clásico **"Hola Mundo"** donde se ha usado para mayor sencillez el esqueleto que usan los documentos de HTML5:

```
<!DOCTYPE html>
<html lang="es">
     <head>
          <meta charset="UTF-8" / >
          <title>Hola Mundo en Javascript</title>
     </head>
     <body>
          <script type="text/javascript" >
               alert("¡Hola Mundo!");
          </script>
     </body>
</html>
```

El código javascript es el contenido del elemento con la etiqueta **script** que aparece en el ejemplo anterior. Es lo único que aparece en el body, pero si introduces otros elementos html (títulos, párrafos, o cualquier otra cosa) diferentes antes y/o después del script verás como la ejecución del mismo interrumpe el renderizado de la página HTML. Hasta que no pulsemos el botón que aparece en el diálogo y desaparezca el mensaje, no proseguirá el dibujo de la página.

8.1. FUNCIONES Y EMPLAZAMIENTO DEL CÓDIGO JAVASCRIPT

Al igual que ocurría con los CSS, el código Javascript se puede situar de diferentes formas dentro del HTML. La forma más directa es la que hemos visto en el ejemplo anterior: directamente en el punto del cuerpo del documento donde queremos que se ejecute. Pero si las funciones son más complejas que mostrar un mensaje o queremos reutilizarlas debemos de crear funciones y emplazarlas en una ubicación diferente.

8.1.1 Funciones

Una función es una pieza de código que puede invocarse desde distintos puntos y en diferentes momentos. El código que vimos en el ejemplo anterior sólo se ejecuta cuando el navegador está dibujando la página y en el momento en que lee el trozo de código que la contiene. Hagamos algunos cambios:

```
<!DOCTYPE html>
<html lang="es">
     <head>
          <meta charset="UTF-8" />
          <title>Hola Mundo en Javascript</title>
          <script language="javascript">
               function holamundo(){
                    alert("¡Hola mundo!");
                    }
          </script>
     </head>
     <body>
          <p>texto 1</p>
          <script type="text/javascript" >
               holamundo();
          </script>
          <p>texto 2</p>
          <script type="text/javascript" >
               holamundo();
          </script>
          <p>texto 3</p>
     </body>
</html>
```

Hemos creado una función que hace exactamente lo mismo que antes y la hemos emplazado en la sección head de nuestro html. Luego la hemos invocado dos veces en distintos puntos del renderizado de nuestra página. Es evidente que se trata de un ejemplo en el que la invocación a la función ocupa lo mismo que la función en si, pero imaginémos que no es así y que se tratan de varias decenas de líneas de código. Ahora tiene más sentido ¿verdad?

Como puedes observar, en algunos puntos se usa el argumento **language="javascript**" y en otros **type="text/javascript**". El navegador entenderá cualquiera de ellos (¡la mayoría, incluso, lo entenderá si no pones ninguno!) pero lo correcto en las últimas versiones de HTML sería usar el argumento **type**.

Las funciones tienen, además, otra particularidad y es la posibilidad de que cada ejecución sea diferente gracias a la personalización que permite el paso de parámetros a las mismas. Hagamos una tercera modificación al código anterior:

```
<!DOCTYPE html>
<html lang="es">
     <head>
          <meta charset="UTF-8" />
          <title>Hola Mundo en Javascript</title>
          <script language="javascript">
               function holamundo(texto){
                    alert(texto);
                    }
          </script>
```

```
        </head>
        <body>
            <p>texto 1</p>
            <script type="text/javascript" >
                holamundo("¡Hola Mundo!");
            </script>
            <p>texto 2</p>
            <script type="text/javascript" >
                holamundo("¿He dicho ya eso de Hola Mundo?");
            </script>
            <p>texto 3</p>
        </body>
    </html>
```

Cuando un documento HTML necesita de muchas funciones en javascript o estas son utilizadas desde diferentes páginas html lo más cómodo, al igual que hacíamos con los css, es sacarlas a un fichero externo de forma que el documento html sólo queden las llamadas a las mismas. Veamos una última versión del ejemplo anterior:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <title>Hola Mundo en Javascript</title>
        <script type="text/javascript" src="func.js"></script>
    </head>
    <body>
        <p>texto 1</p>
        <script type="text/javascript" >
            holamundo("¡Hola Mundo!");
        </script>
        <p>texto 2</p>
        <script type="text/javascript" >
            holamundo("¿He dicho ya eso de Hola Mundo?");
        </script>
        <p>texto 3</p>
    </body>
</html>
```

Y en el interior del fichero func.js tendríamos esto:

```
    function holamundo(texto){
        alert(texto);
        }
```

8.2. FORMULARIOS

El elemento que sirve para identificar un formulario lleva la etiqueta **form**. Dentro de él tendremos diferentes elementos por cada objeto que queramos mostrar en el mismo. Veamos el ejemplo el formulario más sencillo:

```
<form name="form2ASI" action="respuesta.html">
     <p><label for=”nombre”>Nombre:</label>
     <input type="text" id=”nombre” name="cajanombre" /></p>
     <p><input type="submit" value="Enviar" name="botonenv"
     /></p>
</form>
```

El elemento **form** tiene un único argumento imprescindible (**action**) que indica lo que hay que hacer para procesar el funcionario. En el ejemplo anterior se le pide que cargue la página **respuesta.html** que en realidad no hace nada y sólo muestra un mensaje. Lo normal sería llamar a un script cgi o una función javascript que validara los datos, a una página php que los procese en el lado del servidor o, mucho más rudimentario, enviar los resultados por correo electrónico con un mailto (por ejemplo **action="**mailto:josemaria@uponaday.net"). En este último caso debería de aparecer también el argumento **method** y su valor debería de ser **post** (lo veremos a continuación).

El argumento **method** indica la forma en que el formulario envía los datos que el usuario ha escrito o seleccionado en él. Tiene dos valores posibles: **get** y **post.** Por defecto toma el valor **get**.

El método **get** envía los datos del formulario concatenados a la URL de la página. Los limita a 100 caracteres y es un método muy útil cuando, por ejemplo, queremos guardar el resultado en favoritos o enviárselo a alguien por correo. La URL que obtendríamos después de escribir Pepe y pulsar el botón de enviar en el ejemplo anterior debería de ser algo similar a esto:

```
http://10.0.0.1/ej2/respuesta.html?cajanombre=Pepe&botonenv=Enviar
```

El método **post** no tiene límite en su extensión y el resultado no es visible para el usuario pero, cuidado, eso no quiere decir que sea un método seguro para enviar información sensible. Ninguno de ambos métodos lo es puesto que el resultado de rellenar los datos del formulario se enviará en claro aún cuando con el método **post** no podamos verlos directamente.

En los siguientes puntos iremos viendo uno a uno los distintos elementos que pueden formar parte de un formulario y algunos de sus argumentos más usados. La mayor parte de ellos usan la etiqueta **input** e indican el tipo de objeto de que se trata mediante el valor del argumento **type**.

8.2.1. Caja de edición de texto (type=”text”)

Como su nombre indica, sirve para introducir texto con la restricción de que este no puede tener más de una línea.

```
<input type="text" name="nombre" size="20" maxlength="40"
value= "Escribe aquí tu nombre" />
```

El argumento **size** indica el tamaño en caracteres de la caja mientras que **maxlength** indica la longitud máxima que puede introducirse. **value** sirve para introducir un valor por defecto que aparecerá al cargar la página (en caso

contrario aparecerá en blanco). Cuando no queremos que el usuario edite el campo usaremos el parámetro **readonly** con cualquier valor (recuerda que en XHTML no está permitido la minimización de atributos o atributos sin valor, así que lo normal es poner **readonly="yes"** o **readonly="readonly"** aunque si ponemos **readonly="no"** o solamente **readonly** también deshabilitará la escritura en el campo). Además, pueden aparecer los argumentos **class** e **id** que nos serviran para aplicar estilos mediante CSS. El argumento **id** sirve, además, para asociar la caja de texto con una etiqueta (ver más adelante).

8.2.2 Caja de edición de contraseñas (type="password")

Similar a la anterior pero en este caso lo que escribamos sobre ella aparecerá camuflado mediante puntos, asteriscos o algo similar (según el navegador). Los argumentos que podemos usar también son los mismos que en el anterior.

```
<input type="password" name="pwd" size="12" maxlength="12" />
```

8.2.3. Etiquetas (label)

El elemento label sirve para asociar un campo de texto con un objeto de edición de forma que, por ejemplo, si el usuario pulsa con el ratón sobre la etiqueta asociada a un campo de edición de texto el cursos aparecería en modo edición en el interior de dicho campo como si el usuario hubiera pulsado en él. Se trata, por tanto, de una forma de extender la superficie de dichos objetos de forma que resulte más cómodo la cumplimentación de los formularios.

Tenemos dos formas diferentes de realizar esta asociación. La primera de ellas consiste simplemente en anidar el elemento de edición dentro del contenido del elemento label:

```
<label>Dirección:<br/><input type="text" name="direccion"
size="80" maxlength="120" /></label>
```

El segundo método consiste en identificar ambos objetos mediante el valor del argumento **for** del elemento **label** y el valor del argumento **id** del elemento de edición:

```
<label for="txtdir">Dirección:</label><br/><input type="text"
name="direccion" size="80" maxlength="120" id="txtdir" />
```

ATENCIÓN. Si usamos el primero de los métodos hay que tener mucho cuidado porque los estilos que apliquemos al elemento **label** se extenderían también al objeto de edición.

8.2.4 Radio Buttons (type="radio")

Los radio buttons son los característicos objetos redondos para elegir entre varias opciones alternativas pero con la restricción de que sólo puede seleccionarse uno de ellos, de forma que marcar cualquiera implica que se elimina la selección del que hubiéramos seleccionado previamente. Además, debería de haber una

elección obligatoria.

```
<input type="radio" name="sexo" value="hombre"
checked="checked" /> Hombre<br/>
<input type="radio" name="sexo" value="mujer" /> Mujer
```

Puesto que en un mismo formulario podemos tener diferentes grupos de radio buttons, es el valor del argumento **name** el que los mantiene asociados entre si. El argumento **checked** sirve para seleccionar la opción por defecto que aparecerá marcada al cargar la página. Además, podemos usar el argumento **disabled** si queremos que alguno de ellos no sea seleccionable.

ATENCIÓN. Cuidado con las propiedades que aparentemente funcionan pero no lo hacen. Si ponemos un argumento **readonly** en un radio (o en un check box que veremos a continuación), este cambia de aspecto pero puede seguir usándose. El argumento correcto para deshabilitarlos sería **disabled.**

El valor del argumento **value** es lo que se transmitirá en las respuesta del formulario si ese elemento se encuentra seleccionado.

8.2.5 Checkboxes (type="checkbox")

Los checkboxes son muy similares a los radio butons. Se diferencian básicamente en dos aspectos: visualmente son cuadrados y desde el punto de vista lógico deberían de permitir, si así lo deseamos, la elección de más de una opción dentro de un mismo grupo de objetos. Además, debería de permitirnos no elegir ninguno de ellos.

```
<input type="checkbox" name="vehiculo" value="motocicleta"
disabled="disabled" /> Motocicleta<br/>
<input type="checkbox" name="vehiculo" value="moto"
checked="checked" /> Moto<br/>
<input type="checkbox" name="vehiculo" value="coche" />
Coche<br/>
<input type="checkbox" name="vehiculo" value="camión" />
Camión<br/>
<input type="checkbox" name="vehiculo" value="tractor" />
Tractor</p>
```

Todo lo dicho en cuanto a los argumentos en los radio es válido también para estos.

8.2.6. Agrupación de objetos

Podemos agrupar visualmente diferentes objetos de nuestro formulario mediante el elemento **fieldset**:

```
<fieldset>
<legend>Información personal</legend>
...
</fieldset>
```

Esto, por defecto, dibuja un recuadro alrededor de los elementos que contiene. El elemento **legend** sirve para poner título a dicho recuadro.

8.2.7 Caja de edición de texto de múltiples líneas

El elemento **textarea** nos crea una caja de edición que admite más de una línea. Las dimensiones iniciales de la misma se asignan mediante los argumentos **rows** y **cols**.

