

Arrancar con HTML5

Curso de programación

Apoyo en la



Arrancar con HTML5.

Curso de programación

Emmanuel Herrera Ríos



Buenos Aires • Bogotá • México, D.F. • Santiago de Chile

Edición:

Alejandro Herrera

Corrección:

Roberto Alfaro M.

Revisión:

Dr. David Moisés Terán Pérez

Gerente editorial:

Marcelo Grillo

Datos catalográficos

Herrera Ríos Emmanuel

Arrancar con HTML5. Curso de programación

Primera Edición

Alfaomega Grupo Editor, S.A. de C.V. México.

ISBN: 978-607-707-331-4

Formato: 17 x 23 cm

Páginas: 264

Arrancar con HTML5. Curso de programación

Emmanuel Herrera Ríos

Derechos reservados © 2012, por Alfaomega Grupo Editor, S.A. de C.V. México

Primera Edición: *Alfaomega Grupo Editor, S.A. de C.V. México*, octubre de 2011© 2012, por **Alfaomega Grupo Editor, S.A. de C.V.**

Pitágoras 1139, Col. Del Valle, C.P. 03100, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro No. 2317

Internet: <http://www.alfaomega.com.mx>E-mail: atencionalcliente@alfaomega.com.mx**ISBN: 978-607-707-331-4****Derechos reservados**

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Nota importante:

La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. Alfaomega Grupo Editor, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele. Los nombres comerciales que aparecen en este libro son marcas registradas de sus propietarios y se mencionan únicamente con fines didácticos, por lo que Alfaomega Grupo Editor, S.A. de C.V. México no asume ninguna responsabilidad por el uso que se dé a esta información, ya que no infringe ningún derecho de registro de marca. Los datos de los ejemplos y pantallas son ficticios, a no ser que se especifique lo contrario.

Impreso en México. Printed in Mexico.**Empresas del grupo:****México:** Alfaomega Grupo Editor, S.A. de C.V. – Pitágoras 1139, Col. Del Valle, México, D.F. – C.P. 03100.

Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396

E-mail: atencionalcliente@alfaomega.com.mx**Colombia:** Alfaomega Colombiana S.A. – Carrera 15 No. 64 A 29, Bogotá, Colombia,Tel.: (57-1) 2100122 – Fax: (57-1) 6068648 – E-mail: cliente@alfaomega.com.mx**Chile:** Alfaomega Grupo Editor, S.A. – Dr. La Sierra 1437, Providencia, Santiago, ChileTel.: (56-2) 235-4248 – Fax: (56-2) 235-5786 – E-mail: agechile@alfaomega.cl**Argentina:** Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires,Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

Dedicatoria

A mis padres, José Luis y Elba, por su cariño, por apoyarme en todo momento y por darme la oportunidad de aprender. También dedico este proyecto a mi hermanita Yazmín.

Emmanuel Herrera Ríos

Agradecimientos

Deseo agradecer a mi familia por todo su apoyo, especialmente a mis tíos Mónica y Gabriel, por toda su ayuda y afecto a lo largo de los años.

A mis profesores, por todo el conocimiento que me han brindado.

A Luz Romero y Juan Gabriel Cruz, por compartir sus conocimientos, por su ayuda y sobre todo por su amistad.

A todo el personal de Alfaomega Grupo Editor, que con su respaldo y esfuerzo hicieron posible la realización de este libro.

Gracias.

Emmanuel Herrera Ríos

Mensaje del editor

Los conocimientos son esenciales en el desempeño profesional, sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad; el avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionistas conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (IT) para facilitar el aprendizaje. Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento.

Cada capítulo se desarrolla con argumentos presentados en forma sencilla y claramente estructurada hacia los objetivos y metas propuestas.

Cada capítulo concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos para diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y al desarrollo de profesionistas exitosos para beneficio de la sociedad.

Emmanuel Herrera Ríos

Es Ingeniero en Sistemas Computacionales egresado de la Universidad de Colima (UCOL) (www.ucol.mx), universidad pionera en el uso de las Tecnologías de Información como método de apoyo a la enseñanza en América Latina.

Colaborador por parte de la Universidad de Colima en el desarrollo de varios de los primeros contenidos multimedia destinados a la educación, elaborados en México y en América Latina.

Ha trabajado como desarrollador de diversos sistemas de información y control para diferentes consultorías en proyectos para empresas e instituciones como Bancomext, Televisa, Chrysler y Waldo's, entre otras. También se ha desempeñado como diseñador, desarrollador y administrador de sistemas de información financiera para el Grupo Salinas (Banco Azteca y Elektra).

Ha desarrollado y coordinado diversos proyectos multimedia y de capacitación para instituciones públicas ubicadas en México y Latinoamérica a través del Instituto Latinoamericano de la Comunicación Educativa y como consultor independiente.

Es consultor externo y desarrollador independiente. En www.internet80.com encontrará su blog, en el que trata temas de tecnología y desarrollo para internet, multimedia y programación.

Contenido

Arrancar con HTML5. Curso de programación	XI
Acceso al material complementario	XIII

Capítulo 1

¿De dónde viene y qué es HTML?	1
Expectativa.....	2
Introducción	3
Un poco de historia.....	4
El éxito puede ser el mayor problema.....	4
Round 2 (nuevos competidores)	5
¿HTML5 o una versión anterior?	5
HTML4	5
XHTML	6
HTML5	6
Comprendiendo las etiquetas (tags)	6
Comprendiendo las hojas de estilo en cascada (CSS)	7
JavaScript	8
Tecnologías de páginas activas de servidor	8
Resumen	9
Autoevaluación.....	9
Evidencia	10
Referencias.....	11
Bibliografía	11
Páginas Web recomendadas	11
Respuestas sugeridas	12

Capítulo 2

Preparándonos para HTML5.....	13
Expectativa.....	14
Introducción	15
Eligiendo un navegador	15
Tipo de documento.....	17
Etiqueta <!DOCTYPE html>	18

Estructura básica de un documento HTML	18
Etiquetas <head> y <body>	18
Título y Metadatos.....	19
Ejemplo	19
Ejercicio	21
Párrafos y saltos de línea.....	21
Detección de las características de HTML5.....	23
Resumen.....	26
Autoevaluación	26
Evidencia.....	27
Referencias	27
Bibliografía	27
Páginas Web recomendadas	27
Respuestas sugeridas	28

Capítulo 3

Más bases HTML	29
Expectativa.....	30
Introducción	31
¿HTML o XHTML?.....	31
Formato de texto con etiquetas.....	32
Cabeceras	33
Ejercicio	34
Negritas, cursiva, subrayado y otros	34
Ejercicio	36
Subíndices y superíndices	36
Ejercicio	37
Bloque de citas.....	38
Ejercicio	38
Línea Horizontal	39
Listas	40
Listas numeradas y listas no numeradas	41
Listas de definición	42
Ejemplo.....	43

Imágenes	44
Tablas	46
Ejemplo	47
Hipervínculos (enlaces).....	48
Ejemplo	49
Formularios (controles clásicos).....	51
Estructura del formulario	56
Campo de texto	56
Área de texto	57
Campo de contraseña (password)	57
Lista de selección simple (combobox).....	58
Lista de selección múltiple (listbox)...	59
Lista de selección múltiple (listbox multiple selection).....	59
Lista de botones radio (radio buttons)	59
Lista de cajas (checkboxes).....	61
Botón para enviar información	62
Botón para borrar información del formulario	62
Botón simple	62
Imagen (botón)	63
Notas adicionales sobre botones.....	63
Campo oculto	63
Validar páginas HTML5.....	64
Resumen	64
Autoevaluación.....	65
Evidencia	65
Referencias.....	66
Bibliografía	66
Páginas Web recomendadas	66
Respuestas sugeridas	66

Capítulo 4

Los nuevos elementos HTML5.....

Expectativa.....	68
Nueva semántica	69
Elementos estructurales.....	69
article.....	69
aside	70
footer	71
header	71
nav	71
section.....	72
Elementos estructurales complementarios.....	72
address.....	72
hgroup	73

menu	73
Elementos semánticos en línea.....	74
command	74
details y summary	75
dfn	75
figure y figcaption	76
wbr.....	76
Media.....	76
audio	76
canvas.....	77
embed.....	79
svg	80
video.....	81
Resumen.....	83
Autoevaluación	83
Evidencia.....	84
Referencias	84
Bibliografía.....	84
Páginas Web recomendadas	84
Respuestas sugeridas	84

Capítulo 5

Controles nuevos en formularios HTML5

Expectativa	86
Introducción	87
Un vistazo a los controles nuevos	87
Elementos independientes	90
fieldset.....	90
keygen	90
label.....	91
meter	91
progress.....	92
Elementos tipo input.....	92
email	92
number	93
range.....	93
search	93
tel.....	94
url.....	94
Nuevos atributos.....	94
autofocus.....	94
pattern	95
placeholder	95
Resumen.....	96
Autoevaluación	96
Evidencia.....	97
Referencias	97
Bibliografía.....	97
Páginas Web recomendadas	97

Respuestas sugeridas	98
----------------------------	----

Capítulo 6

Primeros pasos con CSS	99
-------------------------------------	-----------

Expectativa.....	100
------------------	-----

Introducción.....	101
-------------------	-----

Un vistazo a hojas de estilo o CSS

(Cascading Style Sheets).....	101
-------------------------------	-----

Utilizar estilos a nivel local	102
--------------------------------------	-----

Utilizar estilos a nivel documento	103
--	-----

Utilizar estilos a nivel de sitio (link).....	105
---	-----

Utilizar clases y el identificador

(class e id)	107
--------------------	-----

Manipular la apariencia de una página	110
--	-----

Manipular texto.....	110
----------------------	-----

Manipular colores.....	114
------------------------	-----

Manipular Bordes	115
------------------------	-----

Agrupación de elementos (span y div)	116
--	-----

span	116
------------	-----

Ejemplo	117
---------------	-----

div.....	117
----------	-----

Ejemplo	118
---------------	-----

Posicionamiento	119
-----------------------	-----

Elementos flotantes (propiedades

float y clear)	119
----------------------	-----

Posicionamiento absoluto	123
--------------------------------	-----

posicionamiento relativo.....	124
-------------------------------	-----

Resumen	126
---------------	-----

Autoevaluación.....	126
---------------------	-----

Evidencia	127
-----------------	-----

Referencias.....	127
------------------	-----

Bibliografía	127
--------------------	-----

Páginas Web recomendadas	127
--------------------------------	-----

Respuestas sugeridas	128
----------------------------	-----

Capítulo 7

Nuevo y mejorado CSS3.....	129
-----------------------------------	------------

Expectativa.....	130
------------------	-----

Introducción.....	131
-------------------	-----

Motores de renderizado y CSS3	131
-------------------------------------	-----

Nuevas notaciones para color	132
------------------------------------	-----

Notación RGBA	132
---------------------	-----

Notación HSL Y HSLA.....	132
--------------------------	-----

Bordes.....	133
-------------	-----

Bordes redondeados.....	135
-------------------------	-----

Bordes sombreados	135
-------------------------	-----

Bordes con imagen	136
-------------------------	-----

Gradientes.....	138
-----------------	-----

Transformaciones.....	144
-----------------------	-----

Transiciones	146
--------------------	-----

Transparencia	150
---------------------	-----

Texto y fuentes descargables	151
------------------------------------	-----

text-shadow	152
-------------------	-----

text-stroke	152
-------------------	-----

@font-face	153
------------------	-----

Herramientas de selección	154
---------------------------------	-----

Selección por atributo	154
------------------------------	-----

Selección por relación padre-hijo	155
---	-----

Selección inversa (not).....	156
------------------------------	-----

Resumen.....	158
--------------	-----

Autoevaluación	158
----------------------	-----

Evidencia.....	159
----------------	-----

Referencias	159
-------------------	-----

Bibliografía	159
--------------------	-----

Páginas Web recomendadas	159
--------------------------------	-----

Respuestas sugeridas	160
----------------------------	-----

Capítulo 8

Novedades en JavaScript	161
--------------------------------------	------------

Expectativa	162
-------------------	-----

Introducción	163
--------------------	-----

Nuevas herramientas de selección	163
--	-----

getElementsByClassName().....	163
-------------------------------	-----

getElementsByTagName	164
----------------------------	-----

querySelector()	164
-----------------------	-----

querySelectorAll().....	165
-------------------------	-----

Almacenamiento de sesión y local.....	165
---------------------------------------	-----

Almacenamiento por sesión	166
---------------------------------	-----

Almacenamiento local	166
----------------------------	-----

Asignando y recuperando datos

de sessionStorage y localStorage ...	166
--------------------------------------	-----

Métodos y propiedades de session

Storage y localStorage	169
------------------------------	-----

Web offline (caché)	169
---------------------------	-----

Geolocalización.....	172
----------------------	-----

WebSocket	176
-----------------	-----

Creación interfaz	183
-------------------------	-----

Conexión de sockets.....	183
--------------------------	-----

Envío de mensaje.....	184
-----------------------	-----

Algunas puntualizaciones.....	185
-------------------------------	-----

Web workers	185
-------------------	-----

Creación interfaz	190
-------------------------	-----

Llamar al web worker.....	191
---------------------------	-----

Construir el código del web worker....	192
--	-----

Resumen: pasos para crear un

Web worker	193
------------------	-----

Algunas puntualizaciones entre

Web workers y navegadores	194
---------------------------------	-----

Resumen.....	195
--------------	-----

Autoevaluación.....	195
Evidencia	196
Referencias.....	197
Bibliografía	197
Paginas Web recomendadas	197
Respuestas sugeridas	198

Capítulo 9

Utilizar Canvas.....	199
Expectativa.....	200
Introducción	201
Primeros pasos con Canvas	201
Preparación del elemento Canvas.....	201
La cuadrícula y su eje de coordenadas	203
Dibujar textos	204
Dibujar figuras rectangulares	206
Dibujar caminos (paths)	207
moveTo ()	209
lines ().....	210
arc ().....	212
Curvas cuadráticas	216
Curvas bezier.....	218

Utilizar imágenes.....	221
DrawImage()	221
Escalar imagen con drawImage ().....	223
Recortar y dibujar parte de una imagen con drawImage ().....	225
Transformaciones.....	227
Guardar y restaurar el estado	228
translate ().....	231
rotate ()	233
scale ()	235
Animación.....	237
Control de una animación	238
Ejemplo 1	239
Ejemplo 2	241
Ejemplo 3	243
Resumen.....	247
Autoevaluación	247
Evidencia.....	248
Referencias	249
Bibliografía	249
Paginas Web recomendadas	249
Respuestas sugeridas	250

Arrancar con HTML5. Curso de programación

Este libro se pensó con la idea de apoyar a todos los lectores de habla hispana interesados en conocer las posibilidades que ofrece la nueva especificación de HTML, popularmente conocida como HTML5. Los capítulos de este libro abordan los conceptos más básicos del lenguaje, acompañado con ejemplos a cada concepto explicado a lo largo del mismo.