```
<textarea name="comentarios" rows="8" cols="50" >Escribe tus
comentarios...</textarea>
```

No existe forma de limitar desde HTML el número de caracteres máximo que admite la caja (esto se controla habitualmente mediante javascript). El contenido del elemento, si existe, será el valor inicial que contenga la caja de edición.

Todos los navegadores modernos permiten cambiar el tamaño de estas cajas de texto visualmente mediante un control que aparece en su esquina inferior derecha. La mejor forma de evitar esto, si así lo deseamos, es en los CSS.

8.2.8 Caja de selección de ficheros

Nos abre una ventana de diálogo (cuyo aspecto depende del navegador y el sistema operativo) que nos permite seleccionar un fichero de nuestro disco duro

```
<label>Fotografía:<br/><input type="file"
name="fichero"/></label>
```

No realiza ningún tipo de validación de la elección. Eso nos corresponde hacerlo a nosotros posteriormente mediante javascript o en el servidor cuando enviemos los datos.

8.2.9 Cajas de selección (Comboboxes y Listboxes)

Existen dos cajas de selección típicas en entornos gráficos: los comboboxes son cajas con distintas opciones en las que sólo podemos seleccionar una de ellas. Además, aunque podemos desplegarla en el momento de la selección, cuando elegimos la lista se cierra y sólo permanece visible nuestra elección. Además, debería de ser obligatorio hacer al menos una elección. Los listboxes, por el contrario, nos suelen permitir elegir más de una opción simultaneamente (o ninguna) y siempre suelen estar total o parcialmente abiertos.

En HTML ambos se implementa mediante el elemento **select** y son los argumentos que usemos los que hacen que se comporte de una u otra forma. Veamos inicialmente un combobox:

```
<select name="oficina" >
     <option>Madrid</option>
     <option>Sevilla</option>
     <option selected="selected" >Alcorcón</option>
     <option>Lisboa</option>
     <option>Burdeos</option>
</select>
```

Un listbox sería de esta forma:

```
<select name="aficiones" multiple="multiple" size="4" >
     <option>Toros</option>
     <option>Cante</option>
     <option>Fútbol</option>
     <option>Juegos de Rol</option>
     <option>Ponerse tibio de beber cocacola</option>
     <option>Dormir en el sofá</option>
</select>
```

Como vemos, la diferencia la marcan dos argumentos: **size** que indica cuantos elementos de la lista serán visibles simultaneamente (por defecto es sólo 1, que es como debería de comportarse un combobox) y **multiple** que es el argumento que indica que podemos hacer más de una selección. También podemos usar el argumento **disabled** ya visto en otros controles anteriores.

En las opciones se usa el argumento **selected** para indicar si queremos que alguna de las opciones aparezca inicialmente seleccionada.

Tanto en uno como en otro podemos usar el elemento **optgroup** para agrupar las distintas opciones en categorías:

```
<select name="oficina">
     <optgroup label="En España">
          <option>Madrid</option>
          <option>Sevilla</option>
          <option selected="selected" >Alcorcón</option>
     </optgroup>
     <optgroup label="En el extranjero">
          <option>Lisboa</option>
          <option>Burdeos</option>
     </optgroup>
</select>
```

8.2.10 Botones

Todo formulario debería de tener, al menos, un botón que sirva para indicar que ya hemos terminado de cumplimentarlo y que da paso al procesado y/o envío de los datos que hemos rellenado. Este botón debe de estar identificado con el argumento **type** y el valor **submit** y el comportamiento normal al pulsarlo es invocar el enlace que hemos puesto como valor del argumento **action** del elemento **form**. Dicho esto, tenemos dos formas de implementar este botón:

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

Ambos elementos se comportarían igual y harían lo mismo. La diferencia está en que el texto que aparece en el primero es el que aparece como valor del argumento **value** (o un texto por defecto que pone el navegador si ese argumento no aparece) y que, por tanto, no podemos aplicarle estilos mientras que en el segundo botón el texto es el contenido del elemento y podemos aplicar estilos con total libertad o, incluso, poner una imagen:

```
<button type="submit" ><img src="ok.jpg" alt="Ok" /></button>
```

Ambos tipos de botones admiten el argumento **disabled**.

El segundo tipo de botones "característicos" de un formulario es uno que nos permite limpiarlo y dejar todos los valores del mismo como si la página se hubiera cargada por primera vez. Se identifica mediante el argumento type con su valor a reset y podemos, igualmente, implementarlo de ambas formas. Todo lo dicho antes sirve también en este caso:

```
<input type="reset" />
<button type="reset" ><img src="clean.jpg" alt="Limpiar"
/></button>
```

Tenemos, además, la posibilidad de introducir en nuestro formulario botones genéricos sin una acción previamente definida que posteriormente nosotros le asignaremos usando código javascript:

```
<input type="image" src="button.png" />
<input type="button" value="Botón" /><br/>
<button type="button">Botón</button>
```

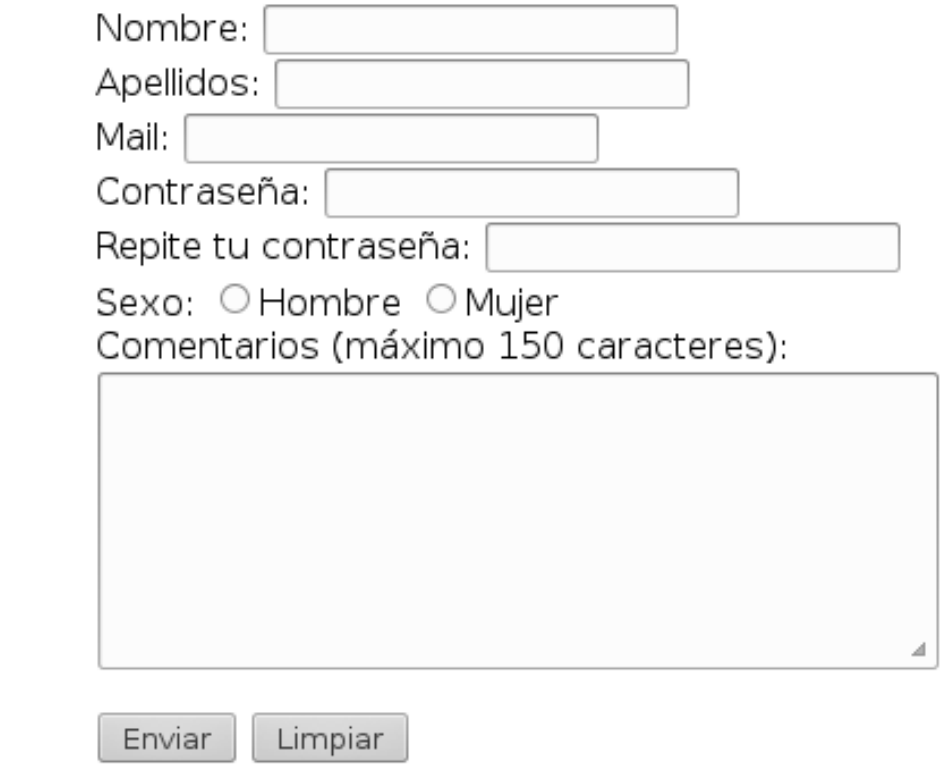
El primero no es propiamente un botón, sino una imagen pero a la que también se le asocia por defecto las mismas propiedades que a un botón con el argumento **type** con valor de **submit**. Los otros dos son botones sin ningún tipo de acción asociada a los que posteriormente se les puede aplicar alguna usando código javascript.

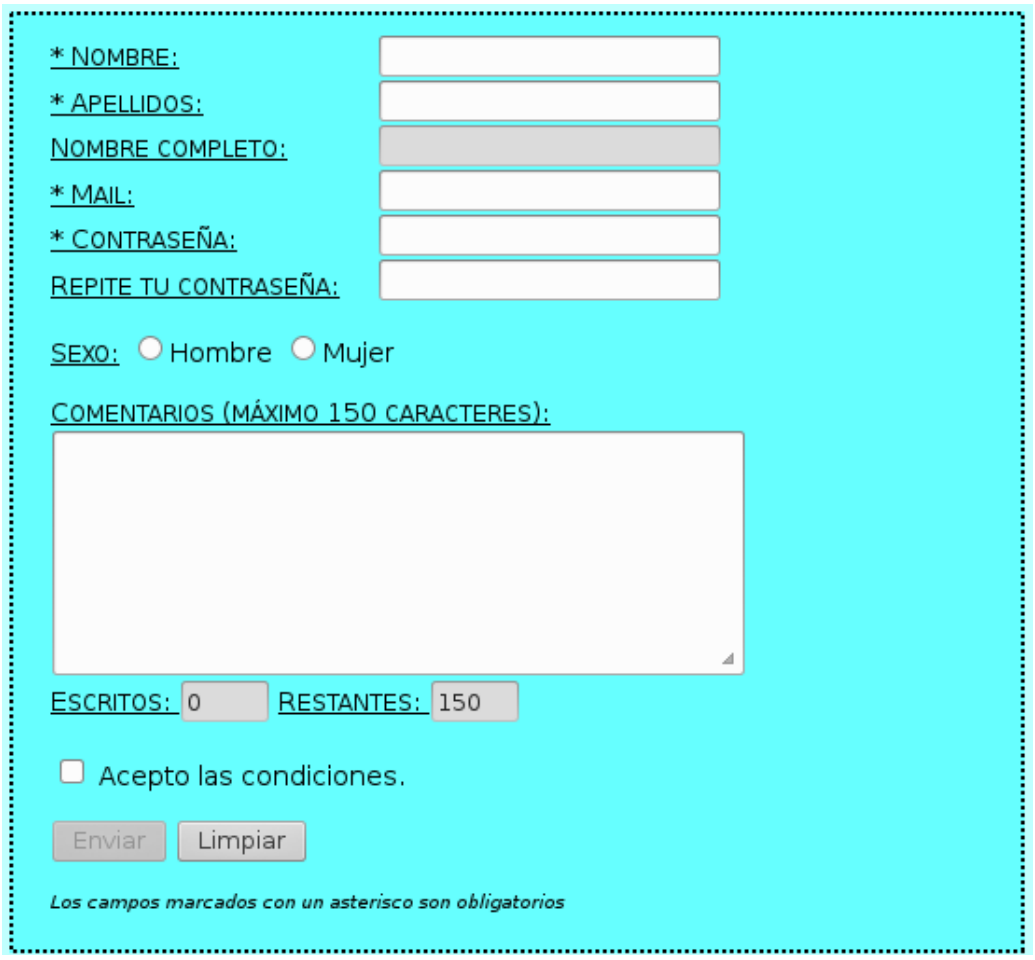
8.2.11. Nuevos elementos en HTML5

Los elementos aquí presentados son, por así decirlo, los clásicos que pueden usarse con la versión 4 de HTML. Pero una de las cosas que más ha potenciado HTML5 son los formularios apareciendo muchos elementos útiles y muy vistosos. Puedes echarle un vistazo en el apartado en el que se trata esta nueva versión del lenguaje al final de este documento.

8.3. CSS EN LOS ELEMENTOS DE LOS FORMULARIOS

Con algunas limitaciones, es posible cambiar el estilo y las propiedades de visualización de los formularios mediante CSS. No nos pararemos en detalle, pero como ejemplo debe de valer la diferencia de aspecto entre los siguientes formularios a los que sólo los separa una hoja de estilos de apenas 30 líneas (y algún elemento adicional que usaremos posteriormente para las validaciones):


Nombre:
Apellidos:
Mail:
Contraseña:
Repite tu contraseña:
Sexo: Hombre Mujer
Comentarios (máximo 150 caracteres):
Enviar
Limpiar


* NOMBRE:
* APELLIDOS:
NOMBRE COMPLETO:
* MAIL:
* CONTRASEÑA:
REPITE TU CONTRASEÑA:
SEXO: Hombre Mujer
COMENTARIOS (MÁXIMO 150 CARACTERES):
ESCRITOS: 0
RESTANTES: 150
Acepto las condiciones.
Enviar
Limpiar
Los campos marcados con un asterisco son obligatorios

8.4. VALIDACIÓN DE FORMULARIOS MEDIANTE JAVASCRIPT

Para entender correctamente la forma de manipular un formulario mediante Javascript tenemos que entender previamente los principios básicos de la programación visual orientada a eventos.

8.4.1. Métodos, eventos y propiedades

Sin meternos en demasiada profundidad, diremos que los eventos son acciones que ocurren sobre los elementos de nuestro formulario, generalmente debido a la acción del usuario: rellenar una caja de edición de texto, pulsar con el ratón, etc. Los eventos vienen predeterminados según el elemento del formulario y no podemos añadir ninguno nuevo.

Los métodos son las funciones que se invocan al producirse los eventos. Existen ya algunos métodos por defecto que vienen predefinidos con los formularios pero nosotros siempre tendremos la posibilidad de crear otros nuevos o modificar los ya existentes.

Por último, las propiedades son, por regla general (aunque no siempre), los valores de los argumentos con que están construidos los elementos del formulario y los valores de los mismos introducidos o elegidos por el usuario y que nosotros podremos manipular desde nuestros métodos.

8.4.2. Eventos del elemento form

El elemento form responde a dos eventos denominados **onSubmit** y **onReset** y que se corresponden con la pulsación de los botones de submit y/o reset respectivamente. Por defecto los métodos que se invocan en cada uno de ambos casos reciben los nombres de **submit** y **reset**, pero nosotros podemos introducir nuestros propios métodos. Esto lo hacemos introduciendo nuevos argumentos en el elemento form. Así, por ejemplo:

```
<form name="form2ASI"  onSubmit="alert('Enviando el
formulario')" onReset="alert('Limpiando el formulario')"
method="post" action="respuesta.html">
…
</form>
```

IMPORTANTE: como ves, en el interior de las funciones **alert** se usan comillas simples. Esto es para no confundir al navegador con las comillas dobles que se usan para el valor del argumento.

Hemos introducido sendas llamadas a **alert,** pero también podríamos haber llamado funciones más elaboradas construidas por nosotros.

8.4.3. Eventos de los botones

Los botones, ya sean del tipo que sean, tienen al menos tres eventos: **onClick**, **onFocus** (cuando obtienen el 'foco' de la aplicación) y **onBlur** (cuando lo pierden) pero por regla general el que nos interesa usar es casi siempre **onClick**, evento

que se genera cuando lo pulsamos.