Éste no es un manual o guía para migrar aplicaciones de las versiones anteriores de HTML a HTML5. Éste es un libro que introduce poco a poco al lector, en forma amable pero consistente con las nuevas alternativas que ofrece actualmente HTML5.

Dado que la especificación de HTML5 estará aún en movimiento por algunos años antes de estar totalmente terminada, este libro, acorde con esta situación, ofrece recursos en línea que serán actualizados y ocasionalmente se presentará contenido adicional para respaldar mejor el aprendizaje del lenguaje obtenido con el texto.

Con la experiencia, una persona se puede dar cuenta de que a veces encontrar y comprender información acerca de algún tema de tecnología puede ser caótico y disperso, incluso la barrera del idioma representa un problema que se suma a las dificultades que enfrentan algunas personas que desean aprender los temas más nuevos y apasionantes en el mundo del desarrollo de tecnologías. Es por estas razones que un libro que ordene ideas, conceptos e información en nuestro idioma es una forma muy bella y eficaz de aprender.

Sin más, ¡bienvenid@ al mundo de HTML5!

La estructura de cada capítulo incluye:

- **Listado de capacidades y competencias.** Se describen brevemente los aprendizajes esperados del participante al término del estudio del tema y que integra conocimientos, habilidades, valores y actitudes como un conjunto de comportamientos sociales, afectivos y habilidades cognitivas, psicológicas sensoriales y motoras, que permiten llevar a cabo adecuadamente un papel, un desempeño, una actividad o una tarea.
- **Evidencias de las capacidades desarrolladas.** Describen brevemente lo que se espera que el lector demuestre al término del estudio y que consiste en demostrar que puede llevar a cabo adecuadamente un papel, un desempeño, una actividad o una tarea.

- **Mapa conceptual del capítulo.** Le dará al lector, mediante un esquema gráfico, una visión de conjunto de los conceptos, su jerarquización y la relación entre ellos
- **Exposición del tema.** Corresponde al desarrollo del capítulo. Cuando se ha considerado pertinente, se han incluido recuadros con ejemplos, casos breves y figuras con esquemas explicativos. También es importante señalar que los conceptos clave están destacados en el texto y sus definiciones están en el glosario al final del libro.
- **Resumen.** Estas secciones buscan entregar una visión general del capítulo y facilitar la retención de conceptos nuevos.
- **Actividades.** Para asegurar un aprendizaje efectivo, resulta imprescindible buscar la interacción del lector con los temas expuestos en cada sección o capítulo.
- **Referencias.** Al final de cada capítulo se ha colocado bibliografía recomendada y páginas Web recomendadas para estimular y facilitar la profundización en el tema.

Nota:

En la mayor parte de los capítulos se indican **Páginas Web recomendadas:** Sitios Web donde el lector podrá ampliar, profundizar y complementar información, localizar casos de éxito y otros recursos para investigar; también se puede bajar textos y encontrar información actualizada en universidades, institutos de investigación y centros de documentación.

Acceso al material complementario

Para tener acceso al material complementario del libro: **Arrancar con HTML5. Curso de programación**, es necesario:

1. Ir a la página: <http://virtual.alfaomega.com.mx/>
2. Regístrese como usuario de sitio llene completamente el formulario. Guarde en un lugar seguro su nombre de Usuario y Contraseña, para emplearlos para futuros ingresos.
3. Ingrese con su Usuario a la sección de libros.
4. Busque y seleccione la imagen correspondiente a este libro para descargar su material complementario.

NOTAS:

1. En adelante se referirá al material complementario como el material que se descargó del sitio Web.
2. Se recomienda respaldar los archivos descargados de las páginas Web en un soporte físico (CD, USB, disco duro o cualquier otro medio de almacenamiento).

¿De dónde viene y qué es HTML?

1

Reflexione y responda las preguntas siguientes:

¿Qué es HTML y cuál es su historia?

¿Por qué usar HTML5?

¿Qué son las etiquetas HTML?

¿Cuáles son las tecnologías relacionadas comúnmente con HTML5?

Contenido

¿De dónde viene y qué es HTML?

Introducción

1.1 Un poco de historia

1.2 El éxito puede ser el mayor problema

1.3 Round 2 (nuevos competidores)

1.4 ¿HTML5 o una versión anterior?

1.5 Comprendiendo las etiquetas (tags)

1.6 Comprendiendo las hojas de estilo en cascada (CSS)

1.7 JavaScript

Expectativa

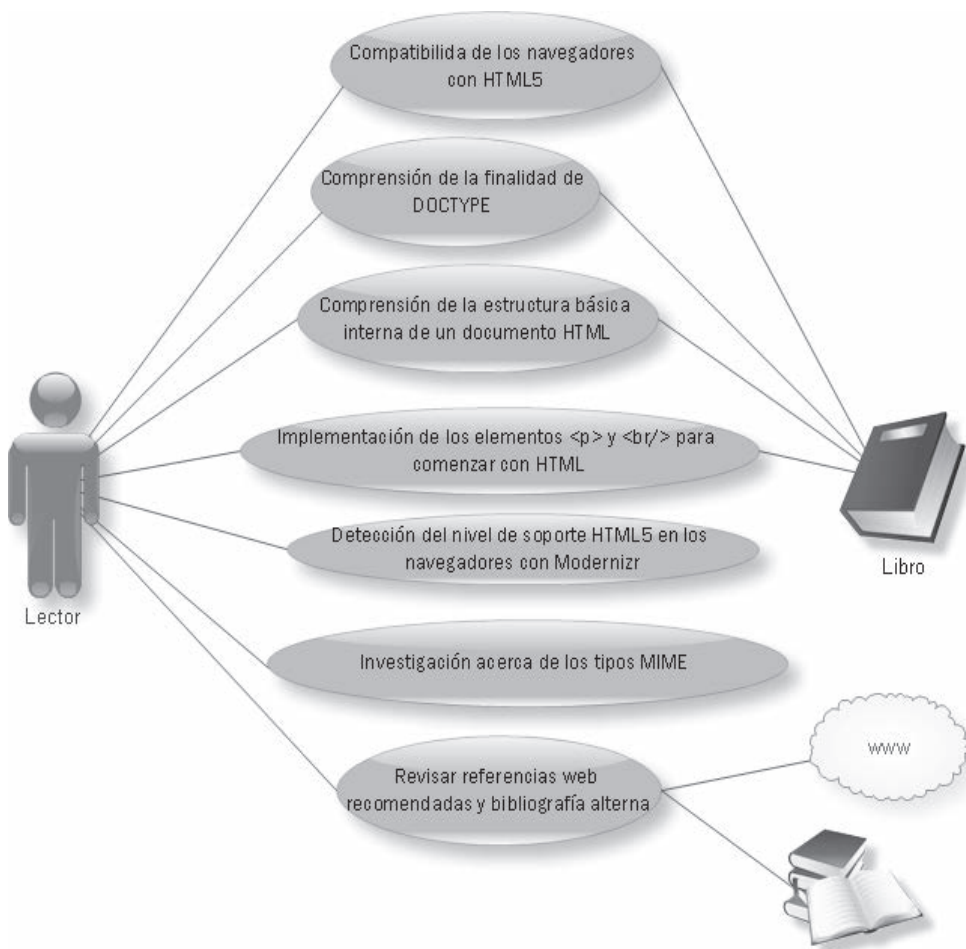
Conocer la evolución y tendencia de HTML.

Conocer las características de las versiones HTML.

Comprender las razones para aprender HTML5

Después de estudiar este capítulo, el lector será capaz de:

- Saber porque usar HTML5.
- Conocer las versiones más relevantes de HTML previas a HTML5.
- Conocer algunas ventajas de las diferentes versiones de HTML.



INTRODUCCIÓN

Antes de entrar de lleno en lo que es propiamente crear una página y sitios utilizando HTML5, considero apropiado dedicar este primer capítulo a dar un breve vistazo a sus orígenes y razones de ser, para dar una mejor perspectiva, sobre todo a aquellos que recién comienzan a adentrarse en el desarrollo Web o en la programación. Si usted es un desarrollador experimentado, quizá sepa las generalidades de la historia de HTML y considere adecuado no leer esta parte, sin embargo, lo invito a leer de cualquier manera este capítulo, ya que el HTML5 nos está poniendo en una situación que no es del todo nueva, pero sí es quizá la de mayor influencia en nuestro futuro inmediato como desarrolladores: la implementación del lenguaje y la especificación del mismo, no están bailando al mismo ritmo.

Imagine, por ejemplo, que está diseñando un automóvil, y pasado un tiempo logra tener un prototipo que no sale aún al mercado, pero que resulta ser tan bueno y necesario, que todos comienzan a hacer cosas basadas en ese prototipo antes de ponerlo a la venta, eso es lo que está sucediendo con HTML5, ha resultado muy interesante y prometedor, y aunque tiene una base sólida y mucho apoyo por parte de la industria, habrá que estar siempre pendientes de cambios y mejoras, mantenga esto siempre en mente al leer éste u otros libros de HTML5.

Con una breve pero dinámica historia, el HTML es parte esencial de la Web y ha logrado cambiar y madurar al ritmo de Internet en general, en las primeras concepciones del HTML fue simplemente una manera útil de usar etiquetas (tags) para determinar cómo una página debería ser desplegada, pero se han ido añadiendo varias características. Hoy Internet sigue siendo acerca de documentos, pero más que nunca, también de aplicaciones, sobre todo si se toma en cuenta que antes sólo las computadoras de escritorio eran las únicas que desplegaban contenido de la Web, pero ahora también los dispositivos móviles lo pueden hacer, por lo que todo parece indicar que es el momento de un nuevo estándar, y HTML5 es ese estándar.



Fig. 1.1 La implementación del lenguaje HTML.

1.1 Un poco de historia

En 1989 Tim Berners-Lee creó un sistema que permitía conectar documentos electrónicos, utilizando un lenguaje que enlazaba un documento con otro, dando algunas características de formato a estos textos. Este lenguaje llamado HTML o Hypertext Markup Language, traducido al español como Lenguaje de Marcado de Hipertexto.

Este lenguaje fue creado deliberadamente para que fuera muy simple, ya que en aquel entonces la Internet se utilizaba con base en comandos de texto, es decir, que aunque Internet existía, no había un navegador que presentara información en ventanas haciendo “clics” como sucede hoy en día, por lo que había que facilitarles esto en la medida de lo posible a las personas que creaban documentos con este formato en el lenguaje.

Por supuesto, esta idea tuvo mucho éxito, y con el tiempo había documentos HTML por todas partes.

1.2 El éxito puede ser el mayor problema

La frase, lejos de desanimar, tiene el objetivo de ilustrar algunos casos como el de HTML. Dado el éxito del formato, los creadores de contenido pronto se percataron de que las propiedades del HTML al momento no eran suficientes para satisfacer sus intereses y necesidades, fue entonces cuando las cosas comenzaron a ponerse interesantes. Muchas personas comenzaron a construir programas o software que funcionaban como intérpretes para el HTML, y se comenzó a crear una especie de competencia para ver quién implementaba más rápido y mejor las características que se iban añadiendo al HTML. En 1993 surgió “Mosaic”, el primer navegador capaz de desplegar imágenes.

Las cosas continuaron así durante algunos meses, y en 1994 un gran jugador surgió, Netscape Navigator, el cual fue el favorito de muchos un buen tiempo, sin embargo, su dominio pronto se vio amenazado por otro grande, en 1995 surge el Microsoft Internet Explorer, que es hasta hoy el navegador más utilizado. A todos estos eventos se les popularizó con el nombre de “la primera guerra de navegadores”.

Mientras todo esto sucedía, había varios grupos tratando de estandarizar el HTML para los navegadores, el más importante de estos grupos fue la W3C o World Wide Web Consortium, liderada por la persona que comenzó con todo este desastre (una vez más, sólo ilustrando), Tim Berners-Lee.

En realidad es que ni Netscape ni Microsoft hicieron gran caso de los esfuerzos de estos grupos, y añadieron características propias tratando siempre de competir entre sí, y aunque surgió el HTML2, ninguno de los grandes competidores adoptó por completo este estándar, de manera similar sucedió con el HTML3.2.

El HTML4 surgió en 1998 y dado que el Internet Explorer fue claramente el ganador de la primera guerra de navegadores, con un uso del 95% por parte de los usuarios, el HTML quedó prácticamente a disposición de Microsoft sin importar demasiado las organizaciones que intentaban estandarizarlo, aunque ciertamente en el año 2002 Internet Explorer 6 utilizó el estándar.

1.3 Round 2 (nuevos competidores)

Durante dos años y ante el dominio de Microsoft no hubo ninguna innovación relevante, y no fue sino hasta el año 2004 cuando la aún muy activa W3C cobró fuerza, principalmente debido a que tuvieron un miembro importante, la Mozilla Foundation y a otro que ha peleado como pocos en el mercado, que es la Opera Software. La carta fuerte fue el navegador Mozilla Firefox lanzado ese mismo año, que implementaba varias innovaciones que seguían casi todos los estándares de la W3C, eventualmente otros navegadores menos difundidos pero sólidos siguieron este ejemplo (Opera, Apple Safari y el reciente Google Chrome).

Ante la presión de la competencia, Microsoft ha anunciado y cumplido en su versión Internet Explorer 9 un mucho mayor compromiso con los estándares de la W3C. Otro competidor que hay que seguir de cerca es Google Chrome, ya que éste ha demostrado tener gran visión del futuro hasta ahora y parece que ha entendido antes que sus competidores hacia dónde se mueven las cosas, además de por supuesto, tener gran influencia para poner pautas a competidores y desarrolladores.