```
<input type="submit" value="Enviar" onClick="enviar()"
name="btnenv" />
<input type="button" value="Enviar" name="btnenv"
onClick="enviar()" />
```

Estos dos botones tienen comportamientos diferentes y es muy importante que los distingamos. El primero, puesto que es del tipo **submit**, cuando sea pulsado ejecutará la función **enviar()** y luego también invocará el evento **submit()** (salvo que lo impidamos desde dentro de la función). El segundo ejecutará la función **enviar()** pero no ejecutará el evento **submit()** (salvo que nosotros lo hagamos expresamente). La diferencia es importante cuando queremos hacer una validación previa del formulario.

8.4.4. DOM, Modelo de objetos del documento

El llamado DOM es la interfaz que nos va a permitir interactuar con la tercera pata que aún no hemos mencionado: las propiedades. Se trata, ni más ni menos, de una nomenclatura especial (que a los programadores habituados a la orientación a objetos le resultará familiar) que nos va a permitir identificar los elementos del formulario y sus propiedades y poder leerlas y modificarlas.

El DOM nos va a permitir darle nombre a los objetos del formulario de forma muy sencilla. Vamos a ver un ejemplo muy sencillo. Imaginemos este botón:

```
<input type="submit" value="Enviar" onClick="enviar()"
name="botonenv" disabled="disabled" />

<input type="reset" value="Limpiar"
onClick="limpiar(this.form)" />
```

Hemos visto ya que a las funciones javascript se les pueden pasar parámetros. **this.form** es una referencia al formulario en el que aparece el elemento desde el que hacemos la referencia. La función limpiar sería así:

```
function limpiar(formulario){
     alert('Haz pulsado el botón de limpiar...');
     formulario.botonenv.disabled=true;
     }
```

Si quisieramos hacer la referencia al formulario desde el evento form, usaríamos simplemente **this**:

```
<form name="form2ASI" onSubmit=”enviar(this)”
action="respuesta.html">
     ...
</form>
```

A partir del parametro que le hemos enviado a la función identificamos al botón de enviar a través del valor de su argumento **name** y a la propiedad **disabled** mediante su nombre. De esta forma conseguimos variar su valor y hacer que cuando el formulario esté limpio el botón de Enviar aparezca deshabilitado.

Existen otras referencias útiles que podemos hacer a través de DOM. Por ejemplo,

document hace referencia a la página web en si y **document.forms.length** nos dice el número de formularios que hay en la página. Cada uno de estos formularios puede referenciarse mediante la estructura **document.form[x],** siendo **x** un número entero. Para el primer formulario de la página **x** valdría 0, para el segundo 1, etc.

El siguiente código nos pintaría dos botones. Al pulsar el primero nos diría el número de formularios de la página y al pulsar el segundo el nombre de este (el valor del argumento **name**):

```
<form name="form2ASI" action="respuesta.html">
     <input type="button" value="Botón 1" onClick="alert('En
     esta página hay ' + document.forms.length + '
     formulario')" />
     <input type="button" value="Botón 2" onClick="alert('El
     nombre del formulario es ' + document.forms[0].name)" />
</form>
```

8.4.5. Validación del formulario

La estrategia habitual a seguir a la hora de validad un formulario sigue uno de los dos patrones siguientes:

- Validamos uno a uno los elementos del formulario mientras que se rellenan (respondiendo a los eventos **onKeyDown**, **onKeyUp**, **onKeyPress**, **onChange** y/o **onBlur**)
- Validamos todos los elementos del formulario una vez que se ha pulsado el botón de enviar respondiendo al evento **onSubmit**

A veces se puede usar una estrategia combinada de ambos.

El primer método es el más sencillo. Veamos un ejemplo muy simple en el que la validación consiste sólo en comprobar que un campo no se ha quedado sin rellenar. El formulario sería este:

```
<form name="form2ASI" onSubmit="return enviar(this)"
action="respuesta.html">
     <p><label>Nombre: <input type="text" name="cajanombre"
     /></label></p>
     <p><input type="submit" value="Enviar" name="botonenv" />
     <input type="reset" value="Limpiar" /></p>
</form>
```

La función javascript que realiza la validación sería algo así:

```
function enviar(formulario){
     alert('Haz pulsado el botón de enviar...');
     if(formulario.cajanombre.value.length == 0){
          alert('No puedes dejar este campo en blanco');
          validacionOK = false;
```

```
        }
    else{
        alert('Haz escrito: '+ formulario.cajanombre.value);
        validacionOK = true;
        }
    return validacionOK;
    }
```

La otra estrategia, más dinámica pero también más complicada, consistiría en validar uno a uno los campos a rellenar y no dejar que el formulario se envíe (deshabilitando incluso el botón de Enviar) hasta que no estén todos correctos. Veamos un pequeño ejemplo también. Primero el formulario:

```
<form name="form2ASI" action="respuesta.html">
    <p><label>Nombre: <input type="text"
    onBlur="validar(this.form)" name="cajanombre"
    /></label></p>
    <p><input type="submit" value="Enviar" onClick="enviar()"
    name="botonenv" disabled="disabled" /> <input
    type="reset" value="Limpiar" onClick="limpiar(this.form)"
    /></p>
</form>
```

Y luego el código Javascript:

```
function validar(formulario){
    if(formulario.cajanombre.value.length == 0){
        alert('No puedes dejar este campo en blanco');
        formulario.botonenv.disabled=true;
        formulario.botonenv.value="¡No puedes Enviarlo
        así!";
        }
    else{
        alert('Haz escrito: ' +
        formulario.cajanombre.value);
        formulario.botonenv.disabled=false;
        formulario.botonenv.value="Enviar";
        }
    }

function enviar(){
    alert('Haz pulsado el botón de enviar...');
    }

function limpiar(formulario){
    alert('Haz pulsado el botón de limpiar...');
    formulario.botonenv.disabled=true;
    }
```

Veamos ahora la forma de validad un campo numérico. Por ejemplo un número

de teléfono. El formulario es muy similar al que hemos visto anteriormente:

```
<form name="form2ASI" onSubmit="return enviar(this)"
action="respuesta.html">
     <p><label>Teléfono: <input type="text" name="cajatlf"
     /></label></p>
     <p><input type="submit" value="Enviar" name="botonenv" />
     <input type="reset" value="Limpiar" /></p>
</form>
```

En el código javascript validaremos por un lado que la longitud sea exactamente de siete dígitos y por otro que sean todos numéros y no letras o caracteres extraños. Esto segundo lo hacemos apoyándonos en dos funciones de javascript: **parseInt** que convierte una cadena de texto a su valor numérico entero (el contenido de los formularios siempre se lee como texto aunque se trate de cifras) y la función **isNAN** que devuelve un 0 si el argumento que recibe no es un número correcto.

```
function enviar(formulario){
     alert('Haz pulsado el botón de enviar...');
     if(formulario.cajatlf.value.length != 9 ||
     isNAN(parseInt(formulario.cajatlf.value)) == false){
          alert('No es un teléfono correcto');
          validacionOK = false;
          }
     else{
          alert('Haz escrito: ' + formulario.cajatlf.value);
          validacionOK = true;
          }
     return validacionOK;
     }
```

Lógicamente se puede complicar mucho más y hacer una validación más estricta: comprobar que empiece por 9, 6 o 7, permitir el uso de prefijos internacionales, etc.

En los ejemplos que acompañan a este texto tienes algún ejemplo más, como por ejemplo como validar que la contraseña repetida dos veces coincida o la longitud máxima de un textarea (ejemplo 5) o como cambiar el contenido de un listbox en función de la selección hecha en un combobox y como pasar elementos entre dos listboxes (ejemplo 6).

8.4.6. Lista de eventos de Javascript disponibles en los formularios

A continuación tienes una tabla con los eventos que podemos usar en un formulario, los elementos a los que afectan y una breve descripción de los mismos.

Evento	Descripción	Elementos
onBlur	Cuando un elemento pierde el foco	button, checkbox, password, radio, select, text, textarea, file
onChange	Cuando un elemento pierde el foco y su valor se ha modificado	file, select, text, password, textarea, radio, checkbox
onClick	Cuando se pulsa sobre el elemento con el botón izquierdo del ratón	button, checkbox, radio, text, password, textarea
onFocus	Cuando el elemento adquiere el foco	button, checkbox, file, password, radio, select, text, textarea
onKeyDown	Cuando el elemento tiene el foco y se pulsa una tecla	text, password, textarea
onKeyPress	Cuando el elemento tiene el foco y se mantiene pulsada una tecla	text, password, textarea
onKeyUp	Cuando el elemento tiene el foco y se deja de pulsar una tecla	text, password, textarea
onMouseOut	Cuando se mueve el ratón fuera de un determinado objeto	text, password, textarea
onMouseOver	Cuando el ratón entra en un determinado objeto	text, password, textarea
onReset	Cuando se pulsa el botón de reset del formulario	form
onSelect	Cuando se selecciona un texto con el ratón	text, password,textarea
onSubmit	Cuando se pulsa el botón de submit del formulario	form

Las propiedades fundamentales que podemos alterar o consultar mediante código javascript de cada uno de los elementos de un formulario son las siguientes:

Elemento	Propiedades
form	name, action, method
button	name, value, disabled
text y password	name, value, disabled, readonly, size, length, maxlength
textarea	name, value, disabled, readonly, size, length, maxlength, cols, rows
radio y checkbox	name, value, disabled, checked, length
select	name, size, disabled, multiple, type
options	value, text
file	name, size, disabled

8.5. JAVASCRIPT FUERA DE LOS FORMULARIOS

Aunque ya hemos dicho que Javascript nació como una forma de validar de forma rápida los formularios, su uso hoy transciende de esto y podemos hacer una página que utilice javascript al margen de los formularios o no sólo con ellos. Veremos algunos ejemplos sencillos de uso en este último punto

8.5.1. Alert y otras ventanas de confirmación o entrada de datos

Hemos usado durante todo el capítulo la función **alert** que nos muestra una ventana con un texto y un botón de Aceptar. Esta función paraliza la ejecución hasta que el usuario no pulsa el único botón que aparece en ella.

Pero Javascript dispone de otros dos tipos de ventanas más útiles según para que las necesitemos. Vamos a verlas.

La ventana **confirm** muestra dos botones (OK y Cancelar) y devuelve la elección del usuario para que actuemos en consecuencia como una variable booleana: **true** si hemos pulsado OK y **false** si hemos pulsado Cancel:

```
respuesta = confirm("Pulsa un botón");
if (respuesta == true){
     alert("Haz pulsado el botón de OK");
     }
else{
     alert("Haz pulsado el botón de Cancel");
     }
```

La ventana **prompt** nos muestra una caja de texto y devuelve el valor que hemos escrito en ella. Además, muestra dos botones y devuelve el valor **null** si hemos salido pulsando el botón de Cancelar en lugar del de OK:

```
texto = prompt("Escribe tu nombre, por favor");

if(texto == null){
     alert("Haz pulsado el botón de cancelar");
```

```
        }
    else{
        if(texto.length!=0){
            alert("Hola " + texto + ", buenos días \n ¿Cómo
            estás?");
            }
        else{
            alert("Buenos días aunque no quieras decirme tu
            nombre... :-(");
            }
        }
```

8.5.2. Lista de eventos de Javascript disponibles de forma genérica

Los eventos que podemos capturar en una página web y asociarlos a un método javascript y que no tienen relación directa con los elementos de un formulario son estos:

Evento	Descripción
onAbort	Cuando se aborta la carga de una imagen
onDblClick	Cuando se hace doble click con el botón izquierdo del ratón
onLoad	Cuando la página termina de cargarse completamente
onMouseDown	Cuando se pulsa un botón cualquiera del ratón
onMosueMove	Cuando se mueve el ratón
onMove	Se mueve la ventana
onResize	Se cambia el tamaño de la ventana

Además, alguno de los eventos que se usan en los formularios pueden tener también aplicación fuera de ellos.

Veamos un trozo de código con ejemplos sencillos de aplicación:

```
<body onLoad="alert('La página se ha cargado completamente');"
onResize="alert('Haz cambiado el tamaño de la página');">
    <h1 onMouseOver="alert('Haz pasado el ratón sobre el
    titular');" >Titular de la página</h1>
    <p onClick="alert('Haz hecho click en el primer
    párrafo');" >Primer párrafo de texto de la página</p>
    <p onDblClick="alert('Haz hecho doble click en el primer
    párrafo');" >Segundo párrafo de texto de la página</p>
</body>
```

En el ejemplo 8 que acompaña a este texto tienes un ejemplo más complicado donde se ve como cambiar el estilo CSS de un elemento de la página a través de

un método que se lanza al capturar el envento de hacer Click en un ratón.

8.5.3. Los objetos document y window

El modelo DOM nos proporciona dos objetos denominados **document** y **window** con una amplia colección de propiedades y atributos que nos permiten interactuar de forma fácil con características interesantes del navegador y el documento HTML.

Por ejemplo, el objeto **window** nos permite obtener las dimensiones en píxeles de la superficie útil del navegador. La siguiente función nos muestra estas dimensiones en una ventana de alerta:

```
function muestraDimensiones(){
     alert("Ancho: " + window.innerWidth + " píxeles.\nAlto: "
     + window.innerHeight + " píxeles.");
     }
```

Tienes una buena referencia de las posibilidades que te ofrece el objeto **window** aquí:

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

La principal utilidad del objeto **document** es que nos permite obtener una referencia a cualquier elemento HTML de la página para interactuar con él a través del valor de su atributo **id** que, como ya sabes, debería de ser único. Lo hacemos mediante la función **getElementById().** La siguiente función obtiene la referencia de un objeto HTML cuyo valor del atributo id es **caja1** y modifica su estilo por uno llamado **estilo2**. Dicho estilo debería de existir en los CSS del documento y ser aplicable al objeto en cuestión (puedes consultar el ejemplo 8 que acompaña a este documento para ver el código completo)

```
function cambiar_estilo(){
     objeto = document.getElementById("caja1");
     objeto.className="estilo2";
     }
```

Puedo, incluso, cambiar el contenido de un elemento a través de la propiedad innerHTML:

```
function cambiaTitular1(titular){
     titular.innerHTML="Nuevo Texto para el Titular";
     }
```

Para ver más propiedades del objeto **document** puedes consultar esta página:

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

Y si quieres ampliar conocimientos o ver más ejemplos sobre cualquier otra cosa de javascript, aquí tienes una completa referencia de todas las funciones disponibles:

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

9. HTML5

HTML5 es la nueva revisión del lenguaje HTML definido por la W3C. Se encuentra aún en fase beta experimental y, por tanto, hay que tratar con mucho cuidado la introducción de sus nuevas características puesto que muchos navegadores, sobre todo si son relativamente antiguos, no contemplan aún estos cambios o no lo hacen en su totalidad. Una de las grandes novedades es que, por primera vez, la definición de HTML y XHTML para esta nueva versión (HTML5 y XHTML5) se han desarrollado en paralelo. XHTML5 es, básicamente, HTML5 con las restricciones de sintaxis que ya planteamos cuando presentamos XHTML1

9.1. ESTRUCTURA DE UN DOCUMENTO HTML5 O XHTML5

La estructura de un documento con html5 o xhtml5 es mucho más sencilla que en versiones anteriores y, además, idéntica para ambos:

```
<!DOCTYPE html>
<html lang="es">
	<head>
		<meta charset="UTF-8" />
		<link rel="stylesheet" href="estilos.css" media="all" />
		<title>Html5</title>
	</head>
	<body>

	</body>
</html>
```

Observa que ya no es necesario usar el argumento type en las etiquetas link que declaran el uso de hojas de estilo (tampoco en las que declaran un fichero con funciones javascript). Tampoco existen diferencias en los argumentos entre HTML5 y XHTML5. De hecho, podemos escribir documentos denominados "políglotas" que puedan ser interpretados como uno o como otro indistintamente y sin ningún problema observando las siguientes condiciones:

- El DOCTYPE no es obligatorio en XHTML pero si en HTML, así que debemos de ponerlo. Además, HTML no es sensible a mayúsculas o minúsculas pero XHTML si, así que hay que ponerlo tal y como aparece en el ejemplo de aquí arriba.
- Debemos de usar la sintaxis de HTML5, y observar todas las restricciones de XHTML5 que ya habíamos visto, a saber:
  - Los elementos deben de cerrarse siempre
  - Los elementos sin contenido deben de usar la sintaxis de autocerrado siempre.
  - La apertura y cierre de los elementos anidados no debe de solaparse.
  - Los valores de los atributos deben de ir siempre entre comillas simples o dobles
  - Los nombres de etiquetas y atributos deben de ir siempre en minúsculas

- No está permitido usar un atributo sin valor

9.2. CAMBIOS EN LAS ETIQUETAS

Lo primero a reseñar en HTML5, un lenguaje de marcas basado en un conjunto de etiquetas, debería de ser los cambios sufridos en el conjunto de estas. Aparecen 25 nuevas etiquetas, hay ligeros cambios en 7 de las antiguas y desaparecen 16.

Etiquetas eliminadas

Las etiquetas eliminadas, de las cuales nosotros no deberíamos de preocuparnos demasiado porque unas las hemos evitado expresamente desde el principio de este curso y otras ni siquiera las hemos presentado, son las siguientes:

acronym, applet, basefont, big, center, dir, font, frame,frameset, isindex, noframes, s, strike, tt, u, xmp

Nuevas etiquetas para la organización del documento.

Dentro de las nuevas etiquetas, el primer grupo que nos interesa ver es el de aquellas que introducen una mayor información semántica a la estructura del documento. Se trata de **header, nav, footer, aside, section** y **article**. Se trata de etiquetas que identifican las secciones más habituales que existen en la gran mayoría de las páginas web y que hasta ahora identificábamos usando la etiqueta div en función con algún id o class, por ejemplo ** ** o ** **.

Estas etiquetas no hacen nada especial y no introducen ninguna información de estilo. Su único propósito es proporcionar información semántica a nuestras páginas de cara, sobre todo, a facilitar su lectura y la valoración del contenido por parte de los buscadores. Pensemos, por ejemplo, en una página web que se titule "Carnicerías de calidad en Sevilla" y otra en cuyo pié de página aparezca el nombre de su autor: "Javier Carnicería Sevilla". Google no debería de tratar a ambas al mismo nivel cuando alguien realice una búsqueda de carnicerías en Sevilla. Estas etiquetas le proporcionan, a él y al resto de buscadores, una forma clara de valorar el contenido de una web según donde se presente cada texto.

Solventando los problemas de versiones antiguas de Internet Explorer

Cuando contemplamos el uso de versiones antiguas del Microsoft Internet Explorer podemos tener un problema con estas etiquetas. El resto de navegadores nos permite "inventarnos" etiquetas y aplicarles estilos, pero Explorer si no reconoce una etiqueta como válida la ignora. Esto hace que una versión antigua de Firefox nos permita aplicar estilos a la etiqueta header, aunque no la conozca, pero Explorer no. Para solucionar esto usamos dos recursos: la posibilidad de incluir comentarios condicionales en Internet Explorer y las librerías html5shiv.

Los comentarios condicionales tienen esta forma:

```
<!--[if IE]>
      <p>Esto sólo se verá en un Internet Explorer</p>
<![endif]-->
```

Es fácil ver que, según la sintaxis HTML que conocemos eso es un comentario y no se verá, pero Explorer lo reconoce como una secuencia especial y si nos lo mostrará. Tenemos otras formas más complejas para detectar una versión concreta de Explorer, o una mayor o menor que una dada:

```
<!--[if IE 6]>
      <p>Sólo en Explorer 6</p>
<![endif]-->
<!--[if lt IE 9]>
      <p>Sólo en versiones de Explorer menores a la 9</p>
<![endif]-->
<!--[if lte IE 7]>
      <p>Sólo en versiones de Explorer menores o iguales a la 7</p>
<![endif]-->
<!--[if gt IE 6]>
      <p>Sólo en versiones de Explorer mayores a la 6</p>
<![endif]-->
<!--[if gt IE 8]>
      <p>Sólo en versiones de Explorer mayores o iguales a la 8</p>
<![endif]-->
```

O, incluso, darle otra vuelta de tuerca a la sintaxis y mostrar código que sólo se verá en navegadores que no sean el Internet Explorer:

```
<!--[if !IE]> -->
      <p>Esto sólo se verá si no estás con un Explorer</p>
<!-- <![endif]-->
```

De esta forma ya podemos mostrar unas etiquetas en Explorer y otras en otros navegadores de forma relativamente fácil, pero existe un método más rápido y limpio de solucionar el problema de las nuevas etiquetas de organización: la librería html5shiv nos permiten que el Explorer aplique estilos a etiquetas que no conoce. Para que funcionen basta con incluirlas en la sección head de nuestra web y, si queremos, podemos incluirlas con un comentario condicional como los vistos para que sólo se carguen cuando realmente las necesitamos:

```
<!--[if lt IE 9]>
      <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

9.3. MEJORAS EN LOS FORMULARIOS: NUEVOS TIPOS DE INPUT

Tal vez una de las mejoras más significativas sea la gran riqueza que HTML5 añade a los formularios con objeto de que podamos construir con facilidad aplicaciones web. Son 13 los nuevos tipos disponibles para la etiqueta **input** además de algún argumento extra añadido a los ya existentes.

El argumento **required**, válido para casi todos los tipos de objetos permite, marcar como obligatorio un campo de forma que no se pueda ejecutar la función submit si no está completo y dando, además, un efecto visual que cada navegador podrá personalizar.

El argumento **autofocus**, que debería de usarse sólo en un objeto por formulario, selecciona el campo del mismo que tomará el foco de forma automática una

vez cargada la página.

El argumento **autocomplete** puede tomar dos valores (**on** o **off**) e indica al navegador si puede hacer sugerencias al usuario dependiendo de lo que haya escrito en anteriores ocasiones en ese mismo campo.

El argumento **placeholder**, válido en todos los campos de texto, nos muestra un texto con instrucciones sobre que debemos de cumplimentar en el mismo.

El argumento **wrap**, existente en versiones anteriores a HTML4 pero desaparecido en esta, vuelve en HTML5. Es válido en los input de tipo textarea y nos permite indicar si queremos que los saltos de línea que el usuario escriba en ellos se tranmsitan (**wrap="hard"**) o no (**wrap="soft"**) que es la opción por defecto.

datalist es una nueva etiqueta que puede asociarce a los campos de texto y que nos permite mostrar al usuario una lista de sugerencias de forma muy similar a la que, por ejemplo, Google utiliza cuando escribimos en su caja de búsquedas. Se usa en conjunto con la etiqueta option de forma muy parecida a la que utilizamos en un campo select:

```
<input type="text" name="provincia" list="provincias" />
<datalist id="provincias">
      <option value="Sevilla"></option>
      <option value="Madrid"></option>
      <option value="Cuenca"></option>
      <option value="Málaga"></option>
</datalist>
```

La identificación entre el input y el datalist se hace a través de los argumentos **list** e **id** cuyos valores deben de ser iguales.

Por último, el argumento **pattern** permite especificar una expresión regular que debe de cumplir lo que pongamos en el campo para que el formulario sea validado. Veamos un par de ejemplos:

```
<input type="text" name="codpostal" pattern="[0-9]{5}" />
<input type="text" name="nif" pattern="[0-9]{8}[A-Za-zÑñ]{1}" />
```

En el primero el patrón especifica que el campo sólo validará si consta de cinco números seguidos sin ningún caracter adicional. En el segundo caso, el campo debe de ser rellenado con ocho números seguidos de una letra, ya sea esta en mayúsculas o minúsculas. Para aprender más sobre como construir expresiones regulares más complejas puedes consultar el siguiente enlace:

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

Y una nota adicional acerca de los patrones: si el campo no está marcado como requerido mediante el argumento antes visto sería posible dejarlo en blanco y eso no cumpliría con el patrón especificado. Así que si se quiere obligar no olvides también este argumento.

Los tipos nuevos introducidos a los input son **email, url, number, date, month,**

week, time, datetime, datetime-local, search, color, range y **tel.** Proporcionan controles especializados de diferentes peticiones bastante cotidianas con validación automática. Veamos algunos ejemplos con los argumentos más habituales de cada uno de ellos (aparte de los ya vistos: **required, disabled, readonly**, etc. y que funcionarían perfectamente con la mayoría de ellos).

```
<input type="email" name="email" />
<input type="url" name="web" />


<!-- funcionan en opera y en chrome -->

<input type="number" name="edad" min="18" max="100" step="1" value="18" />
<input name="cursor" name="unidades" type="range" min="3" max="12"
value="6" step="3" />


<!-- funcionan sólo en opera -->

<input type="date" name="nacimiento"  />
<input type="month" name="alta" />
<input type="week" name="vacaciones" />
<input type="time" min="07:00" max="23:00" step="3600" value="09:00" />
<input type="datetime" name="fecha1" />
<input type="datetime-local" name="fecha2" />
<input type="color" name="color" value="#c0ffee" />


<!-- No implementados aún por ningún navegador -->

<input type="search" name="busqueda" /> <input type="button"
value="Buscar" />
<input type="tel" name="telefono" />
```

Intimamente relacionado con los formularios, aunque mucho más universal y sin dependencia directa con estos, tenemos el nuevo agumento **contenteditable** que convierte, cuando su valor es **true**, a casi cualquier elemento, ya sea de bloque o de linea, en editable de forma directa por el usuario:

```
<h1 contenteditable="true">Titular editable</h1>
<p contenteditable="true">Este párrafo también es editable</p>
<div contenteditable="true">Y este bloque div lo mismo</div>
<p>Pero, incluso <strong contenteditable="true">estas negritas</strong>  o
este <span contenteditable="true">texto con un span</span> que son
elementos de línea pueden ser editables</p>
```

9.4. MODERNIZR

Modernizr es una librería escrita en javascript y que nos sirve de ayuda en la transición a HTML5. Su funcionamiento es muy simple: nos permite "interrogar" al navegador del usuario que está viendo nuestras páginas sobre si entiende determinadas características de HTML5 o CSS3 y, en ese caso, elegir lo que vamos a mostrarle. Veamos un ejemplo muy sencillo y explicamos con él como instalarlas

y hacerlas funcionar de forma básica.

Lo primero que debemos de hacer es descargar las librerías de modernizr de la web http://www.modernizr.com/ y copiar el fichero js en el directorio de nuestro proyecto. Tenemos dos opciones: o descargarnos las librerías completas de la versión de desarrollo o generar una librería a medida que sólo contenga las comprobaciones de las etiquetas y atributas que usamos. Esta segunda opción será mucho más ligera y acelerará la carga y ejecución de nuestro proyecto.

En segundo lugar tenemos que incluir una llamada a la librería en el **head** de nuestro html:

```
<!DOCTYPE html>
<html lang="es">
	<head>
		<meta charset="UTF-8" />
		<script src="modernizr-2.5.3.js"></script>
		<link rel="stylesheet" href="estilos.css" media="all" />
		<title>Ejemplo de uso de Modernizr</title>
	</head>
	<body>
	</body>
</html>
```

Dos indicaciones en este paso: en primer lugar el nombre de la librería puede cambiar según lo que nos descarguemos de la web. Revísalo antes de incluirlo en tu proyecto. En segundo lugar, la carga de la librería de modernizr debe de hacerse antes de la carga de las hojas de estilo si queremos hacer comprobaciones también en estas.

Ahora ya lo único que tenemos que hacer es, desde nuestro documento, interrogar a Modernizr si el navegador es capaz de interpretar determinadas etiquetas y/o argumentos y actuar en consecuencia. Podemos ver una demostración sencilla introduciendo este código en el body de nuestro ejemplo anterior:

```
<script>
	if (Modernizr.input.required) {
		alert("Tu navegador soporta el atributo required");
		}
	else {
		alert("Tu navegador no soporta el atributo required");
		}

	if (Modernizr.inputtypes.range) {
		alert("Tu navegador soporta el input type=range");
		}
	else {
		alert("Tu navegador no soporta el input type=range");
		}
</script>
```

Pero lo que nosotros realmente queremos, la mayoría de las veces, no es advertir de nada al usuario sino crear una página 'alternativa' de forma que el no se de cuenta de nada aunque, lógicamente, pierda algunas de las funcionalidades que ofrece HTML5. Para ello usamos la función document.write de javascript que nos permite crear parte del documento HTML en el mismo momento de su ejecución y dependiendo de las indicaciones que nos haga Modernizr. Veamos una alternativa más útil al código anterior con estas indicaciones:

```
<script>
      if (Modernizr.input.required) {
            document.write("<label>Nombre: <input name='nombre'
            type='text' required='required' /></label><br/>");
            }
      else {
            document.write("<label>Nombre: <input name='nombre'
            type='text' /></label><br/>");
            }
      if (Modernizr.inputtypes.range) {
            document.write("<label>Edad: <input name='edad' type='range'
            min='1' max='120' value='18' step='1' /></label><br/>");
            }
      else {
            document.write("<label>Edad: <input name='edad' type='text'
            size='3' maxlenght='3' value='18' /></label><br/>");
            }
</script>
```

Otra posibilidad es cargar una hoja de estilos diferente de forma condicional a los elementos que nuestro navegador soporte usando la función load de Modernizr. A continuación vemos un ejemplo sencillo:

```
<script>
      Modernizr.load({
           test: Modernizr.input.placeholder && Modernizr.inputtypes.date,
           yep : 'estilos1.css',
           nope: 'estilos2.css'
      });
</script>
```

En el ejemplo anterior "testeamos" dos elementos HTML5 (el tipo de input date y el argumento placeholder) y en función de que estén soportados, o no, cargamos una hoja de estilos diferente. Lógicamente, se trata de una función que encontraremos más útil a la hora de probar el soporte de elementos de CSS3, pero puesto que aún no los hemos tocado lo vemos aquí y más adelante haremos algún otro ejemplo.

Dos apuntes importantes sobre esto. La función load de Modernizr no viene en la versión de desarrollo de la libería y, por tanto, para usarla debemos de generar una librería desde la opción de producción incluyendo esta utilidad (está marcada por defecto en el apartado de extras) y todas las comprobaciones

adicionales de tipos, argumentos o propiedades que vayamos a hacer.

NOTA IMPORTANTE: Lo visto aquí se trata de un uso muy básico de estas librerías que no incluye todas las posibilidades y comprobaciones previas que deberíamos de hacer en un proyecto profesional. Para más información visita la página de documentación:

http://www.modernizr.com/docs/

9.5. ETIQUETAS MULTIMEDIA

HTML5 incluye dos etiquetas, **audio** y **video** para facilitar la interpretación directa por parte del navegador de contenido multimedia y **embed** para incrustar archivos en formato flash.

Audio

La etiqueta audio nos permite insertar un archivo de audio en nuestra página web sin necesidad de usar flash ni un plugin de javascript como hacíamos hasta ahora. Será el propio navegador el que realice la reproducción y dependemos tanto de que la etiqueta esté implementada como de los codecs de audio que sea capaz de interpretar. El uso básico es este:

```
<audio src="archivo_audio">Su navegador no soporta la etiqueta
audio</audio>
```

El argumento **src**, que define dónde se encuentra el archivo de audio a reproducir, es el único obligatorio. El contenido de la etiqueta sólo será visible si el navegador no la soporta, pero si lo que no soporta es el codec de audio, no será capaz de reproducir el archivo pero no nos informará sobre ello. Así escrito, el archivo no se reproducirá ni se hará visible de ninguna forma. Existen dos argumentos adicionales, que pueden usarse a la vez, para permitir que el archivo se reporduzca: **controls** y **autoplay**. El primero mostrará un pequeño cuadro de mandos con un botón de play/pausa, sonido on/off y una barra de progreso de la duración del archvio. El segundo hará que el sonido se reproduzca al cargar la página. Ambos son parámetros sin valor por lo que en XHTML tendremos que usarlos de esta forma:

```
<audio src="archivo_audio" controls="controls" autoplay="autoplay" >Su
navegador no soporta la etiqueta audio</audio>
```

El argumento **loop** (**loop="loop"**) reproduciría en bucle una y otra vez el sonido. Por último, el parámetro **preload** nos permite cargar el archivo de audio al mismo tiempo que la página y no cuando el usuario le da a play para evitar demoras. Este último tiene tres valores posibles: **none**, **metadata** (para precargar sólo los metadatos del archivo o **auto**).

Formatos de audio

Son cuatro los codecs de audio diferente a los que, en la actualidad, se da soporte (de forma irregular) en los diferentes navegadores que implementan esta etiqueta: **ogg, mp3, wav** y **acc (ó m4a).** Ningún navegador los implementa todos ni ninguno de ellos es implementado por todos los navegadores. Al menos no por el momento. Para solucionar esto tenemos la etiqueta **source** que nos permite especificar diferentes archivos de forma que el navegador tenga elección:

```
<audio controls="controls" >
      <source src="finaldetrayecto.ogg" />
      <source src="finaldetrayecto.mp3" />
      <source src="finaldetrayecto.wav" />
      Etiqueta no soportada
</audio>
```

Cuando el navegador procese estas etiquetas se quedará con el primer archivo que sea capaz de reproducir. Por tanto el orden en que aparezcan es importante. Si nosotros preferimos que un navegador que pueda reproducir tanto el .ogg como el .mp3 se quede con el primero, este debería de aparecer también por delante. La etiqueta source tiene un parametro opcional para especificar el tipo de archivo y el codec que usa el mismo, por ejemplo **type="audio/mpeg"** o **type="audio/ogg; codecs=vorbis"**

Vídeo

La etiqueta video, su uso y problemática es muy similar a la de audio. El caso más sencillo de uso sería este:

```
<video src="archivo_video"> Su navegador no soporta la etiqueta
video</video>
```

En este caso al usar la nueva etiqueta sin especificar autoplay ni controls si que se reserva espacio para el primer fotograma del mismo pero la única forma de reproducirlo sería pulsando con el botón derecho y usando el menú contextual que nos suele ofrecer el navegador. El atributo **src** sigue siendo, pues, el único obligatorio. Los parámetros **controls, autoplay, loop** y **preload** también existen y su uso es idéntico a los vistos para la etiqueta audio. Los nuevos parámetros disponibles para la etiqueta video son los siguientes:

width y **height** nos permiten controlar las dimensiones de visualización del vídeo. Si no usamos ninguno de estos parámetros el navegador nos mostrará el vídeo con sus dimensiones originales. Si especificamos uno cualquiera de ambos, pero no el otro, se ajustará el no especificado para respetar la proporcionalidad de la imagen.

El atributo **poster** nos permite definir una imagen estática que aparecerá mientras que el vídeo se carga o hasta que el usuario decide comenzar su reproducción pulsando el botón de play. Si no se especifica aparecerá el primer fotograma del vídeo. Un ejemplo con algunos de estos argumentos:

```
<body>
      <video src="video_ogv.ogv" controls="controls" width="700"
      poster="thatsallfolk.jpg"> Su navegador no soporta la etiqueta
      video</video>
</body>
```

Formatos de vídeo

Con los formatos de vídeo tenemos el mismo problema que con los de audio: ningún navegador implementa todos los reconocidos ni existe ninguno de ellos que sea soportado por todos los navegadores. **Ogv**, **H.264**, y **WebM** son las posibilidades con las que jugamos en este caso. Ogv es el más libre, equivalente al formato ogg de audio. H.264 y WebM son dos formatos propietarios, el primero empujado por Apple y el segundo propiedad de Google. La etiqueta source se puede usar también en este caso para proporcionar alternativas al navegador:

```
<video controls="controls" >
      <source src="video_ogv.ogv" />
      <source src="video_mp4.mp4" />
      <source src="video_webm.webm" />
      Su navegador no soporta la etiqueta video.
</video>
```

Archivos de Flash

HTML5 incorpora también la posibilidad de incrustar directamente archivos de animación flash a través de la etiqueta **embed**:

```
<embed src="animacion_flash.swf" />
```

src es, al igual que en audio y video, el único atributo obligatorio. **height** y **width** también son válidos y funcionan de igual forma que hemos descrito en la etiqueta vídeo.

9.6. DIBUJO MEDIANTE LA ETIQUETA CANVAS

La etiqueta canvas define un objeto que actuará como un lienzo dibujable mediante la llamada a funciones de javascript, proporcionando un potente elemento de dibujo y diseño capaz de hacer competencia directa con flash. El siguiente ejemplo dibuja un rcuadrado verde:

```
<canvas id="myCanvas" height="300" width="500">
      Tú navegador no soporta la etiqueta canvas.
</canvas>

<script>
      var canvas=document.getElementById("myCanvas");
      var ctx = canvas.getContext("2d");
      ctx.fillStyle = "#00ff00";
      ctx.fillRect(0,0,100,100);
</script>
```

El contenido de la etiqueta canvas sólo se visualizará si nuestro navegador no la soporta. canvas define un lienzo transparente susceptible de ser "rellenado" mediante funciones de dibujo de un API de javascript incorporado al navegador. En las dos primeras líneas del script del ejemplo anterior obtenemos primero una referencia al objeto y en el segundo elegimos el API que usaremos para el dibujo en función del argumento de la función. El contexto **"2d"** es el más extendido por el momento aunque existe una especificación muy avanzada que permite el dibujo en 3D usando el argumento **"webgl"** y las librerías de dibujo OpenGL, la competencia libre de las DirectX:

http://www.khronos.org/registry/webgl/specs/latest/

Las dos líneas siguientes del ejemplo dibujan un rectángulo de 100 píxeles de ancho por 100 de alto (un cuadrado, en realidad) que empieza en la coordenada 0,0 (en la esquina superior derecha del canvas) y que se rellena de color verde (#00ff00 en hexadecimal).