La manera de percibir la Web es hoy diferente; para los desarrolladores la Web ya no es una novedad, ya no se piensa en colocar sólo páginas informativas, ahora las páginas más simples dan lugar a verdaderas aplicaciones que interactúan con el usuario y la Web al mismo tiempo.

1.4 ¿HTML5 o una versión anterior?

Aunque similares, cada versión de HTML utiliza etiquetas diferentes, y ya que en sí mismo el HTML5 tiene objetivos más ambiciosos de los que pretendían cubrir versiones anteriores, éste tiene su propio y más extenso juego de etiquetas, pero analice un poco.

HTML4

Ventajas: Es ampliamente aceptado hoy en día, su implementación está prácticamente en todos los navegadores, es permisivo con pequeños errores de código (esto puede verse también como desventaja), si la compatibilidad entre navegadores es un requerimiento primordial puede ser la mejor opción.

Desventajas: Aunque las funcionalidades propietarias añadidas a lo largo de los años han creado cierta flexibilidad, no son comparables con las de HTML5. Las capacidades para formularios son pocas, limitadas y esencialmente feas. La mayoría de los navegadores soportan alguna forma de JavaScript, lo que permite

mayor dinamismo, pero las implementaciones siguen teniendo amplias diferencias en algunos casos.

XHTML

Ventajas: Un estándar estricto basado en el HTML4 y XML (por lo que su implementación es también casi universal), no es permisivo con los errores, como sí lo es HTML4, lo que obliga a formar correctamente un documento HTML como sucede con un documento XML.

Desventajas: Básicamente las mismas de HTML4.

Nota

W3C trabajaba en XHTML2, esto no continuará más. Se ha trabajado en la serialización de HTML5 conocida como XHTML5 por ahora las únicas diferencias son el tipo MIME que utiliza y que la etiqueta DOCTYPE es opcional.

HTML5

Ventajas: Posee una gran cantidad de funcionalidades que los desarrolladores sólo lograban con el uso de algún plug-in de terceros como Applets de Java o Flash embebidos en el código.

Las mejoras en el manejo de multimedia son ampliamente superiores; imágenes, video, audio y fuentes de texto son mucho más manipulables. En muchos casos, estas mejoras se pueden tomar como una respuesta directa a Flash.

Desventajas: HTML5 es tan nuevo que algunas personas que todavía usan computadoras viejas podrían usar navegadores que no pueden visualizarlo aún.

En resumen, HTML5 nos proporcionará una manera de hacer un código más limpio, más fácil de leer y escribir, cubriendo al mismo tiempo y de mejor manera la cada vez mayor demanda de funcionalidades por parte de programadores, diseñadores y usuarios.

1.5 Comprendiendo las etiquetas (tags)

El código HTML consiste de un archivo de texto delimitado por etiquetas o “tags” (palabra usada en inglés), dentro de estas etiquetas se coloca diversa información, como por ejemplo, qué texto se debería visualizar, en qué lugar deben aparecer los elementos, qué imágenes se van a utilizar, entre muchas otras cosas.

Para entender el porqué del uso de este sistema de etiquetado se tendría que remontar a los días donde no existían mensajeros instantáneos y todos utilizaban listas de correo con conexiones por módem dial-up, y donde gente bastante inteligente, utilizando estos recursos, se comunicaba para determinar qué etiqueta

serviría para qué cosa y de qué manera debería escribirse. En este libro no se hará “arqueología tecnológica” para contar estas historias, pero si usted está interesado hay bibliografía disponible para consultar esos detalles.

Un ejemplo de etiqueta es, por ejemplo, la etiqueta `<h2>`, la cual sirve para darle a un texto características de título o cabecera. Utilizando esta etiqueta, el código HTML sería:

```
<h2>Hola, Bienvenido</h2>
```

Si se observa, el código se encontrará con los paréntesis angulares `<` `>` (los símbolos “menor que” y “mayor que”, que enmarcan la etiqueta de apertura o inicio, y `</>`, que acotan la etiqueta de cierre o final), éste es el caso para etiquetas que requieren apertura y cierre.

La etiqueta `` permite colocar imágenes en sus páginas, el uso de esta etiqueta se vería así:

```

```

Esta etiqueta no utiliza cierre, es por eso que se le llama simple o vacía, e introduce una nueva característica en el uso de etiquetas, llamado *atributo*. Los atributos en una etiqueta indican propiedades o comportamientos que la etiqueta en cuestión debe tener, en este ejemplo se utilizó la propiedad `src`, la cual indica que la imagen que debe desplegarse corresponde al archivo `pelota.jpg` y no a cualquier otro. Los atributos pueden ser opcionales u obligatorios, dependiendo de las especificaciones dadas a cada etiqueta.

Desde luego, existen muchas más etiquetas, y la destreza del programador o diseñador radica en saber cuáles usar en el momento adecuado. Se utilizan etiquetas para dar formato a un texto, colocar un enlace, una imagen o algún otro tipo de elemento, el navegador lee e interpreta estas etiquetas y las coloca en una forma “traducida” y visualmente amable para el usuario. Esto es básicamente el funcionamiento de las cosas.

1.6 Comprendiendo las hojas de estilo en cascada (CSS)

CSS (Cascading Style Sheets: Hojas de Estilo en Cascada), es un lenguaje que no es propiamente parte del HTML en ninguna versión definida por la W3C, su objetivo es definir la presentación de un documento escrito en HTML o XML (y claro en XHTML) pero no es parte del código HTML del documento en sí.

Como recordarán los desarrolladores experimentados, era una hazaña dar formato a 100 páginas, donde el frustrante procedimiento era copiar y pegar páginas con la estructura y aspecto básicos una y otra vez y editar manualmente aquella pequeña diferencia o variación en cada página.

Fue entonces cuando las hojas de estilo llegaron al rescate. Usualmente en un archivo separado, el programador usa CSS para especificar el formato para un tipo de etiqueta o para una etiqueta en particular. La idea principal detrás de todo esto es separar el código de estructura del código de presentación, de manera que cuando se modifica el archivo con código CSS, éste inmediatamente afecta a todas las páginas HTML que hacen referencia a él, evitando así editar cada página HTML de forma individual.

Si usted ha utilizado versiones anteriores de HTML pero no ha aprendido CSS y realmente quiere dar el salto a HTML5 es el momento de aprender, ya que mientras que CSS era opcional en HTML4, en HTML5 es indispensable.

1.7 JavaScript

JavaScript es otro lenguaje de programación que tiene una estrecha relación con HTML, y es demasiado extenso, como para dedicarle un libro completo (ya existen varios en el mercado), por esta misma razón, adentrarse en JavaScript escapa del objetivo de este libro, sin embargo, debe quedar claro que JavaScript es esencial para HTML5, varias de las características más importantes de HTML5 requieren de JavaScript, como son la nueva etiqueta `<canvas>`, la geolocalización, el almacenamiento local de datos y otras.

En el Cap. 8 (Novedades en JavaScript), se trata las nuevas características de JavaScript con respecto a HTML5, pero le invito a que se integre rápida y necesariamente al mundo de JavaScript recurriendo a alguna bibliografía especializada en el tema.

1.8 Tecnologías de páginas activas de servidor

Estas tecnologías son sin duda alguna parte muy importante del desarrollo Web hoy por hoy, sin embargo, para llegar a ellas requiere de bases sólidas en HTML. Para interpretarse, las páginas HTML se alojan en el navegador cuando se consultan, a esto se le llama lado cliente, ya que las páginas están literalmente en nuestra computadora o dispositivo, las páginas de servidor están alojadas en la máquina servidor (de ahí el nombre), esto es lo que se conoce como “lado servidor.” Muchas cosas interesantes están pasando gracias a estas tecnologías que se mantienen vigentes y evolucionando, algunas son ASP, PHP, JSP y otras, que se combinan con gestores de datos como Oracle, SQL Server o MySQL para manejar grandes cantidades de información.

Todas estas tecnologías también requerirían libros enteros, por lo que no se abordarán en este libro, pero si usted está interesado en estos temas, existe bibliografía disponible.

Actividades para el lector

1. Investigue las versiones y variantes de HTML.
2. En un reporte enumérelas, liste e identifique sus principales características.
3. Describa cada una de las versiones HTML que se presentaron y que no se describieron en este capítulo.
4. Mencione la funcionalidad de todas y cada una de las versiones de HTML existentes.

RESUMEN

Este capítulo es una introducción al HTML, señalando principalmente:

- Origen e historia del HTML.
- Por qué usar HTML5.
- El sistema de etiquetado del HTML.
- Las tecnologías comúnmente relacionadas con HTML5.

Autoevaluación

1. ¿Para qué es HTML?
2. ¿Por qué usar HTML5?
3. ¿Funciona HTML5 en los navegadores más importantes?
4. ¿En qué casos no se recomienda usar HTML5?
5. ¿Con qué tecnologías funciona paralelamente HTML5?

EVIDENCIA

☐

Investigó, determino y enumeró las versiones y variantes de HTML e identificó sus principales características.

☐

Describió las versiones HTML que ha presentado y que no fueron descritas en este capítulo.

☐

Citó la funcionalidad de todas y cada una de las versiones existentes de HTML.

REFERENCIAS

Bibliografía

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.

Keith, Jeremy (2010). HTML5 For Web Designers, 1a. ed., A Book Apart, EUA.

ORÓS, Juan Carlos (2010) Diseño de páginas Web con XHTML, JavaScript y CSS. 3ª ed., México Alfaomega.



Páginas Web recomendadas

<http://en.wikipedia.org/wiki/HTML>

<http://dev.w3.org/html5/spec/Overview.html>

<http://diveintohtml5.org/>

<http://html5demos.com/>

<http://www.html5rocks.com/>

Respuestas sugeridas a las preguntas de autoevaluación

1. El lenguaje de marcado de hipertexto se utiliza como base para la creación de páginas Web.
2. Proporciona nuevas funcionalidades antes sólo disponibles con recursos de terceros, y estandariza técnicas habituales, pero que no eran reconocidas oficialmente.
3. Sí, pero no todos tienen el mismo nivel de soporte, ya que la especificación HTML5 aún no está terminada.
4. No es conveniente usar HTML5 si la compatibilidad internavegador y con navegadores o máquinas viejas es demasiado importante.
5. HTML5 requiere paralelamente de CSS y JavaScript para utilizar toda su funcionalidad.

Preparándonos para HTML5

2

Reflexione y responda las preguntas siguientes:

¿Cuáles navegadores soportan HTML5?

¿Cómo crear un archivo HTML?

¿Cuál es la estructura básica de un documento HTML?

¿Qué es y cómo usar la librería *Modernizr*?

Contenido

Preparándonos para HTML5

Introducción

2.1 Eligiendo un navegador

2.2 Tipo de documento

2.3 Estructura básica de un documento HTML

2.4 Título y Metadatos

2.5 Párrafos y saltos de línea

2.6 Detección de las características de HTML5

Expectativa

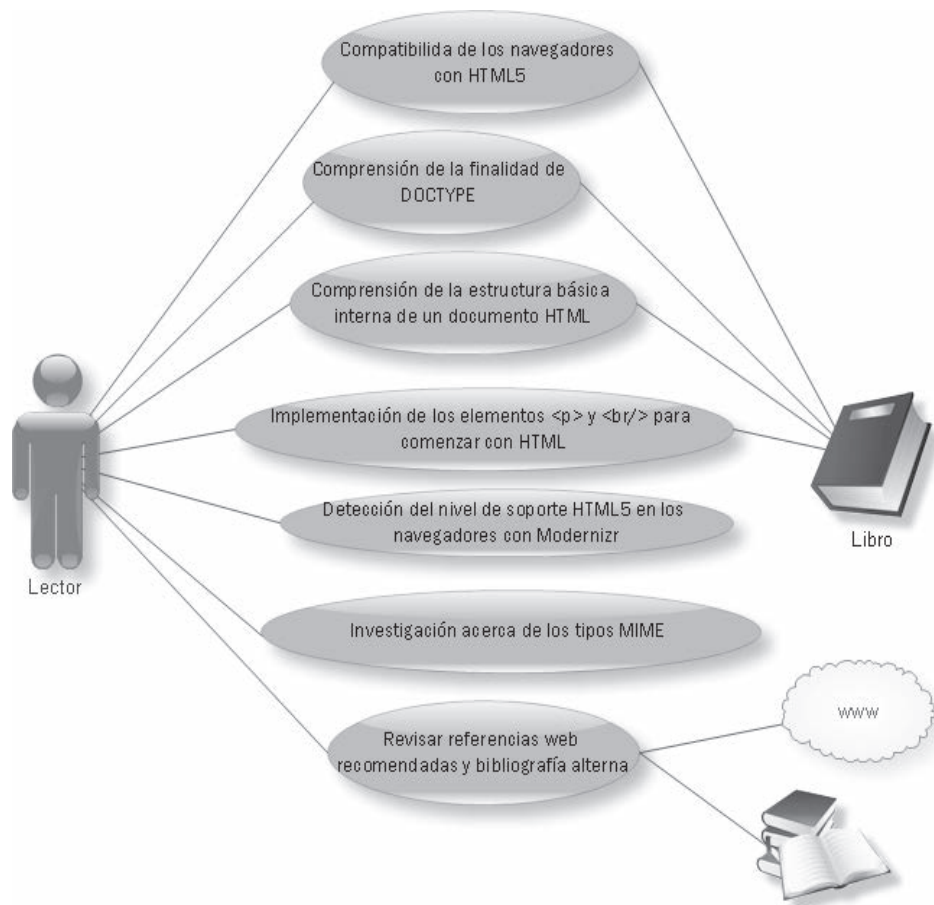
Saber en qué consiste un documento HTML.

Conocer el nivel de soporte que brinda a HTML5 un navegador.

Crear sus primeros documentos bajo la nueva especificación HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Conocer las capacidades HTML5 de un navegador.
- Crear la estructura básica de un documento HTML5.
- Crear sus primeros documentos HTML5.
- Implementar las etiquetas para párrafos y saltos de línea.