Veamos algunas de las funciones básicas del dibujo mediante canvas:

- **fillStyle** y **strokeStyle** son variables que nos permiten elegir el color, patrón o gradiente de color que usaremos para relleno o línea de dibujo, respectivamente. Para seleccionar un color podemos usar los mismos métodos que en CSS (por ejemplo, **"#c0ffee"**, **"#fff", "orange"** o **"rgb(255,160,23)"**) pero siempre entre comillas como si fuese una cadena. Para seleccionar un gradiente o un patrón asignamos el objeto donde está definido el mismo (veremos como hacerlo mas adelante).
- **fillRect** y **strokeRect** son funciones que nos permiten dibujar un rectángulo especificando cuatro parámetros: el punto de su vértice superior derecho y las dimensiones de sus lados **(x, y,ancho, alto)**.
- **clearRect** (x,y,ancho,alto) limpia el contenido del rectángulo especificado. Para reiniciar el canvas completo podemos hacer un clearRect con las dimensiones totales del mismo o reasignar el valor de ancho o de alto del mismo, como por ejemplo así:

```
cv.width = cv.width;
```

- **moveTo (x,y)** mueve el "pincel" de dibujo al punto de coordenadas especificadas.
- **lineTo (x,y)** dibuja una línea desde el punto actual done se encuentra el pincel hasta el punto especificado en las coordenadas. En realidad las líneas no se dibujan sobre el lienzo del canvas sino que se almacenan en una especie de buffer temporal que sólo se volcará realmente en la página web cuando llamemos a la función **stroke().** El color usado será el que hayamos definido en **strokeStyle.**
- **lineWidth** es una propiedad que define, en pixeles, el ancho de las líneas

trazadas.

- **font** es una variable que nos permite definir un tipo de letra. Recibe una cadena con las características de este, de forma similar a la de la propiedad font de CSS. Por ejemplo "**bold italic 30px monospace**"
- **fillText("texto", x, y)** es una función que escribe el texto que recibe como primer parámetro a partir del punto definido mediante su segundo y tercer parámetro. El color que escogerá será el último que hayamos definido mediante **fillStyle**.
- **strokeText("texto", x, y)** hace lo mismo pero sólo dibujando la línea de contorno del texto sin rellenarlo.

El trabajo con gradientes es bastante más delicado. En el siguiente ejemplo vemos como definir un gradiente y rellenar un rectangulo con él:

```
gradiente = cv.createLinearGradient(250, 50, 350, 250);
gradiente.addColorStop(0, "#0f0");
gradiente.addColorStop(1, "#f00");
cv.fillStyle = gradiente;
cv.fillRect (250, 50, 100, 200);
```

La primera línea crea el objeto que contendrá el gradiente. Este será lineal e irá desde el punto 250,50 hasta el 350,250. Las coordenadas están refereridas a la superficie del canvas. Las dos siguientes líneas definen los colores inicial y final del gradiente. Por último, usamos el objeto gradiente para definir la variable fillStyle y dibujamos un rectángulo que se rellenará con el mismo. Nota que el rectangulo coincide exactamente con la superficie del gradiente pero no tendría porque ser así. El gradiente lineal puede crearse mediante una diagonal (como en el ejemplo) pero también mediante una línea vertical u horizontal. También podemos crear un radiante cónico mediante la siguiente función:

```
createRadialGradient(x0, y0, r0, x1, y1, r1)
```

Los tres primeros parámetros definen un círculo y los tres últimos otro. El gradiente se creará dibujando una superficie cónica entre ambos círculos.

El dibujo de arcos se realiza mediante la función **arc** que recibe seis parámetros: las coordenadas del centro, el radio, el punto de inicio y de final del arco en radianes y una variable booleana que define si el arco se trazará en sentido horario (false) o antihorario (true). El 0 de los grados estaría situado en el punto de las tres en una esfera horaria. La constante **Math.PI** también nos resultará útil para expresar los grados en radianes. Una vez definido el arco o el círculo podemos llenarlo o dibujar sólo su línea mediante las funciones **fill** o **stroke**, respectivamente. Veamos un ejemplo:

```
cv.beginPath();
cv.arc(150, 300, 70, 0, 2 * Math.PI, true);
cv.fillStyle = "#9cf";
cv.fill();
```

```
cv.beginPath();
cv.arc(100, 150, 200, 0, Math.PI, false);
cv.lineWidth = 10;
cv.strokeStyle = "black";
cv.stroke();
```

La función **beginPath** sirve para indicar a canvas que comenzamos un nuevo trazado independiente de lo que hayamos hecho anteriormente y es válida tanto al trazar arcos como líneas rectas.

Los efectos de sombreado se consiguen mediante las propiedades shadowOffsetX, shadowOffsetY, shadowBlur y shadowColor que definen, respectivamente, el desplazamiento horizontal y vertical de la sombra en píxeles, el "difuminado" y el color de la misma. Una vez ajustados estos valores todos los elementos tendrán su sombra hasta que los pongamos a cero. Un ejemplo:

```
cv.shadowOffsetX = 10;
cv.shadowOffsetY = 10;
cv.shadowBlur = 5;
cv.shadowColor = "black"
cv.fillRect(400,250,75,75);
```

Tenemos disponibles tres funciones que nos van a permitir hacer transformaciones sobre los elementos dibujados: **translate(x,y)**, **rotate (angulo)** y **scale(x,y)**. La primear desplaza un elemento al punto indicado por las coordenadas que enviamos como parámetros. La segunda va a rotar el elemento el número de grados (en radianes) en el sentido de las agujas del reloj. Por último la tercera permite cambiar la escala de un objeto indicando el factor de escalado horizontal y vertical como parámetros. Todas ellas son acumulativas y se aplican sobre todos los elementos que se vayan a dibujar desde el momento en que se indican hasta que se desactivan. Un ejemplo:

```
for(j=0; j<6; j++){
      cv.fillRect(400,50,75,75);
      cv.translate(50, 50);
      cv.rotate(Math.PI/16);
      cv.scale(0.7, 0.7);
      }
```

Canvas nos va a permir también interactuar con las coordenadas del ratón para realizar gráficos interactivos. En el siguiente ejemplo del libro diveintohtml5 puedes ver un ejemplo y pulsando sobre el enlace etiquetado como Rules of play te explican lo fundamental de su funcionamiento:

http://diveintohtml5.ep.io/examples/canvas-halma.html

En los siguientes enlaces tienes todas las funciones, su descripción y algunos ejemplos de lo que puedes usar como contenido de la etiqueta canvas usando el

API 2d:

http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html
http://www.w3schools.com/html5/html5_ref_canvas.asp

9.7. GEOLOCALIZACIÓN

La geolocalización consiste en situar exactamente mediante coordendas geográficas (longitud y latitud) al usuario que está viendo nuestra página web en un plano. HTML5 introduce esta posibilidad de forma nativa mediante su propio API y sin necesidad de usar librerías externas como hasta ahora. Los datos para proporcionar dichas coordenadas pueden tomarse de tres formas diferentes, que pueden ser complementarias y que, en escala de menor a mayor precisión son las siguientes:

- A través de la dirección IP pública de acceso a Internet proporcionada por nuestro proveedor.
- Por triangularización a través de la localización de los puntos de acceso wifi a los que estemos conectados.
- A través de un GPS integrado en nuestro equipo.

El navegador, si soporta esta función, tomará las coordenadas de geolocalización del usuario de la forma más precisa posible y en base a estos métodos (según los que tenga disponibles) y siempre después de pedir autorización al usuario. Veamos el body de un pequeño ejemplo:

```
<ul>
	<li>Latitud: <span id="latitud">Favor de esperar...</span></li>
	<li>Longitud: <span id="longitud">Favor de esperar...</span></li>
	<li>Precisión: <span id="precision">Favor de esperar...</span></li>
	<li>Altura: <span id="altura">Favor de esperar...</span></li>
	<li>Precisión en la altura: <span id="precisionAltura">Favor de
	esperar...</span></li>
</ul>

<script>
	if (navigator.geolocation) {
		navigator.geolocation.getCurrentPosition(function(position) {
			var latitud = position.coords.latitude;
			var longitud = position.coords.longitude;
			var altura = position.coords.altitude;
			var precision = position.coords.accuracy;
			var precisionAltura = position.coords.altitudeAccuracy;

			document.getElementById("latitud").innerHTML = latitud;
			document.getElementById("longitud").innerHTML =
			longitud;
			document.getElementById("altura").innerHTML = altura;
			document.getElementById("precision").innerHTML =
			precision;
```

```
                        document.getElementById("precisionAltura").innerHTML =
                        precisionAltura;
                  }     );
            }
            else {
                  alert("Su navegador no soporta la geolocalización");
                  }
      </script>
```

El script, que es donde usamos las funciones, tiene tres partes bien diferenciadas. En primer lugar comprobamos si nuestro navegador soporta la función de geolocalización interrogando a la variable **navigator.geolocation.** Si el resultado es negativo mostramos una alerta y, en caso contrario, llamamos a una función (**navigator.geolocation.getCurrenPosition**) a través de la cual obtenemos la latitud y la longitud (**positions.coords.latitude** y **position.coords.longitude**), la precisión en estas medidas según el método empleado para obtenerlas de los tres vistos en metros (**position.coords.accuracy**), la altura (**position.coords.accuracy**) y la precisión en la medida de esta (**position.coords.altitudeAccuracy**). La altura y la precisión en ella sólo estará disponible si contamos con GPS y será cero en otro caso. La última parte del script muestra los datos obtenidos en nuestra página HTML usando la función Javascript **document.getElementById** que nos permite sustituir el contenido de cualquier etiqueta (**innerHTML)** simplemente indicando el ID de la misma.

Existen otros dos parámetros más además de los vistos en este ejemplo: **position.coords.heading** que te proporciona la posición en grados respecto al norte y **position.coords.speed** que, tomando como referencia la última posición y el tiempo transcurrido entre una y otra, nos proporciona la velocidad a la que nos movemos en metros por segundo.

Tienes otro ejemplo más completo que enlaza con el API de google maps para mostrar las coordenadas en un plano aquí:

http://benwerd.com/lab/geo.php

9.8. MÁS COSAS

Existen muchas otras novedades en HTML5: funcionalidad de arrastrar-soltar (drag-drop) nativa en el navegador, un sistema mejorado de comunicaciones bidireccionales con el servidor, cookies que permitiran almacenar más información, etc. Muchas de estas nuevas funcionalidades, al igual que ocurre con las antes vistas de geolocalización o de canvas, todas ellas se proporcionan mediante el uso de un API externo realizado en javascript. Para seguir aprendiendo y ampliando conocimientos, así como novedades y cambios de última hora, la mejor referencia es la que proporciona el comité del W3C:

http://dev.w3.org/html5/spec/Overview.html

Y si quieres algo más didáctico puedes consultar el libro gratuito diveintohtml5:

http://diveintohtml5.ep.io/

10. CSS3

Los cambios en CSS3 han sido amplios y significativos para conseguir páginas web más límpias y atractivas con un menor esfuerzo para el codifidador. Repasaremos algunas de las más importantes pero no olvidemos que ni el estándar está cerrado ni lo que hay es compatible con muchos de los navegadores que se siguen usando ampliamente.

10.1. LOS NUEVOS SELECTORES DE CSS3

Pseudo-elementos

Los pseudo-elementos existentes en CSS2.1 se mantienen invariables en CSS3 salvo un pequeño detalle: ahora se distinguen más claramente de las pseudo-clases usando :: delante en lugar de : como hasta ahora. Así, por ejemplo, usaremos p::first-letter en lugar de p:first-letter como hasta ahora. Además, se añade un nuevo pseudo-elemento llamado **::selection** que permite aplicar estilos diferenciados al texto que el usuario ha seleccionado mediante el ratón o el teclado.

Pseudo-clases

El principal cambio en los selectores de CSS3 se encuentra aquí, donde se han añadido 14 selectores nuevos.