INTRODUCCIÓN

En mi opinión la mejor forma de aprender es “haciendo”, y de esta manera se trabajará a lo largo de este libro, se utilizará el editor de texto más simple que ofrezca su sistema operativo (Bloc de notas en el caso de Windows) para crear y modificar archivos, para posteriormente visualizar sus resultados en un navegador.

Nota

Algunos lectores con cierta experiencia, se preguntarán por qué usar el Bloc de notas o su equivalente si hay mejores y más sofisticados editores para programar HTML, la respuesta es que mientras más sencillo sea el editor, menos ayuda ofrece y mientras menos ayuda se tenga del editor de texto, será necesario escribir mayor cantidad de código, y por lo tanto se ayuda a recordar mejor las etiquetas HTML. Si el lector toma este consejo, a la larga no se arrepentirá.

2.1 Eligiendo un navegador

Como se ha mencionado, el estándar HTML5 no ha sido aceptado oficialmente, por lo que el lector tiene que asegurarse de utilizar un navegador que soporte la mayoría de sus características.

Nota

No existe aún un navegador que soporte absolutamente todas las características de HTML5, cada fabricante ha intentado añadir las propias especificaciones y características para darle “ventaja” a su navegador. Después de esto se preguntará si vale la pena aprender HTML5. ¡Por supuesto que sí!



Fig. 2.1 HTML5 en distintos navegadores.

Los grandes han aprobado la mayor parte de las ideas del estándar. *Microsoft*, *Google*, *Apple*, *Mozilla Foundation* han anunciado su soporte para HTML5 y las ideas más importantes ya están disponibles en las versiones más recientes de sus navegadores y están alentando a usuarios y desarrolladores para usar HTML5.

HTML5 genera mejores hábitos de codificación. La separación de código de presentación y de estructura es más adecuada en el desarrollo de programación actual, además de obligar a usar un código más consistente al no ser tan permisivo.

En el momento de escribir estas líneas, Google Chrome 12, Microsoft Internet Explorer 9, Firefox 5, Opera 11 y Apple Safari 5 dicen soportar HTML5, pero en realidad soportan muchas características comunes del estándar, pero otras son soportadas por un navegador y por otro no. Hay algunas técnicas para detectar el soporte HTML5 de un navegador, pero son tan vertiginosas las mejoras en los navegadores para HTML5 que hacen extremadamente difícil mantener la pista de lo que está pasando, actualizaciones o nuevas versiones de los navegadores están disponibles todo el tiempo. Por esta razón el autor ha hecho una página para ayudarle a ver qué elementos HTML5 son soportados por su navegador.

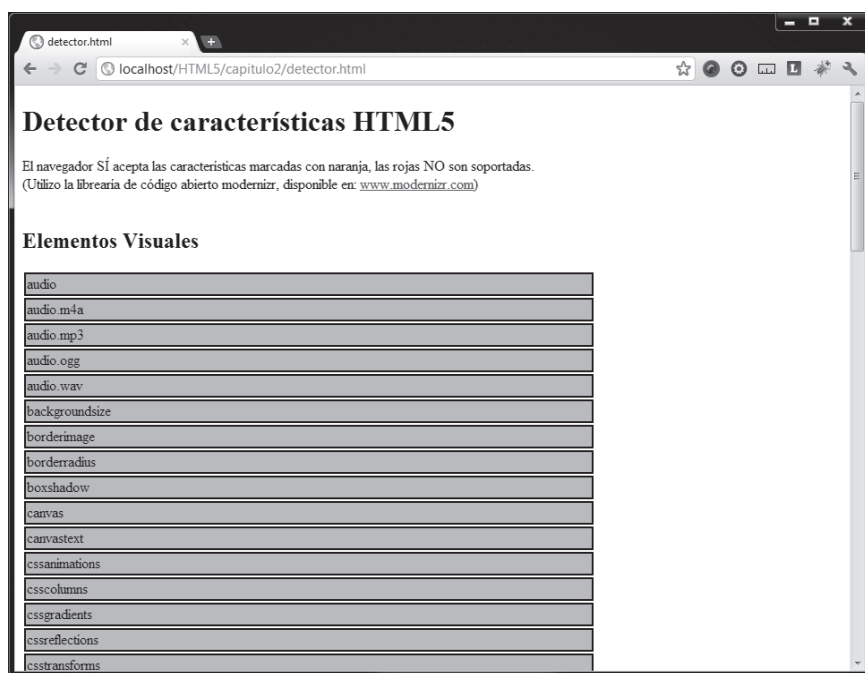


Fig. 2.2 Aspecto de la página del detector de características HTML.



En esta página el autor utilizó la librería JavaScript *Modernizr* para detectar las características disponibles en su navegador. En los materiales adicionales encontrará los archivos: *detector.html* y *modernizr-2.0.js*

El navegador que le convenga será aquel que cubra con los elementos que desea utilizar, en este libro se utilizará Google Chrome, porque al momento de escribir estas líneas es el navegador que mayor soporte tiene para HTML5 y es el más rápido ejecutándolo, esto puede cambiar en cualquier momento, estamos en medio de una “guerra de navegadores” y algún otro puede tomar el liderazgo.

Si desea utilizar un navegador diferente a Chrome, siéntase con la libertad de hacerlo, únicamente asegúrese de utilizar la herramienta de detección, existe la posibilidad de que algún ejemplo no funcione en otro navegador.

2.2 Tipo de documento

Hasta los rebeldes más osados y las personas más aventureras, en algún momento se dieron cuenta de que para lograr ciertos objetivos necesitaban un plan o estrategia, llegar a acuerdos e incluso crear reglas para lograr tal objetivo, de lo contrario simplemente no podían alcanzar lo que buscaban.

La programación en general y por ende el HTML, no es la excepción a la necesidad de orden. Usted puede lograr ser enormemente creativo cuando hace contenido Web, pero debe de seguir cierta estructura para que sus páginas puedan ser interpretadas correctamente por un navegador.

2.2.1 Etiqueta `<!DOCTYPE html>`

El fijar un estándar siempre ha sido un problema, y una larga cadena de tensión e inconvenientes ha rodeado al HTML como estándar a lo largo de su implementación entre navegadores, como solución a uno de estos problemas surgió la etiqueta DOCTYPE, que en HTML5 se observa de esta manera:

```
<!DOCTYPE html>
```

Con esta etiqueta se pueden establecer hasta hoy 15 modos de estándar diferentes, pero la expuesta es la más corta, amena y simple que proporciona HTML5 (*Standard Mode*). DOCTYPE siempre debe escribirse al principio del documento, en mayúsculas y con un signo de exclamación al comienzo.

Si el navegador encuentra `<!DOCTYPE html>` asumirá que está utilizando HTML5 y usará el *Standard Mode*, pero de no hacerlo el navegador interpretará que hay algo peculiar en el documento y utilizará el modo llamado *Quirk Mode*, incluso si sólo hay un simple espacio antes de la etiqueta.

No deseo abrumar al lector novel con todo este asunto de los diferentes modos de DOCTYPE, porque en realidad estas distinciones son fruto de aquellos tempranos días en el desarrollo de antiguas versiones del HTML y de antiguas versiones de navegadores en los distintos sistemas operativos. Para efectos prácticos, en HTML5 se puede decir que `<!DOCTYPE html>` es la manera en que se asegura al navegador que en su página no habrá nada más que código HTML5.

2.3 Estructura básica de un documento HTML

Antes de continuar, abra el editor de texto más simple que tenga, la mayoría de los editores de texto simple guardan por default un archivo con extensión .txt.

Una vez que escriba su código, guarde sus documentos con las extensiones .htm o .html, esto es importante, ya que de lo contrario ningún navegador interpretará como una página HTML al archivo, evite guardar sus archivos con extensión .txt.

Con el editor de texto abierto coloque la primera etiqueta que requiere HTML5, que es por supuesto DOCTYPE, enseguida coloque una etiqueta <html>, ésta es una etiqueta que requiere cierre, por lo tanto, consta de dos partes, las etiquetas <html> y </html> que funcionan como delimitador de todo el resto de etiquetas en el documento (excepto de DOCTYPE). Hasta este momento su código debe verse así:

```
<!DOCTYPE html>

<html>

</html>
```

2.3.1 Etiquetas <head> y <body>

Para seguir con una estructura correcta, su documento debe tener dos etiquetas que costan de dos partes: las etiquetas <head> y <body>. Dentro de la etiqueta <head> use otra etiqueta que es destinada a darle título a la página, también de dos partes llamada <title>. Hay otras etiquetas que también se colocan en la sección Head, como las etiqueta para colocar scripts (la cual se usará en este libro para JavaScript) y la etiqueta para metadatos <meta> que sirve para especificar información acerca de sus páginas. Se abordará con más detalle las etiquetas <title> y <meta> en la sección 2.4 Título y Metadatos.

Ahora implemente estas etiquetas para las secciones de cabecera (<head> y cuerpo (<body>) de su página, escriba el siguiente código:

```
<!DOCTYPE html>

<html>

    <head>

        <title></title>

        <meta encoding="iso-8859-1" />

    </head>
```

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

Escriba este código en su editor de texto y guárdelo como *primer_html5.html* en la ubicación que le resulte más conveniente, pero procure tenerlo a la mano, ya que este archivo servirá como base para próximos ejercicios y ejemplos.

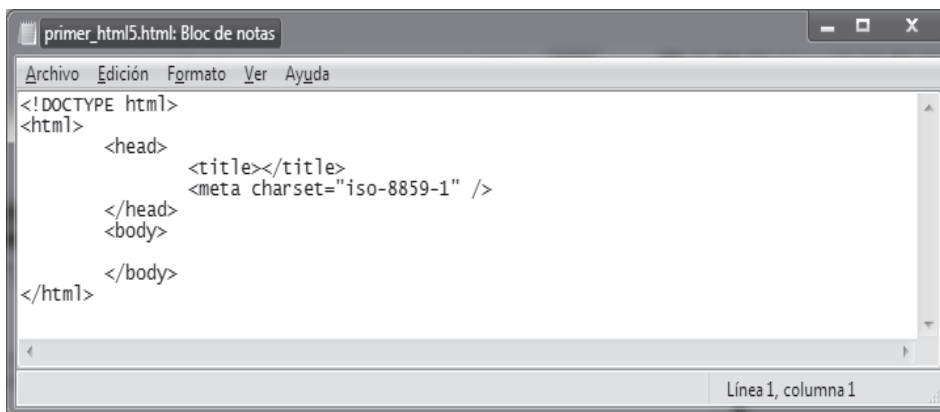


Fig. 2.3 Se muestra cómo se ve en Bloc de notas el código que ha escrito.



En los materiales adicionales encontrará el archivo *primer_html5.html*

2.4 Título y Metadatos

Como se mencionó en la sección `<head>`, se puede indicar el título de una página con las etiquetas `<title>` y `</title>`, las cuales tienen un efecto usualmente visible que aparece en las barras de título del navegador que utilice.

Ejemplo

```
<head>
<title>Tienda de Mascotas</title>
</head>
```

La W3C ha enfatizado siempre que un documento debe tener “significado” en un ambiente estructurado de información, y que el HTML es justamente la manera para lograr esa estructura. Los metadatos son una manera de atender ese punto de

vista, se trata de dar “información sobre nuestra información”, para ello en sus documentos, en la sección <head> utilice la etiqueta vacía (sin cierre) <meta>.

Esta etiqueta tiene varios propósitos, puede colocar palabras claves que tengan relación con información que presentan sus documentos, de esa manera facilita a las personas y a algunos sistemas como los buscadores (yahoo, bing, etc.) encontrar su página y colocarla en sus resultados de búsqueda (indexar su página). Imagine que tiene una tienda de mascotas y su página habla de gatos, aves, perros, roedores e insectos, usted querría poner algunas palabras clave como metadato. Podría usar el siguiente código en la sección <head>:

```
<head>
  <title>Tienda de Mascotas</title>
  <meta name="keywords"
content="mascotas,cahorros,perros,gatos,hamsters" />
</head>
```

Otro uso común es indicar la codificación de caracteres que se desea utilizar. Esto es básicamente para indicar el juego de letras, números y símbolos que la página mostrará, si se desplegó normalmente información en inglés, no representa gran problema, pero se considera un buen hábito hacerlo, normalmente para el juego de caracteres en inglés se usa la codificación UTF-8. En el caso del idioma español la codificación más utilizada es la ISO-8859-1 (para evitar problemas con acentos o con la letra ñ).

```
<head>
  <title>Tienda de Mascotas</title>
  <meta name="keywords"
content="mascotas,cahorros,perros,gatos,hamsters" />
  <meta encoding="iso-8859-1" />
</head>
```

También puede usarse esta etiqueta para redireccionar a los visitantes de su página a otra URL. Suponga que está dando mantenimiento a sus páginas y mientras tanto desea que todas las personas que accedan a su página vean un anuncio de “En mantenimiento”, sus visitas serán enviadas a la nueva dirección usando la URL indicada en 3 segundos, por ejemplo:

```
<meta http-equiv="refresh" content="3";
url="http://www.midominio.com/mantenimiento.html" />
```

Ejercicio

1. Abra el archivo *primer_html5.html* ya que se usará como base.
2. Coloque el título “Tienda de Mascotas”.
3. Coloque los metadatos para palabras claves “mascotas, cahorros, perros, gatos, hamsters” y para codificación ISO-8859-1.
4. Al terminar guarde el archivo con el nombre de *principal.html* en el mismo directorio del primer ejercicio (*primer_html5.html*). La Fig. 2.4 muestra cómo queda el código de su archivo.

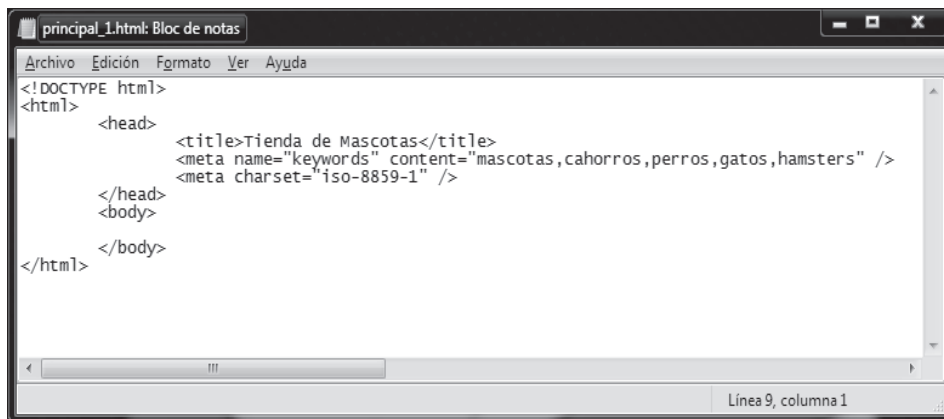


Fig. 2.4 Código de documento básico HTML con metadatos.