El selector **:root** representa al elemento raíz del documento HTML que siempre ha de ser la propia etiqueta html. No existe ninguna diferencia apreciable entre usarl este o la etiqueta **body**, salvo que **:root** tendría una mayor especificidad.

:empty es un selector que demarca a todos los elementos vacíos de contenido y sin hijos. Por ejemplo, tengamos el siguiente HTML:

```
<p>A paragraph.</p>
<p></p>
<p>Another paragraph.</p>
```

El párrafo central, vacío y sin hijos, sería el único que respondería a este selector:

```
p:empty{ width:200px; height: 40px; background: red; }
```

El selector **:target** sirve para seleccionar el elemento activo, o sea, aquél enlace interno realizado mediante la etiqueta **a** y que acabas de visitar. Veamos este código:

```
<p><a href="#enlace1">Ir al punto 1</a></p>
<p><a href="#enlace2">Ir al punto 2</a></p>
<p><a name="enlace1"></a>La, la, la...</p>
<p><a name="enlace2"></a>Bra, bra, bra...</p>
```

Y usemos esta regla combinada con el selector **:before**

```
:target:before{
      content: "Estás aquí -> ";
      }
```

:last-child viene a acompañar al selector **:first-child** que ya conocíamos y selecciona a un determinado elemento siempre y cuando sea el último hijo de un padre cualquiera. Por ejemplo, supongamos el siguiente código:

```
<body>
      <p>Primer párrafo del body</p>
      <h1>Titular</h1>
      <div>
            <p>Primer párrafo del div</p>
            <p>Segundo párrafo del div</p>
            <p>Tercer párrafo del div</p>
      </div>
</body>
```

El selector **p:first-child** seleccionaría a los párrafos marcados en azul (el primer hijo del body y el primer hijo del div), mientras que el selector **p:last-child** seleccionaría al párrafo marcado en rojo (el último hijo del div).

Siguiendo en esta línea, **:only-child** seleccionaria a un elemento que fuese hijo único de su padre. **:nth-child(n)** seleccionaría al elemento que está en la posición n dentro de la descendencia de su padre. El primer elemento sería el 1 y, por lo tanto, **p:nth-child(1)** sería equivalete a **p:first-child**. Por último, **:nth-last-child(n)** seleccionaría al elemento que está en la posición n pero empezando a contar por el último y no por el primero.

:nth-child(n) permite un uso avanzado configurando adecuadamente su argumento único. Podemos cambiar la letra n por las palabras claves **odd** o **even** y, en ese caso, seleccionaríamos a todos los elementos impares o pares, respectivamente. También podemos usar expresiones como **3n** (los elementos múltiplos de 3) o usar tres reglas diferentes (**3n+1**, **3n+2** y **3n+3**) para alternar entre tres estilos diferentes en una lista de elementos iguales.

Similares a estos, tenemos un conjunto de selectores que se comportan de igual forma pero con los elementos de un determinado tipo. Son **:first-of-type**, **:last-of-type**, **only-of-type**, **:nth-of-type(n)** y **:nth-last-of-type(n).**

Nuevos selectores para uso con formularios

Existen, además, tres pseudo-clases nuevas especificamente pensadas para su uso con formularios que son **:enabled**, :**disabled** y **:checked**. Como su nombre indica, aplican a los elementos de tipo input que cumplen con lo especificado en el nombre del selector.

El selector :not()

Existe una última pseudo-clase que trataremos aparte por su especial

importancia. Se trata del selector :not() que sirve para aplicar estilos a aquellos elementos que no cumplen lo especificado ente paréntesis. Veamoslo con algunos ejemplos. La siguiente regla se apliaría al contenido de cualquier div que no fuese de la clase "comun"

```
div:not(.comun){color: red; }
```

La siguiente regla aplicaría a todos los elementos strong que están dentro de un div pero que no están dentro de un p

```
div *:not(p) strong{color: red; }
```

En la siguiente, lo combinamos con un selector de atributo para seleccionar todos los input salvo aquellos de tipo password.

```
input:not([type="password"]){color: red; }
```

Podemos usar también una lista como argumento del not. Lo siguiente se aplicará a todos los titulares de nivel h2 salvo a aquellos que pertenecen a las clases "politica" y "economia":

```
h2:not(.politica, .economia){color: red; }
```

Y, por supuesto, se puede combinar con cualquier otro selector que hemos visto antes: descendientes, etc.

Nuevos selectores de atributos

CSS3 añade, además, tres nuevos selectores de atributos que nos permiten un control más fino. El primer selector que veremos, que usa el símbolo distintivo **^** sirve para seleccionar a etiquetas con un atributos cuyo valor empieza por una cadena. En el ejemplo, a aquellas etiquetas **a** con un atributo **href** cuyo valor empieze por mailto:

```
a[href^="mailto:"]{ color: blue; }
```

El segundo selector de atributo, que usa el símbolo distintivo **$** selecciona a aquellas etiquetas con un atributo concreto cuyo valor termina en determinada cadena. Por ejempo, las etiquetas **a** cuyo atributo **href** terminan en html:

```
a[href$=".html"] { color: yellow; }
```

Por último, se incluye un selector que usa el símbolo * que selecciona a aquellas etiquetas con un atribuyo cuyo valor contenga en cualquier posición la cadena especificada. En el siguiente ejemplo, seleccionaría a cualquier imagen cuyo título contenga la palabra importante.

```
img[title*="importante"] {border: 3px red dashed; }
```

Todos los selectores de atributo pueden combinarse entre si y, por supuesto, con otros selectores como siempre. Por ejemplo, el siguiente selector cuadraría con las

imágenes cuyo titulo empieze por Estadísticas y contenga la palabra importante:

```
img[title^="Estadísticas"][title*="importante"] {border: 3px red dashed; }
```

Selector general de elementos hermanos

El último de los nuevos selectores de CSS3 amplía las posibilidades del selector de hermanos existente en la versión anterior. Recordemos que el selector de hermanos (que usaba el símbolo **+**) sólo se "activaba" cuando los dos elementos de la regla eran hermanos e iban uno a continuación de otro de forma inmediata. El nuevo selector usa el símbolo **~** y se activa cuando ambos elementos son hermanos y el segundo va después del primero pero sin importar que sea de forma inmediata. Veamos un ejemplo sencillo:

```
<h1>Titular 1</h1>
<h2>Titular 2</h2>
<p>Párrafo 1</p>
<div>
      <h2>Titular 3</h2>
</div>
<h2>Titular 4</h2>
```

Una regla que use el selector de hermanos "clásico" (por ej. h1+h2 {color: red;}) sólo seleccionaría el Titular 2, mientras que usando el selector general h1 ~ h2 {color: red; } seleccionaría el Titular 2 y el Titular 4.

9.2. LAS NUEVAS PROPIEDADES DE CSS3

Son muchas las propiedades añadidas en CSS3 y muy irregular la implantación en los diferentes navegadores hasta el punto de que cada uno de ellos en muchos casos ha desarrollado por su cuenta algunas de las implementaciones añadiendo un prefijo distintivo que es **-o-** para ópera **-moz-** para firefox, **-webkit-** para chrome, safari y otros que usen el Webkit y **-ms-** para Internet Explorer. Veremos ejemplos de esto en varios de los apartados siguientes.

Gradientes y una nueva paleta de colores

CSS3 adopta (con ligeros cambios) la paleta de colores stándard de X11 que consta de 147 elementos que podemos usar mediante un nombre característico. La lista de colores X11 puedes consultarla aquí:

http://en.wikipedia.org/wiki/X11_color_names#Color_name_charts

En CSS3 los usaremos siempre en minúsculas y eliminando los espacios. Por ejemplo, el color Blanched Almond pasaría a ser **blanchedalmond**.

Además, se incorpora la posibilidad de realizar degradados de forma que ya no necesitamos usar imágenes estáticas para estos fines. Existen dos tipos de gradientes: lineales y radiales. Veamos un ejemplo de cada uno:

```
background-image: linear-gradient(bottom, rgb(107,168,104) 13%,
rgb(204,137,180) 57%, rgb(164,219,245) 83%);

background: radial-gradient(center, circle, white, #7DA5CD);
```

Desglosemos las particularidades de estas reglas. En primer lugar, cualquiera de ellas puede ir en la propiedad background o background-image. En segundo, el valor de la propiedad debería de empezar por **linear-gradient** o **radial-gradient** según el tipo de gradiente deseado. En tercer lugar, ambas deben de llevar un número de parámetros separados por comas que serán un mínimo de tres en el caso del lineal y de cuatro en el caso del radial. Veamos estos parámetros.

En el caso del lineal, el primer parámetro define la dirección del degradado definiendo su inicio y pudiendo usar para ello una o dos de las palabras clave **top**, **bottom**, **right** o **left**. A continuación se enumeran los colores del degradado (dos, al menos pero pudiendo ser más). Cada color se define de cualquiera de las formas admitidas habitualmente en CSS (con un valor hexadecimal, una palabra distintiva o una función rgb) y pudiendo ir acompañado de un porcentaje que define la suavidad de la trancisión entre uno y otro color.

En el caso del radial, el primer parámetro también define la dirección pero además de las palabras claves anteriores tenemos otra, **center**, que en caso de usarse debería de ir sola. El segundo parámetro define la forma del degradado y puede ser **circle** o **ellipse**. A continuación irían los colores del degradado que siguen el mismo formato explicado anteriormente y deberían de ser al menos dos.

Se trata esta de una de las propiedades de que hablábamos cuyo estándard no se encuentra generalizado, así que si queremos que los principales navegadores lo entiendad debemos repetir la regla con los diferentes prefijos distintivos. Por ejemplo así:

```
p.degradadoradial{
      background: radial-gradient(center, circle, white, #7DA5CD);
      background: -o-radial-gradient(center, circle, white, #7DA5CD);
      background: -moz-radial-gradient(center, circle, white, #7DA5CD);
      background: -webkit-radial-gradient(center, circle, white, #7DA5CD);
      background: -ms-radial-gradient(center, circle, white, #7DA5CD);
      }
```

En el caso de los gradientes lineales, tenemos una herramienta web muy útil que nos permite generar las reglas de forma automática aquí:

http://gradients.glrzad.com/

Fondos múltiples

CSS3 incorpora la posibilidad de incluir fondos múltiples en una misma caja sin necesidad de tener que superponer una caja por cada imagen. La posición, y características de cada imagen (repetición, etc.) se define también de forma independiente para cada una de ellas. Veamos un par de ejemplos. Ahora definir imágenes independientes para personalizar las esquinas de una caja es tan sencillo como esto:

```
p.caja2{
      padding: 60px;
      border: 1px solid black;
```

```
        background: url(esquina1.png) top left no-repeat,
                    url(esquina2.png) top right no-repeat,
                    url(esquina3.png) bottom right no-repeat,
                    url(esquina4.png) bottom left no-repeat;
        background-color: white;
        }
```

Y si queremos solapar dos imágenes de fondo transparente en la misma caja nos basta con hacer lo siguiente:

```
p.caja3{
        width: 300px;
        height: 400px;
        background: url(imagen1.gif) bottom no-repeat,
                    url(imagen2.gif) bottom no-repeat;
        }
```

Esquinas redondeadas y bordes decoradas

CSS3 incluye, por fin, la posibilidad de redondear bordes sin necesidad de complicadas maniobras como hemos tenido que hacer hasta el momento. La propiedad para hacer esto se llama **border-radius**. Su sintaxis es bien sencilla:

```
border-radius: 10px;
```

El valor indica el radio del arco de circunferencia que delimita el arco. Se puede usar cualquier unidad de medida válida en CSS (pulgadas, em, etc.) y definir de forma independiente cada una de las cuatro esquinas de la caja con la siguiente sintaxis:

```
border-top-right-radius: 5px;
border-bottom-right-radius: 2em;
border-bottom-left-radius: 3em;
border-top-left-radius: 10px;
```

Con el formato abreviado podemos poner dos valores diferentes (el primero para las esquina superior izquierda e inferior derecha y el otro para las otras dos) o cuatro valores (uno para cada esquina empezando por la superior izquierda y continuando en el sentido de las agujas del reloj):

```
border-radius: 10px 5px;
border-radius: 10px 5px 20px 25px;
```

Podemos usar también esquinas irregulares especificando dos radios diferentes: uno para el eje X y otro para el eje Y. El formato cuando todas las esquinas son iguales es este:

```
border-radius: 10px / 50px;
```

Cuando especificamos las esquinas una a una no necesitamos la barra:

```
border-top-right-radius: 5px 10px;
```

Y cuando las esquinas son diferentes pero las especificamos en grupo:

```
border-radius: 10px 10px / 20px 30px;
border-radius: 5px 10px 5px 10px / 10px 5px 10px 5px;
```

El redondeado de esquinas mediante esta propiedad se puede aplicar, además de a una caja, también directamente a una tabla o a una imagen:

```
table {
      width: 200px;
      background-color: rgb(185,205,225);
      text-align: center;
      margin: 30px;
      border: 1px solid gray;
      border-radius: 20px;
      }

#imgredondo {
      width: 600px;
      height: 150px;
      margin: 0 auto;
      border-radius: 1em;
      border: 1px solid gray;
      background-image: url(header.jpg);
      }
```

Sombras

CSS3 incorpora dos propiedades que sirven par dar sombras a textos y cajas que se llaman, respectivamente, **text-shadow** y **box-shadow**. El formato básico de ambas es similar. Veamos un par de ejemplos:

```
text-shadow: 15px 5px 10px orange;
box-shadow: -5px -10px 5em #000;
```

Los dos primeros parámetros son el desplazamiento vertical y horizontal de la sombra. Valores positivos desplazan esta hacía abajo y la derecha. Valores negativos hacia arriba y la izquierda. El tercer parámetros marca la intensidad y/o difuminación de la sombra (valores más pequeños definen una sombra más intensa y marcada, mientras que valores grandes crean una sombra muy difuminada y poco definida). No se puede usar un porcentaje como ninguno de estos tres parámetros. El cuarto parámetro es el color de la sombra y puede expresarse de cualquiera de las formas válida vistas en CSS.

box-sahdow permite, además, dos parámetros opcionales: uno de ellos iría delante del color y define, mediante una medida, la distancia al objeto del foco que provoca la sombra, de forma que valores positivos acercan el foco provocando una sombra mayor y valores negativos alejan el foco provocando una sombra menor. El siguiente parámetro opcional es la palabra clave **inset** que iría al final de la declaración y provocaría una sombra interior a la caja.

```
box-shadow: -5px -10px 5em 15px #000 inset;
```

Opacidad

La propiedad **opacity** permite definir elementos parcial o totalmente transparentes. El valor de esta propiedad es un número decimal entre 0 y 1, de forma que 1 define la total opacidad (cuerpo sólido y no translucido) y el 0 la total transpariencia.

```
opacity: .3;
```

Más control de la tipografía

En CSS3 se normaliza el uso de tipografías externas (como las de Google Fonts que vimos hace ya tiempo) de forma que podamos usar cualquier tipo de fuente disponible en la red en nuestras páginas. La incorporación y descarga de fuentes se hace mediante la instrucción **@font-face** que tiene el siguiente formato:

```
@font-face {
      font-family: "fantasia";
      url("fantasia.ttf");
      }
```

Si quisiéramos importar más de una fuente en nuestro proyecto necesitamos una instrucción @font-face por cada una de ellas:

```
@font-face { font-family: vera; src: url("VeraSeBd.ttf"); }
@font-face { font-family: fantasia; src: url("fantasia.ttf"); }
```

Inexplicablemente, aunque las fuentes ttf se han popularizado gracias a Microsoft, estas no son soportadas por ninguan versión de explorer. Microsoft prefiere usar las tipografías eot. Para que los usuarios con Explorer vean la fuentes necesitamos convertir las ttf a este formato y usar esta sintaxis que contempla ambos tipos:

```
@font-face {
      font-family: fantasia;
      src: url("fantasia.ttf");
      src: url("fantasia.eot");
      }
```

Existen diferentes programas capaces de convertir entre diferentes formatos. También puedes usar este recurso web:

http://www.kirsle.net/wizards/ttf2eot.cgi

La nueva propiedad **font-size-adjust** permite ajustar la altura de fuentes diferentes mediante el uso de un coeficiente de ajuste. Sólo está implementada en Firefox:

```
font-size-adjust: .448;
```

Aunque puede aproximarse "a ojo" por el método de prueba y de error, existe un recurso web que nos lista las fuentes disponibles en nuestro sistema y nos calcula los coeficientes de ajuste:

http://www.brunildo.org/test/xheight.pl

La propiedad **text-overflow** decide lo que se hará con un texto que no cabe completamente dentro de la caja en la que se encuentra. Tiene dos posibles valores: **clip** (lo corta sin más) o (lo corta pero antes coloca tres puntos suspensivos).

La propiedad **font-stretch** permitirá ajustar el ancho de los caracteres, pero por el momento ningún navegador la implementa. Sus valores posibles son, de más ancha a más estrecha, **wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, normal, semi-expanded, expanded, extra-expanded** y **ultra-expanded.**

La propiedad **text-wrap** define como cortar las líneas de un texto y tampoco está implementada por el momento. El valor **normal**, por defecto, indica que sólo se pueden romper en los espacios y puntos. **none** indica que no pueden romperse, **unrestricted** que pueden romperse entre dos caracteres cualesquiera y **suppress** indica que no sólo no pueden romperse sino que se suprimen incluso los saltos de línea manuales que hayamos colocado.

Columnas más fáciles

CSS3 dispone de siete propiedades que nos van a permitir mostrar el texto mediante columnas de forma muy fácil. Son **columns, column-gap, column-fill, column-span** y **column-rule**. Al igual que ocurre con las propiedades relacionadas con los gradientes, esta colección de propiedades precisa llevar el prefijo del navegador (**-moz-, -webkit-**) para funcionar en firefox y chrome pero no así en opera. Explorer no las implementa aún. Los ejemplos a continuación se verán con la sintaxis necesaria para mozilla salvo que se indique lo contrario.
columns define el número de columnas y el ancho mínimo esperado para cada una de ellas. Puede "separarse" en dos propiedades independientes: **column-width** y **column-count.** El funcionamiento de ambos parámetros está íntimamente relacionado pero la interpretación que hacen de ellos los diferentes navegadores no es unánime. Mozilla, por ejemplo, da preferencia al número de columnas mientras que Opera lo hace con el ancho de estas.

```
/* -moz-columns: 200px 3; */
-moz-column-width: 200px;
-moz-column-count: 3;
```

La propiedad **column-rule** define un separador entre las columnas y también

puede dividirse en tres propiedades distintas: **column-rule-width, column-rule-style** y **column-rule-color**. Los estilos permitidos son los mismos que los de un borde.

```
/* -moz-column-rule: 1px solid gray; */
-moz-column-rule-width: 1px;
-moz-column-rule-style: solid;
-moz-column-rule-color: gray;
```

column-gap define el espacio de separación entre columnas:

```
-moz-column-gap: 50px;
```

column-span nos permite marcar algún elemento para que no se propague entre diferentes columnas sino que esté completo en una de ellas. Imaginemos, por ejemplo, un texto de entradilla o un titular. Tiene dos valores válidos: **1** y **auto** según que deba limitarse a la primera columna o expandirse por las que precise. Sólo lo implementan Opera y Chrome:

```
column-span: 1;
-webkit-column-span: 1
```

column-fill define como se "llenaran" las columnas. Por defecto su valor es balance, que hace que el navegador divida el contenido de la caja en tantas partes iguales como columnas para que estas sean lo maś uniformes y parejas posible. Si su valor es auto, el navegador llenará primero la primera columna y no pasará a la segunda hasta que no quepa nada en ella y así sucesivamente. Esta propiedad aún no está implementada en ningún navegador.

Transformaciones y trancisiones

La propiedad **transform** permite realizar diferentes transformaciones en 2D o 3D sobre cualquier elemento (textos, cajas...) de nuestra web. Se trata de una de las propiedades sobre las que aún se está trabajando (sobre todo en el terreno de 3D), implementada en todos los navegadores excepto el explorer, que hay que usar con los prefijos propios de cada navegador (**-moz-, -webkit-** y **-o-**) y para la que sólo veremos algunos ejemplos básicos.

Las transformaciones disponibles son rotación cambio de escala, deformación oblicua y desplazamiento.

La rotación se puede expresar en grados (**deg**), radianes (**rad**), aplicarse a los ejes X e Y por separado (**rotateX** y **rotateY**), a ambos por igual (**rotate**), y especificar un número de grados positivo (en el sentido de las agujas del reloj) o negativo (sentido contrario). Un ejemplo:

```
transform: rotate(-20deg);
-moz-transform: rotate(-20deg);
-webkit-transform: rotate(-20deg);
-o-transform: rotate(-20deg);
```

Los cambios de escala se especifican mediante un número entero o decimal que

puede ser mayor que cero (ampliación). Menor (reducción) o negativo (reflexión). También podemos aplicar un factor de escala diferente al eje X y al Y (**scaleX** y **scaleY**). Otro ejemplo:

```
-moz-transform: scale(2.5);
-webkit-transform: scale(2.5);
-o-transform: scale(2.5);
transform: scale(2.5);
```

La deformación oblicua usa el parámetro shew y nos permite especificar un ángulo en grados, radianes o gradientes (**grad**), positivos, negativos (para el sentido, igual que en la rotación) y especificar una deformación común o separada para cada eje:

```
-moz-transform: skew(-30deg);
-webkit-transform: skew(-30deg);
-o-transform: skew(-30deg);
transform: skew(-30deg);
```

Por último, translate nos permite definir un desplazamiento indicando este en una medida concreta o en tanto por ciento, con valores positivos (abajo y hacia la derecha) o negativos (arriba y hacia la izquierda). En este caso el valor de la propiedad puede llevar uno o dos valores aplicando en el primer caso a ambos ejes y en el segundo distinguiendo entre ambos:

```
-moz-transform: translate(100px, -50px);
-webkit-transform: translate(100px, -50px);
-o-transform: translate(100px, -50px);
transform: translate(100px, -50px);
```

Otras transformaciones en diferentes estados de normalización son: **matrix, matrix3d, translate3d, translateZ, scale3d, scaleZ, rotate3d, rotateZ** y **perspective.**

La propiedad **transition**, por último, nos va a permitir realizar pequeños efectos de animación sin necesidad de recurrir a javascript. Usamos para ello cuatro propiedades: **transition-property, transition-duration, transition-timing-function** y **transition-delay** (o **transition**, sin más). Tampoco está disponible para Explorer. Veamos primero un ejemplo y luego explicaremos los diferentes parámetros:

```
div.trancision {
      margin: 10px auto;
      padding: 10px;
      text-align: center;
      width: 200px;
      height: 100px;
      background-color: pink;
      border: 5px solid blue;

      -webkit-transition: border 5s ease-out ;
      -moz-transition: border 5s ease-out;
      -o-transition: all 5s ease-out;
```

```
        transition: border 5s ease-out;
        }

div.trancision:hover{
        border-right: 200px solid #9cf;
        }
```

En la primera regla especificamos que la trancisión afectará al borde (**trancision-property**), que durará 5 segundos (**trancision-timing-function**) y que será rápida al principio y más lenta al final (**ease-out**). En la segunda regla, que se activará al colocar el ratón encima, hacemos un cambio en el borde del elemento (tanto de tamaño como de color) y la trancisión realizará la transformación entre el estado original y el final con los datos especificados. Al retirar el ratón la animación se invertirá.

La propiedad puede ser el ancho (**width**), el alto (**height**), el **color**, una lista de propiedades diferentes separadas por coma o **all** (todas las propiedades que cambien entre uno y otro estado). Los tiempos y retrasos se especifican en segundos (**s**) o milisengudos (**ms**). Por último, el timing-function puede ser constante (linear), de lento a rápido (**ease-in**), de rápido a lento (**ease** o **ease-out**) o lento al principio y al final y más rápido en el centro (**ease-in-out**).

Se pueden especificar también diferentes parámetros para diferentes propiedades:

```
transition: width 2s ease, height 3s linear;
```

11.ENLACES, BIBLIOGRAFÍA Y REFERENCIAS

Es absolutamente imposible reseñar todas las ayudas y referencias que a lo largo del año en que he redactado estos apuntes y durante mucho tiempo antes he tomado de la web, libros de consulta, ejemplos de páginas encontradas en Internet, etc. Pero si tuviera que quedarme con algunas como referencias "de cabecera", sería justo mencionar al menos a las siguientes.

- http://www.librosweb.es/ es una colección de libros web escritos en castellano que abarcan todas las disciplinas que hemos tocado aquí.
- http://www.w3schools.com/ es, sin duda, la mejor referencia que existe en la web cuando quieres consultar los argumentos de una etiqueta, los valores que puede tomar o, simplemente, ver algunos ejemplos simples de cualquier elemento HTML y/o CSS
- http://meyerweb.com/ es la página web de Eric A. Meyer, un verdadero monstruo de esta disciplina y artífice de la gran mayoría de trucos y técnicas que hemos visto en el capítulo 7

Y, para los fanáticos del papel, si tengo que mencionar dos libros de mi biblioteca personal sobre este tema me quedo con estos:

- HTML5 y CSS3. Van Lancker, Luc. Ediciones ENI. 2011
- CSS, Técnicas profesionales para el diseño moderno. Meyer, Eric A. Editorial Anaya Multimedia. 2011