2.5 Párrafos y saltos de línea

Todos los elementos que efectivamente se visualizarán en el navegador estarán dentro del ámbito de `<body>` y `</body>`. Cuando se emplea párrafos en HTML ocupe la etiqueta con cierre `<p>`, en la que el texto de párrafo se encontrará delimitado por las dos partes de la etiqueta, es decir, entre `<p>` y `</p>`, como sigue:

```
<p>Nuestro texto</p>
```

El navegador siempre inserta un espacio vertical después de cada párrafo, de manera que si por ejemplo se colocan párrafos consecutivos, como a continuación, habrá un espacio entre ambos:

```
<p>Bienvenido(a) a la Tienda de Mascotas!</p>
<p>Nuestra misión es facilitarle el camino para hacer de una de
nuestras mascotas un gran compañero para usted y toda su familia,
disfrute la visita a nuestro sitio y esperamos servirle pronto en
nuestra tienda</p>
```

La figura 2.5 muestra el aspecto que tiene el código anterior cuando lo prueba en el navegador.

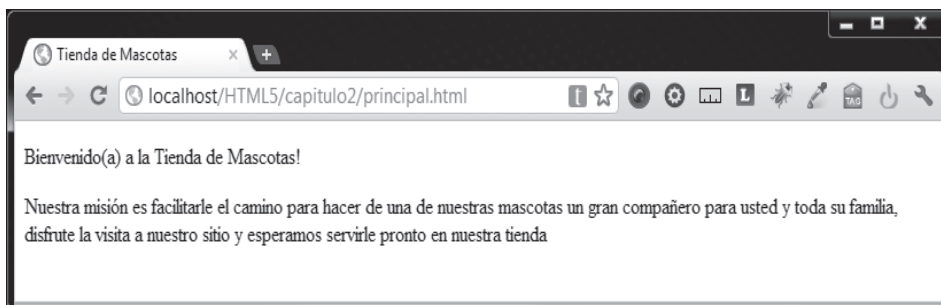


Fig. 2.5 Página que utiliza párrafos `<p>`.

Muchas veces el espacio vertical entre párrafos no es algo deseable para la estética de sus páginas, por ejemplo, cuando se quiere hacer un breve resumen de indicaciones o especificaciones.

Para insertar un salto de línea, simplemente se utiliza la etiqueta `
`, esta etiqueta no requiere cierre. Hay que aclarar que el uso de `
` no depende de ninguna manera de la etiqueta `<p>`, por tanto se puede utilizar dentro o fuera de un párrafo o sin ningún párrafo presente, como se muestra:

```
<p>Correo electrónico:</p><br />prueba@midominio.com
```

o bien, si así lo desea:

```
<p>Correo electrónico:<br />prueba@midominio.com</p>
```

Modifique el archivo *principal.html* para agregar un párrafo con la leyenda “Bienvenido(a) a la Tienda de Mascotas!”, después el pequeño mensaje informativo “Nuestra misión es facilitarle el camino para hacer de una de nuestras mascotas un gran compañero para usted y toda su familia, disfrute la visita a nuestro sitio y esperamos servirle pronto en nuestra tienda”. Finalmente, utilizando un párrafo y dentro saltos de línea, coloque lo siguiente:

```
Nuestra Dirección:  
Calle: San Ciprián No. 45  
Colonia: Centro  
México D.F.  
C.P. 07640
```

La Fig. 2.6 muestra cómo se verá la página en el navegador, una vez salvada.



Fig. 2.6 Página que utiliza saltos de línea `
`.

Apoyo en la



En los materiales adicionales encontrará el archivo *principal.html*

2.6 Detección de las características de HTML5

No quiero dejar pasar la técnica que utilicé para hacer el detector de características, porque es probable que algunos lectores con algo de experiencia deseen saber cómo lo hice, pero si usted es aún un principiante, no se preocupe si no entiende totalmente el código expuesto, si continúa consultando este libro lo entenderá por completo antes de lo que cree.

Todas las técnicas que hay para detectar la viabilidad de HTML5 se basan en intentar crear un elemento, para después llamar a alguna propiedad de este elemento y ver si efectivamente está ahí. Como mencioné anteriormente, utilicé la popular librería de código libre llamada *Modernizr* licenciada por el MIT, los pasos básicos para hacer su propio detector son los siguientes:

1. Descargue la librería *Modernizr* del sitio oficial <http://www.modernizr.com> y coloque el archivo de esta librería en el mismo directorio donde se encontrará su página (utilice la última disponible).
2. Abra el archivo *primer_html5.html* y guárdelo como *midetector.html* (si prefiere hacer todo desde cero, adelante).
3. Dentro de la cabecera de su página (etiqueta `<head>`) coloque una etiqueta para poder leer JavaScript, *Modernizr* es esencialmente un archivo JavaScript con extensión `.js` que se invocará desde su página.

```
<script src="modernizr-2.0.js"></script>
```

- Hay que añadir una clase que requiere *Modernizr* para dejarle saber a las hojas de estilo (CSS) si JavaScript está o no disponible en el navegador, ésta se coloca en la etiqueta `<html>` como sigue:

```
<html class="no-js">
```

- Coloque un contenedor para depositar el resultado obtenido del análisis con *Modernizr*, para esto se usa la etiqueta `<div>` con el identificador "resultado" y la indicación de disparar la función desde la etiqueta `<body>` con su evento `onload` llamando a la función JavaScript `detectar()`.

```
<body onload="detectar()">
  <div id="resultado">
  </div>
</body>
```

- Finalmente se debe colocar una función llamada en nuestro caso "detectar" que utiliza *Modernizr* para averiguar si una característica HTML5 existe, para fines del ejemplo se detectará una característica que está presente en todos los navegadores que soportan HTML5, ésta es la etiqueta `<canvas>`. Para enriquecer un poco el detector, haga que le indique si está disponible una característica HTML5 para gráficos en 3D llamada WebGL disponible hasta ahora sólo en Chrome y en Firefox en sus últimas versiones, nuestra función `detectar`:

```
<script type = "text/javascript">
function detectar(){
  if (Modernizr.canvas){
    document.getElementById("resultado").innerHTML = "El navegador
puede usar la etiqueta canvas :))) <br />";
  }else{
    document.getElementById("resultado").innerHTML = "El navegador
no puede usar la etiqueta canvas :( <br />";
  }
  if (Modernizr.webgl){
    document.getElementById("resultado").innerHTML += "El navegador
puede usar WebGL :))) ";
  }else{
    document.getElementById("resultado").innerHTML += "El navegador
no puede usar WebGL :( ";
  }
}
</script>
```

La Fig. 2.7 muestra la página en ejecución.



Fig. 2.7 Página en ejecución que utiliza *Modernizr*.

Nota

Si desea hacer aún más sofisticado su detector de características HTML5 requiere entonces ir más a fondo en la librería *Modernizr* y mejorar el código JavaScript.

Actividades para el lector

1. Investigue qué son los tipos MIME.
2. Haga una tabla donde indique los tipos MIME, sus características y acciones.
3. Determine el tipo MIME utilizado implícitamente por HTML5.
- 4.- Utilice el detector de características HTML5 y compare las capacidades de por menos dos navegadores.
- 5.-Elabore un documento HTML5 cuya estructura básica sea: título, metadatos y dos párrafos incluya un salto de línea entre ellos.

RESUMEN

En este capítulo se han analizado los puntos más esenciales para poder a comenzar a desarrollar bajo la especificación HTML5:

- Navegadores que soportan HTML5 y cómo elegir uno.
- Como crear la estructura base de un HTML5.
- Utilizar los elementos HTML para crear párrafos y saltos de página para familiarizarse con estos elementos.
- Filosofía de etiquetado de HTML.
- Cómo establecer el nivel de soporte de HTML5 por parte de un navegador utilizando la librería *Modernizr*.

Autoevaluación

1. ¿Qué navegador es el más avanzado en cuanto a soporte para HTML5?
2. ¿Cómo indica en un documento HTML que debe ser interpretado como HTML5?
3. ¿Qué elemento indica el encabezado de un documento HTML?
4. ¿Qué elemento indica el cuerpo de un documento HTML?
5. ¿Qué es *Modernizr*?

EVIDENCIA

☐

Investigó los tipos MIME, en un tabla indica sus características y acciones.

☐

Determinó que tipo MIME utiliza HTML5

☐

Comparó y documentó las capacidades de diferentes navegadores con el detector de características HTML5.

☐

Implementó la estructura básica de un documento HTML5.

☐

Utilizó párrafos y saltos de línea, en documentos HTML5.

REFERENCIAS

Bibliografía

Pilgrim, Mark, (2010). HTML5: Up and Running, 1a. ed., O'Reilly, Estados Unidos de América.

Wempen, Faithe, (2011). HTML5: Step by Step 1a. ed., O'Reilly, Canadá.



Páginas Web recomendadas

<http://www.modernizr.com/>

<http://dev.w3.org/html5/spec/Overview.html>

<http://diveintohtml5.org/>

Respuestas sugeridas a las preguntas de autoevaluación

1. Google Chrome es el navegador con mayor soporte para HTML5 (Octubre de 2011)
2. Utilizando el elemento DOCTYPE (`<!DOCTYPE html>`).
3. El elemento `<head>`.
4. El elemento `<body>`.
5. Es una librería de código libre proporcionada por el MIT que facilita la detección del soporte de los navegadores para elemento HTML5.

Más bases HTML

3

Reflexione y responda las preguntas siguientes:

¿Qué es XHTML y cuál es la diferencia con HTML?

¿Cómo enriquecer el contenido de los documentos HTML?

¿Qué son y para qué sirven los formularios HTML?

¿Cómo validar que las páginas construidas cumplan con la especificación HTML5?

Contenido

Más bases HTML

Introducción

3.1 ¿HTML o XHTML?

3.2 Formato de texto con etiquetas

3.3 Listas

3.4 Imágenes

3.5 Tablas

3.6 Hipervínculos (enlaces)

3.7 Formularios (controles clásicos)

3.8 Validar páginas HTML5

Expectativa

Fortalecer y ampliar sus conocimientos de los fundamentos de HTML.

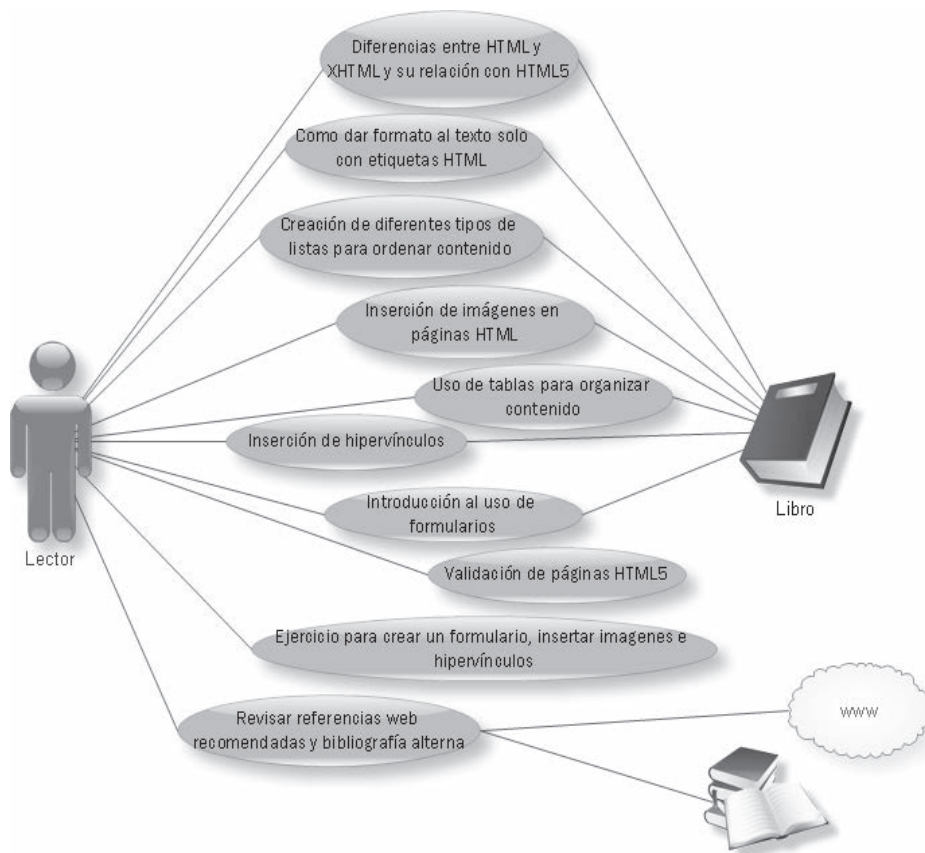
Comprender algunos elementos para manipular la apariencia de documentos HTML.

Conocer la construcción de formularios.

Validar la correcta formación de documentos bajo la especificación HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Conocer la diferencia entre HTML y XHTML.
- Manipular la estructura visual de una página.
- Enriquecer con más elementos las páginas HTML.
- Crear formularios con los elementos clásicos de HTML.
- Validar sus propias páginas HTML5.



INTRODUCCIÓN

Continuando con el viaje en HTML, hay puntos interesantes e incluso polémicos acerca del HTML, en este capítulo se fortalecerán los fundamentos del lenguaje y al mismo tiempo hará más ricas sus páginas con el uso de imágenes, enlaces, formatos de texto y algunos métodos de organización visual como tablas y listas.

3.1 ¿HTML o XHTML?

XHTML, acrónimo de eXtensible Hypertext Markup Language (Lenguaje Extensible de Marcado de Hipertexto), es el lenguaje pensado para sustituir a HTML como estándar para las páginas Web. En su versión 1.0, XHTML es solamente la versión XML de HTML, por lo que tiene, básicamente, las mismas funcionalidades, pero cumple especificaciones sintácticas más estrictas, como lo hace un documento XML. Su objetivo... lograr una Web semántica, donde la información y la forma de presentarla estén claramente separadas (objetivo presente también en HTML5). La versión 1.1 parte a la especificación en módulos.

Ahora la parte polémica; a pesar de las nobles intenciones del XHTML la verdad es que no ha logrado reemplazar al HTML, ya que en su fortaleza lleva consigo la carga de su propia debilidad, la razón es que mientras algunos programadores sostienen que XHTML es de acabado más profesional y de mejor nivel, además de que permite cierto nivel de validación para encontrar errores, lo que obliga a ser más claro y consistente al escribir código, otros programadores argumentan que simplemente significa trabajar más sin una necesidad real, y esta afirmación, trágicamente parece tener cierto viso de verdad, cuando se descubre que el 99% de las páginas hoy publicadas en la Web no cumplen cabalmente con las exigencias del XHTML, incluyendo la mayor parte de las páginas que claman hacerlo.

Algunas diferencias que existen entre HTML y XHTML.

Los elementos vacíos (sin etiqueta de cierre) deben cerrarse siempre:

Incorrecto: `
`

Correcto: `
</br>` o `
` o `
`

Los elementos no vacíos, también deben cerrarse siempre:

Incorrecto: `<p>Primer párrafo<p>Segundo párrafo`

Correcto: `<p>Primer párrafo</p><p>Segundo párrafo</p>`

La W3C trabajó en la versión XHTML 2.0, sin embargo, publicaron que este esfuerzo no seguiría siendo impulsado y que HTML5 sería el estándar apoyado.

Se anunció que se trabaja paralelamente a HTML5 con una versión serializada como XML llamada XHTML5, la única diferencia entre HTML5 y XHTML5 es el tipo MIME que se utiliza (`application/xhtml+xml` o `application/xml`) y que la etiqueta DOCTYPE es opcional.

Nota

HTML5 aún no es oficialmente un estándar, ni rechaza a XHTML, de hecho toma las mejores cosas de la sintaxis de HTML4 y XHTML.

La W3C no se ha manifestado con relación a XHTML5, pero todo indica que esto es un esfuerzo para serializar el contenido de HTML5 y ayudar a que aplicaciones especiales que leen el código de una página, pero no la visualizan como lo haría un navegador, lo hagan más fácilmente.

- **Validación y tradición estricta.** Usted puede validar sus documentos con útiles herramientas que están emergiendo en la red, se usará el validador de la W3C (<http://validator.w3.org>). El estándar en la sintaxis de HTML5 se acerca más a XHTML que a HTML4, la brecha entre el código “flojo” y XHTML será mucho más corta.
- **DOCTYPE simple.** Nuevamente es posible recordar el DOCTYPE y usar un editor simple sin tener que copiarlo de algún lugar.
- **Separación de contenido y apariencia.** Muchas etiquetas han perdido los atributos que afectaban su apariencia en favor del uso de los estilos CSS.
- **Mejor integración con otros lenguajes.** A pesar de que el HTML5 sigue haciendo de HTML la piedra angular en la Web, cede paso a otras tecnologías como CSS, JavaScript o páginas de servidor para aumentar la versatilidad y poder de sus páginas.
- **Nuevas características.** HTML5 integra nuevas características de otras tecnologías, en esto se pondrá especial atención en este libro.

Notas

La traducción de la especificación oficial se encuentra en:

<http://www.sidar.org/recur/desdi/traduc/es/xhtmll/xhtmll-basic.html>

Aunque HTML5 no es XHTML, los ejemplos y código del libro se basan en la sintaxis de XHTML, para ayudar al lector a fomentar buenos hábitos de programación.

3.2 Formato de texto con etiquetas

Cuando se escribe algo en un procesador de texto, en el cliente de correo, en el mensajero instantáneo o incluso en un papel, existen ocasiones en que se quiere enfatizar, separar o resaltar algo; es en ese momento cuando se utilizan letras en negritas para alguna frase, se usan los formatos itálico, títulos o subtítulos para organizar contenido. HTML no es la excepción a esas necesidades.

HTML5 incluye algunas etiquetas para controlar el formato de un texto. Por ahora aprenderá estas etiquetas y quedará, para otro capítulo, el estudio de las hojas de estilo (CSS), que mejoran la apariencia de sus páginas Web.

Si lo que se busca es separar el código de estructura del código para dar apariencia ¿por qué siguen existiendo etiquetas de formato para texto?, la razón es porque se trata más de un tema de semántica que de formato, es decir, que existen situaciones en las que es de mayor prioridad el sentido de nuestro documento, que el cómo se ve, esto se puntualizará cuando se trate el tema de cabeceras.

Nota

Cabe aclarar a los lectores más experimentados que en este libro de HTML5, no se utilizará etiquetas aún soportadas por los navegadores, pero consideradas obsoletas por el HTML5, como son, entre otras: `<basefont>`, `` o `<center>`

3.2.1 Cabeceras

La idea de las cabeceras o encabezados (*headers* en inglés) es muy simple en realidad, únicamente tienen el objetivo de ayudar a organizar los textos en secciones, de la misma forma que se usan títulos y subtítulos en un documento común y corriente. Las etiquetas de cabecera requieren cierre y utilizan números del 1 al 6 para indicar su tamaño, siendo 1 el título de fuente más grande y 6 el de fuente más pequeña, es decir, se usa desde `<h1>` hasta `<h6>`. No existen dimensiones específicas para cada tamaño, éste está sujeto a la configuración del navegador donde se desplieguen los *headers*, hay que tener esto en cuenta para decidir si es apropiado usar algún determinado tamaño, ya que usuarios con navegadores y configuraciones diferentes verán también tamaños diferentes.

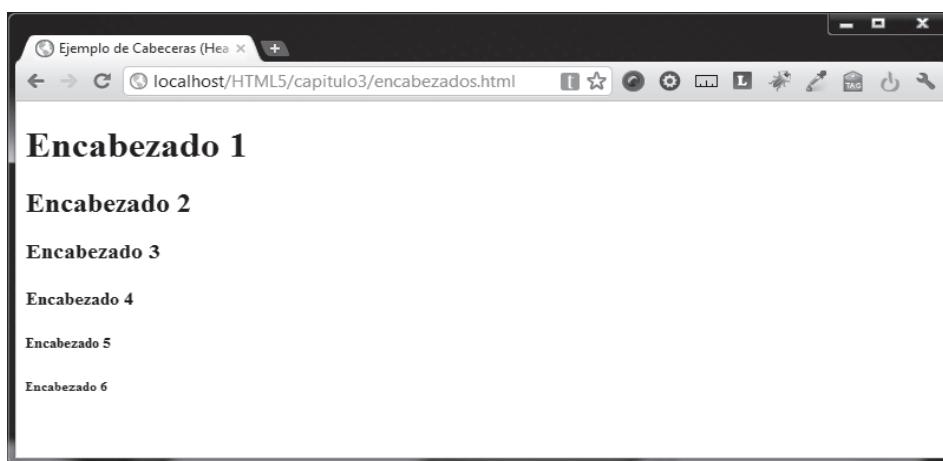


Fig. 3.1 Cabeceras desde `<h1>` hasta `<h6>` .

Ejercicio

Agregue algunos encabezados a su página *principal.html*. Inmediatamente después de la etiqueta `<body>`, coloque el siguiente código:

```
<h1>Tienda de Mascotas</h1>
<h2>Reuniendo nuevos mejores amigos!</h2>
```

Justo antes del párrafo (`<p>`) que contiene el texto “Nuestra Dirección:” coloque:

```
<h3>Visítenos!</h3>
```

Al terminar guarde el archivo y consulte el resultado en el navegador.

La figura 3.2 muestra el resultado después de agregar encabezados y ver la página en el navegador.

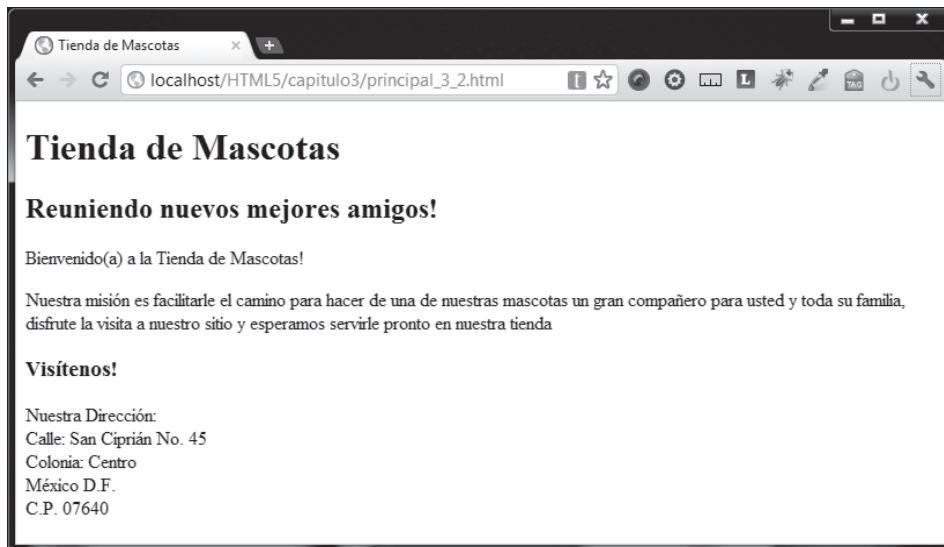


Fig. 3.2 Encabezados `<h1>` hasta `<h3>`.

3.2.2 Negritas, cursiva, subrayado y otros

Para llamar la atención del usuario sobre una letra, palabra o frase, una técnica a la que se recurre con frecuencia en HTML es utilizar texto en negritas (*bold*) o quizá texto con letras itálicas (*italic*), justo como sucede en prácticamente todo los procesadores de palabras actuales.

Para utilizar negritas utilice la etiqueta con cierre `` y para itálicas la etiqueta, también con cierre `<i>`. Para utilizar estas etiquetas simplemente coloque dentro de ellas el texto que se quiere resaltar, por ejemplo:

```
<p>Me ha gustado <b>mucho</b> el postre de hoy</p>  
<p>Tengo que hacer un resumen de la película <i>El Padrino</i> para  
la tarea de esta semana</p>
```

Es posible que usted desee que algún texto se vea con negritas y con itálica al mismo tiempo, para lograr esto usted puede anidar ambas etiquetas como sigue:

```
<p>El título del libro que leí es <b><i>La Evolución de Calpurnia  
Tate</i></b></p>
```

O bien

```
<p>El título del libro que leí es <i><b>La Evolución de Calpurnia  
Tate</b></i></p>
```

Si quiere hacer que una cabecera entera esté en itálicas simplemente haga lo siguiente:

```
<h1><i>Mi cabecera h1</i></h1>
```

Recuerde respetar siempre la estructura de árbol que tienen los documentos HTML, algo como lo siguiente sería incorrecto:

```
<i><b>La Evolución de Calpurnia Tate</i></b>
```

Nota

Algunos navegadores podrían ser permisivos e interpretar “bien” lo que deseamos hacer, pero aun así el código sería incorrecto.

De la misma manera trabaja la etiqueta `<ins>`, que sirve para resaltar un texto con una línea de subrayado:

```
<ins>La Evolución de Calpurnia Tate</ins>
```

Existen más etiquetas para dar formato el texto que trabajan en la misma lógica de las anteriores, se mencionan las ya soportadas por HTML5 (recuerde que HTML5 aún evoluciona y podría añadir más etiquetas o cambiarlas de algún modo).

```
<big>Texto grande</big>  
  
<em>Texto enfatizado</em>
```

```
<small>Texto pequeño</small>

<strong>Texto fuerte</strong>

<del>Texto tachado</del>
```

Ejercicio

En el siguiente ejercicio se usarán negritas e itálicas. Abra nuevamente el archivo *principal.html* y delimite el texto “Reuniendo nuevos mejores amigos!” utilizando una etiqueta `<i>` dentro de `<h2>` para dar al encabezado formato itálico. Una vez hecho esto ubique el primer párrafo (`<p>`) y ponga en negritas con la etiqueta `` el texto “Tienda de Mascotas”, finalmente subraye en el último párrafo el texto “Nuestra Dirección” con la etiqueta `<ins>`. El resultado del ejercicio se muestra en la Fig. 3.3.

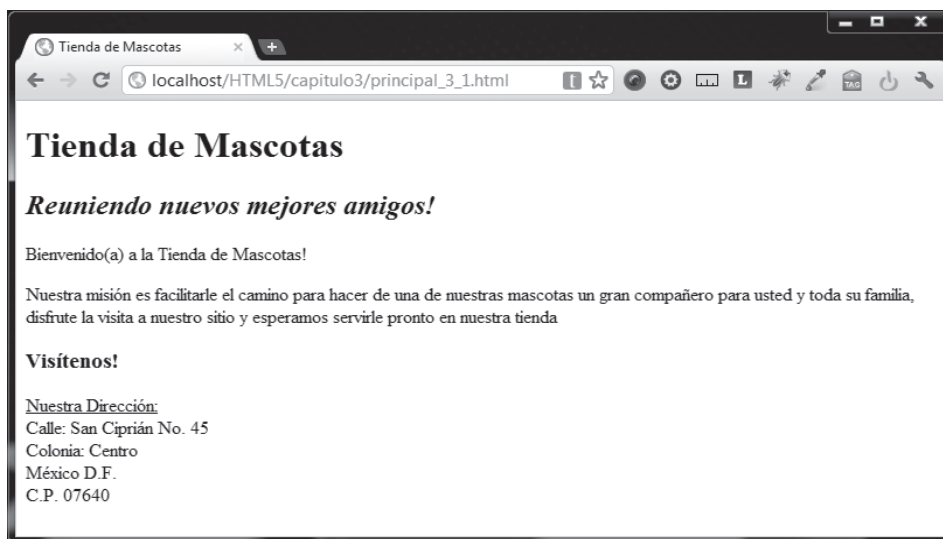


Fig. 3.3 Página con texto itálico, en negrita y subrayado

3.2.3 Subíndices y superíndices

En mi experiencia he encontrado una duda común entre las personas que desean colocar fórmulas o referencias a notas, tienen la duda de cómo colocar números o letras por arriba o por debajo de la línea base del texto. Para solucionar esto, HTML propone un par de alternativas llamadas Superíndice (*Superscript* en inglés) y Subíndice (*Subscript* en inglés). Ahora que sabe cómo usar negritas, itálicas y subrayado, la implementación de la etiqueta `<sup>` para superíndice y la etiqueta `<sub>` para subíndice, ambas con cierre, le resultará sencilla, por lo que se ejemplificarán en el siguiente ejercicio.

Ejercicio

Abra el archivo *principal.html* con el que ha trabajado y modificado, justo después de los primeros 2 párrafos (<p>), coloque el siguiente código:

```
<p>Acuda a nuestra tienda para conocer a nuestras mascotas y a  
nuestro extenso catálogo de juguetes y accesorios disponibles para  
su diversión y cuidado <sup>1</sup></p></p>
```

Ahora, antes de la etiqueta de cierre </body> agregue otro párrafo:

```
<p><sup>1</sup>Venta de comida, productos y accesorios para perros,  
gatos, pájaros, roedores, reptiles y peces.</p>
```

Guarde el resultado y observe el resultado en su navegador.

La Fig. 3.4 muestra la página con el número 1 como superíndice al final del tercer párrafo y al principio de la última línea.

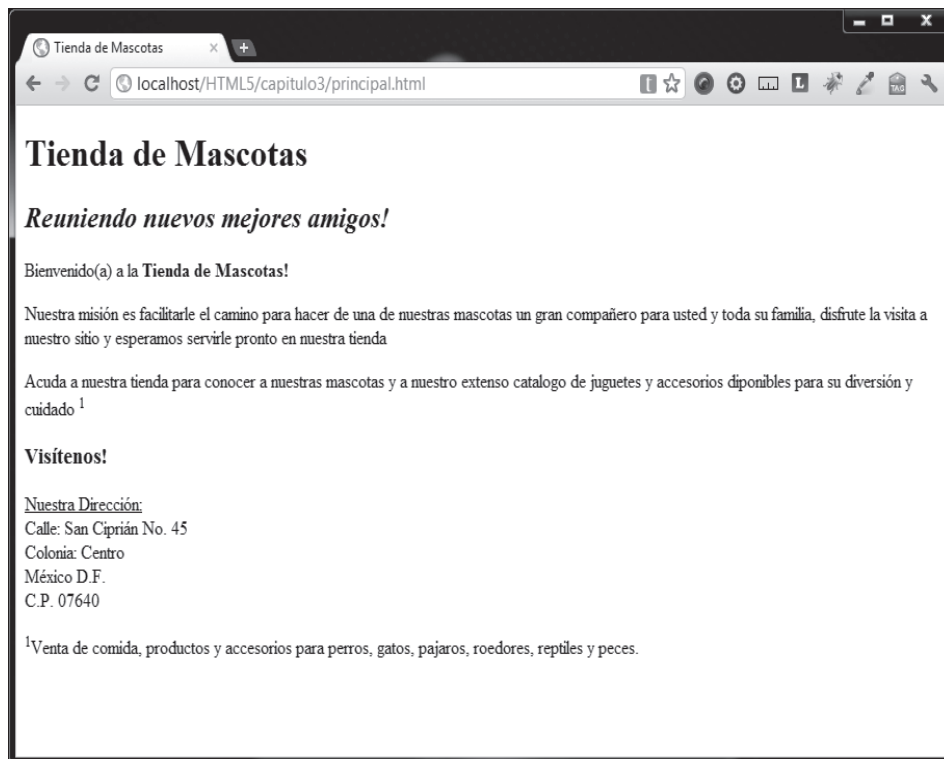


Fig. 3.4. Página con superíndices.

Para incluir un subíndice <sub> se trabaja exactamente de la misma manera, con la única diferencia de que, si se usa como en el ejemplo anterior, el número 1 aparecería por debajo de la línea base del texto común y no por arriba.

3.2.4 Bloque de citas

La etiqueta con cierre <blockquote> da un resultado similar al párrafo, inserta espacio antes y después de un bloque de código, pero también inserta un margen a la izquierda y a la derecha. El resultado visual tiene la finalidad de manejar el texto dentro las etiquetas <blockquote> </blockquote> como los libros lo hacen con una frase que ha dicho alguien más, es decir, manejar el texto como una fuente de información externa, el código sería:

```
<blockquote>El texto que deseamos colocar</blockquote>
```

El bloque de citas <blockquote> tiene un parámetro opcional, *cite*, que sirve para indicar la URL de donde se ha obtenido el texto de la cita, utilizando este atributo el código es similar a éste:

```
<blockquote cite="http://www.midominio.com">Nuestro  
texto</blockquote>
```

Ejercicio

Abra el archivo *primer_html5.html* con la estructura HTML básica guárdelo con el nombre de *albert.html*.

Dentro de las etiquetas <body> coloque un encabezado <h1> con el texto “El genio de cabello alborotado”, después un párrafo <p> con la leyenda “Albert Einstein dijo:” y finalmente un bloque de citas con el siguiente código:

```
<blockquote  
cite="http://www.proverbia.net/citasautor.asp?autor=327">  
Todos somos muy ignorantes. Lo que ocurre es que no todos ignoramos  
las mismas cosas.  
</blockquote>
```

El código final deberá verse como muestra la Fig. 3.5

Fig. 3.5 Código de página *albert.html*.

Finalmente guarde el archivo y vea el resultado en el navegador, en la figura 3.6 muestra el resultado que debería obtener:

Fig. 3.6 Página que utiliza el elemento `</blockquote>`.

Apoyo en la



En los materiales adicionales encontrará el archivo *albert.html*

3.2.5 Línea Horizontal

Hay un elemento vacío (sin cierre) en HTML llamado `<hr />` que no tiene un efecto directo sobre las secciones de texto, pero puede ayudar a organizarlo con respecto a otros bloques de texto o cualquier otro elemento en nuestras páginas. Esto lo hace únicamente trazando una línea horizontal justo donde se coloca la etiqueta `<hr />`.

Edita el archivo *principal.html*, justo después del cierre del bloque de citas `</hgroup>` coloca la etiqueta `<hr />`, el código es el siguiente:

```
<h1>Tienda de Mascotas</h1>
<h2><i>Reuniendo nuevos mejores amigos!</i></h2>
<hr/>
```

Salve su archivo y observe cómo se ve su página.

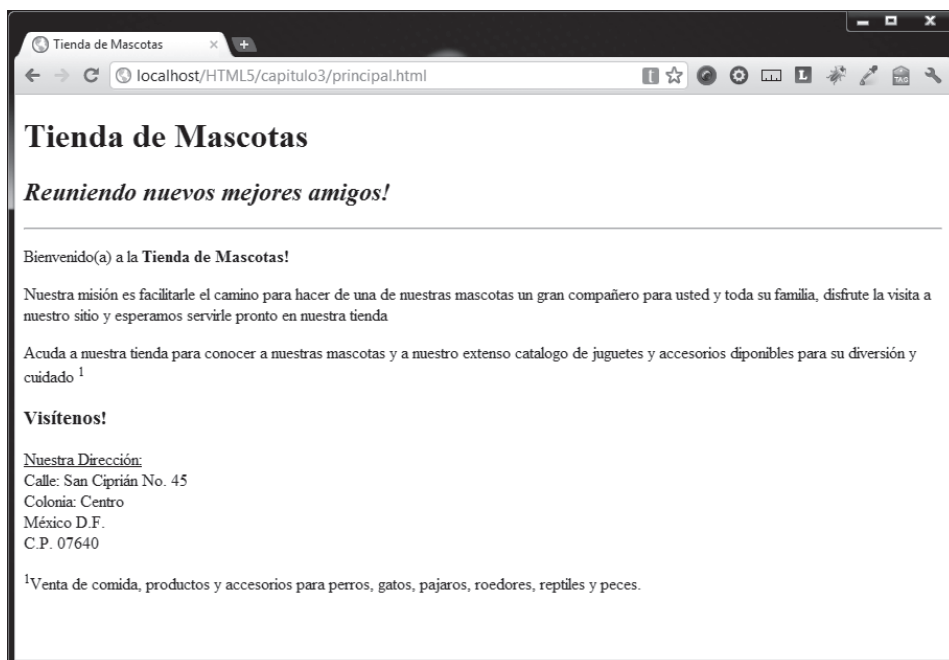


Fig. 3.7 Página que muestra la línea hecha por con `<hr/>`

La figura 3.7 incluye los cambios hechos en los puntos 3.2.1, 3.2.2. y 3.2.3.



En los materiales adicionales encontrará el archivo *principal.html*.

3.3 Listas

Imagine que tiene un documento donde maneja una considerable cantidad de información. Usualmente cuando existe esta situación, usted desea abreviar el tiempo para que cualquier persona encuentre más rápida y eficientemente cierto punto de interés entre toda la cantidad de información disponible, para lograr esto se recurre a elaborar algún tipo de lista (como un índice, por ejemplo).

En este punto se explorarán las posibilidades de HTML con respecto al uso de listas, como son: listas numeradas, conocidas también como listas ordenadas, listas no numeradas llamadas también listas no ordenadas y las listas de definición. También se aprenderá cómo anidar listas dentro de otras.

3.3.1 Listas numeradas y listas no numeradas

La etiqueta con cierre para listas no numeradas es ``, cada elemento de la lista utilizará un símbolo llamado *bullet*. Para listas numeradas se usa la etiqueta ``, aquí, a cada elemento de la lista le precederá un número. Al iniciar la lista se utiliza la etiqueta `` o bien la etiqueta ``, se coloca cada elemento de la lista delimitado por la etiqueta `` y su cierre ``, y de inmediato se cierran estas listas con `` o `` según sea el caso.

Observe el código utilizando `` (una lista no numerada):

```
<ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
  <li>Elemento 4</li>
</ul>
```

Una lista numerada trabaja de la misma manera pero utilizando la etiqueta ``:

```
<ol>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
  <li>Elemento 4</li>
</ol>
```

Si desea puede anidar listas, colocando una dentro de otra. En el siguiente código se colocará una lista numerada dentro de una lista no numerada:

```
<ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>
    <ol>
      <li>Elemento 1</li>
      <li>Elemento 2</li>
      <li>Elemento 3</li>
    </ol>
  </li>
</ul>
```



```
</li>
    <li>Elemento 4</li>
</ul>
```

Note cómo la lista no numerada `` está dentro de una etiqueta de elemento ``, esto es porque la lista no numerada es un elemento de la lista padre ``.

Pruebe los códigos anteriores. Utilice el archivo *primer_html5.html* y guárdelo como *listas.html*, enseguida dentro de la sección `<body>...</body>` coloque los tres ejemplos de código anteriores. Guarde el archivo y vea el resultado en su navegador.

El ejercicio se ve como se muestra en la Fig. 3.8.

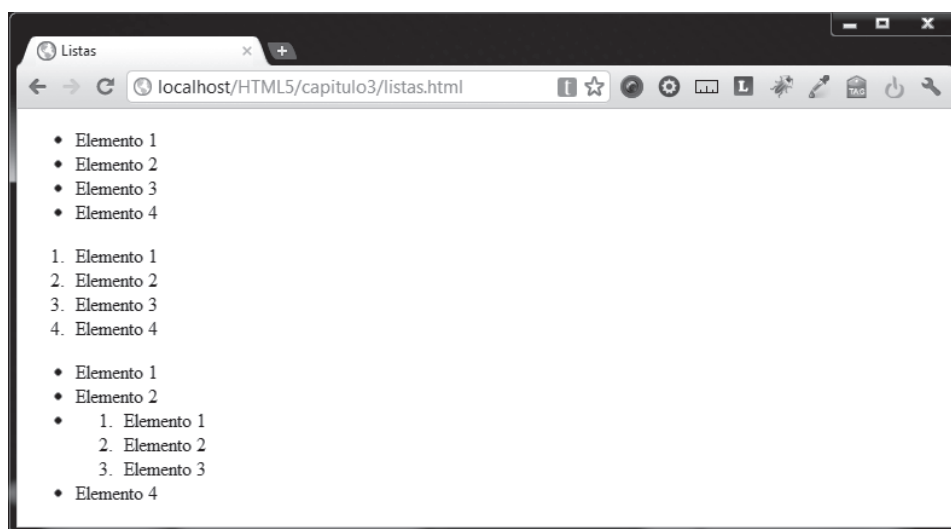


Fig. 3.8 Muestra de lista no numeradas, numerada y anidadas.



En los materiales adicionales encontrará el archivo *listas.html*.

3.3.2 Listas de definición

Las listas de definición son justamente eso, listas que muestran definiciones, para ellas se utilizan las etiquetas con cierre `<dl>` (que se origina del término en inglés *definition list*), la etiqueta `<dt>` (del término en inglés *definition term*) y la etiqueta `<dd>` (del inglés *definition description*).

Ejemplo

```
<dl>
  <dt>Pastor Alemán</dt>
  <dd>Éste es un perro de pastor robusto, bien musculoso, de talla
mediana, orejas erguidas y pelo doble de longitud mediana a larga.
  </dd>
  <dt>Esquimal Canadiense</dt>
  <dd>Muy resistente como tirador de trineo en invierno y perro de
carga en verano, logrando llevar mucho peso en larga distancia y
poco tiempo.
  </dd>
</dl>
```

No es necesario tener sólo una lista de definición por palabra o término, puede usar múltiples definiciones si lo desea.

Amplíe un poco el código ejemplo y coloque dos definiciones más, use su archivo básico *primer_html5.html* y guarde como *listas2.html*, dentro de `<body>...</body>` coloque un encabezado `<h1>` con el texto “Razas Caninas”, después, además del código de ejemplo de lista de definición anterior, añada las siguientes definiciones nuevas:

```
<dt>Basenji</dt>
<dd>La característica más destacable de la raza es que es un perro
que no ladra. Sólo emite una especie de aullido o canto. De
carácter independiente, inteligente y afectuoso.</dd>
<dt>Doberman</dt>
<dd>Esencialmente amistoso y pacífico. Muy dependiente de la
familia y cariñoso con los niños. Se le fomenta una bravura y un
temperamento medianos, además de un umbral de excitación
mediano.</dd>
```

Al terminar guarde y verifique el resultado en su navegador, la Fig. 3.9 muestra el resultado.

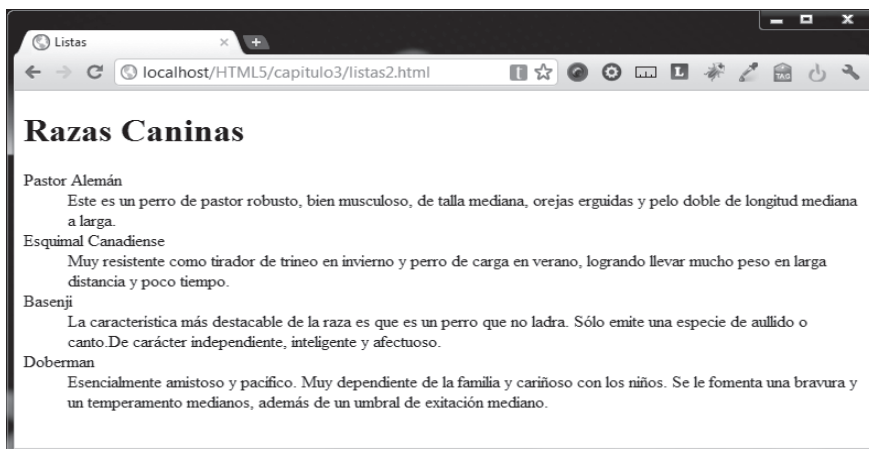


Fig. 3.9 Ejemplo de listas de definición.



En los materiales adicionales encontrará el archivo *listas2.html*.

3.4 Imágenes

Cuando ha aprendido cómo hacer la estructura básica de un documento HTML y cómo colocar texto en un documento de varias maneras, es momento de hacer sus páginas más interesantes. A diferencia de hace unos años, hoy es difícil encontrar una página Web que no contenga alguna imagen como ilustración, fondo o incluso como simple propaganda, es por eso que ahora aprenderá a insertar imágenes en sus páginas.

La etiqueta sin cierre que permite insertar una imagen es `` a la cual acompañan varios atributos que explicaré a continuación:

- **Atributo `src`**. Es quizá el más importante, ya que en este atributo se indica el origen de nuestra imagen. El origen de nuestra página puede ser una ruta local de archivo o bien una url, si no se indica una ruta válida (error muy común) la imagen no será desplegada.
- **Atributo `alt`**. Permite colocar un texto alternativo que hace alguna referencia a la imagen que se utiliza.
- **Atributo `border`**. Establece si la celda donde la imagen se despliega debe tener borde o no. El valor debe ser un número entero, el cual entre mayor sea mayor será el ancho del borde.
- **Atributo `height`**. Especifica el alto de una imagen.
- **Atributo `width`**. Especifica el ancho de una imagen.

Nota

En HTML5 existen dos atributos más para la etiqueta `` que escapan del enfoque de este libro, éstos son `ismap` y `usemap`, que tiene que ver con el uso de imágenes con zonas sensibles llamadas mapas.

Use el archivo *primer_html5.html*, guárdelo como *imagenes.html*; en la misma ubicación de nuestro archivo *imagenes.html*, coloque un archivo de imagen, dentro de `<body> ... </body>` escriba el siguiente código (sustituya el texto "suimagen" por el nombre completo de su imagen):

```

```

Guarde el archivo y compruebe el resultado.

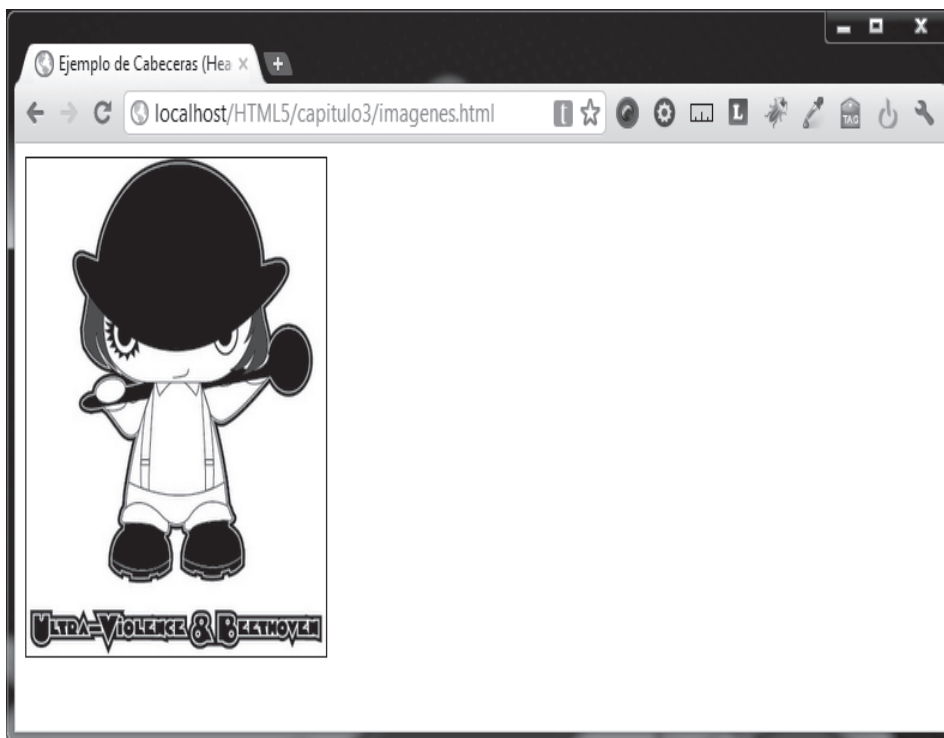


Fig. 3.10 Archivo que utiliza `` para mostrar una imagen.

Apoyo en la



En los materiales adicionales encontrará el archivo *imagenes.html*.

3.5 Tablas

Las tablas en HTML son a las que más se recurre para organizar el contenido de una página en su totalidad o simplemente para organizar un contenido parcial.

Una tabla en HTML viene marcada por la etiqueta con cierre `<table>`. Entre la etiqueta de inicio y cierre se define el contenido y la forma de la tabla, esto es, las celdas y las columnas que se requieren. Cabe mencionar que ésta es una de las etiquetas que más cambios ha sufrido en HTML5, ya que se han eliminado prácticamente todos los atributos que poseía, excepto por el de `border`.

- **Atributo `border`.** Éste es el único atributo rescatado en HTML5 con respecto a versiones anteriores de HTML, establece si la tabla tendrá borde o no. El valor debe ser un número entero, el cual, entre mayor sea, mayor será el ancho del borde exterior (no entre celdas).

El código para comenzar una tabla sería entonces: (se usará `border="1"` para ver la tabla más claramente):

```
<table border="1"></table>
```

Una vez creada la tabla proceda a crear en ella algunas filas, esto se logra utilizando la etiqueta con cierre `<td>`. De esta manera puede crear tantas filas como necesite. Aquí el código para tres filas:

```
<table border="1">  
<tr></tr>  
<tr></tr>  
<tr></tr>  
</table>
```

Las filas a su vez contienen columnas o celdas que van dentro de cada fila, tiene que escribirlas con la etiqueta `<td>` y su correspondiente cierre `</td>`. En HTML5 posee algunos atributos:

- **Atributo `colspan`.** Especifica a lo largo de cuántas columnas se extenderá una sola celda.
- **Atributo `headers`.** Relaciona una celda con la celda header correspondiente (etiqueta `<th>`). No hay efecto visible en el navegador, este atributo es para ayudar los lectores automatizados de texto (programas robot, programas para invidentes, etcétera).

- **Atributo** `rowspan`. Especifica a lo largo de cuántas filas se extenderá una sola celda.

También tiene la etiqueta con cierre `<th>` que utiliza los mismos atributos y funciona exactamente igual que `<td>`, la única diferencia es que estas celdas son usadas como *headers* y, sin que haga ninguna especificación, el texto que se coloque dentro estará en negritas.

Ejemplo

Código usando `<td>` y `<th>`.

```
<table border="1">
<tr>
  <th>Header 1</th>
  <th>Header 2</th>
  <th>Header 3</th>
</tr>
<tr>
  <td>Celda de columna 1</td>
  <td>Celda de columna 2</td>
  <td>Celda de columna 3</td>
</tr>
<tr>
  <td>Celda de columna 1</td>
  <td>Celda de columna 2</td>
  <td>Celda de columna 3</td>
</tr>
</table>
```

Use el archivo base *primer_html5.html* (recuerde que si desea crear un archivo base nuevo siéntase libre de hacerlo), guárdelo como *tablas.html*. Dentro de la estructura de `<body>` coloque el código anterior, guárdelo y vea el resultado.

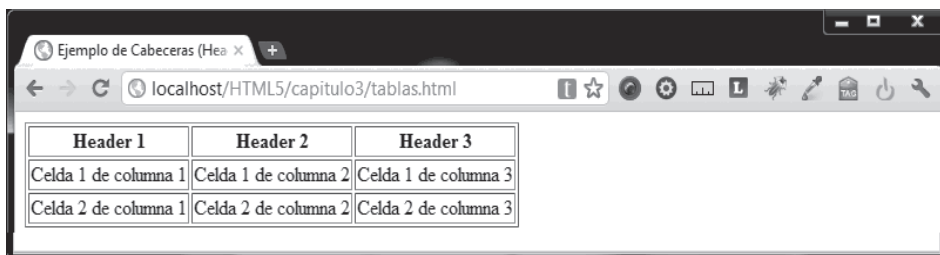


Fig. 3.11 Muestra del uso de tablas.

Apoyo en la



En los materiales adicionales encontrará el archivo *tablas.html*.

3.6 Hipervínculos (enlaces)

Los hipervínculos o enlaces (*hyperlinks*, *hypertext* o *links* en inglés), me atrevo a decir, son parte de la misma esencia de la Web como la conocemos hoy. Es fácil agregar un enlace a una página, sin embargo, son más de lo que parecen, ya que no sólo muestran texto, sino que al hacer clic sobre un enlace el navegador carga una nueva página o nos lleva a otra sección en el mismo documento.

La etiqueta utilizada para hacer un hipervínculo es la etiqueta con cierre `<a>`, todo lo que esté dentro de las etiquetas de inicio y cierre, ya sea texto o una imagen, estará considerado como un enlace y el navegador lo interpretará así. Los atributos con los que cuenta `<a>` son:

- **Atributo `src`.** Es quizá el más importante, ya que en este atributo se indica el origen de nuestra imagen. El origen de nuestra página puede ser una ruta local de archivo o bien una url, si no se indica una ruta válida (error muy común) la imagen no será desplegada.
- **Atributo `href`.** Describe el destino o ruta del enlace después de hacer clic. La ruta puede ser relativa o absoluta:
 - **Ruta relativa.** El valor de `href` puede ser un archivo común. Si el archivo está en el mismo directorio que la página, usted puede colocar únicamente el nombre del archivo. Esto es conocido como ruta relativa, ya que el navegador interpreta que el archivo enlazado está en la misma carpeta o en el mismo servidor.
 - **Ruta absoluta.** El valor de `href` puede ser una dirección Web completa. Esto es conocido como ruta absoluta. Si usted desea hacer referencia a páginas o archivos en el servidor de alguien más, debe usar este tipo de ruta.
- **Atributo `hreflang`.** Especifica el lenguaje del documento enlazado. Este atributo es solamente descriptivo, no tiene efecto visible.
- **Atributo `media`.** Especifica para qué tipo de medio o dispositivo está diseñado el documento enlazado. Este atributo es solamente descriptivo, no tiene efecto visible.
- **Atributo `rel`.** Establece la relación entre el documento actual y el documento enlazado.
- **Atributo `target`.** Indica dónde debería abrirse el enlace:
 - `_blank`. Abre el enlace en una nueva ventana o nueva pestaña.

- `_self`. Abre el documento enlazado en el mismo marco (frame) en el que se hizo clic.
- `_parent`. Muestra la nueva página en el `<frameset>` que contiene al marco donde se encuentra alojado el enlace.
- `_top`. Abre el documento enlazado en la misma ventana en donde se hizo clic.
- `frameName`. Abre el documento enlazado en un `<iframe>` con nombre específico.

Nota

`_parent`, `_top` y `frameName` se refieren normalmente a `iframes`, ya que los `frames` y los `framesets` han sido descontinuados en HTML5.

- **Atributo `type`**. Especifica el tipo MIME del documento enlazado (cuando un documento es procesado por un servidor Web para ser enviado a un navegador, el servidor clasifica e informa al navegador el tipo de documento que se enviará, esto es el tipo MIME).

Nota

Los atributos que más usaremos serán `href` y `target`, pero tenga presente todos para situaciones particulares.

El navegador Web tiende a intentar abrir cualquier documento que pueda leer, usted tiene la libertad de hacer enlaces a cualquier tipo de archivo. Tenga esto presente, ya que en ocasiones querrá sólo descargar y no visualizar el archivo enlazado dentro del navegador.

Ejemplo

Abra el archivo *primer_html5.html* y guárdelo con el nombre *enlaces.html*. Coloque en la sección `<body>` una cabecera `<h1>` con el texto "Enlaces", enseguida dos cabeceras `<h2>`, la primera con el texto "Enlace con ruta relativa" y el segundo con "Enlace con ruta absoluta". Después de la primera cabecera `<h2>` coloque un párrafo con el siguiente código:

```
<a href="ClockworkOrange.jpg">Mi Imagen</a>
```

Inmediatamente después de la segunda cabecera `<h2>` coloque:

```
<a href="http://www.alfaomega.com.mx">Mi editorial</a>
```


El código dentro de las etiquetas `<body>` y `</body>` se verá de esta manera:

```
<h1>Enlaces</h1>

<h2>Enlace con ruta relativa</h2>

<p>
Enlace a: <a href="ClockworkOrange.jpg">Mi Imagen</a>
</p>

<h2>Enlace con ruta absoluta</h2>

<p>
Enlace a: <a href="http://www.alfaomega.com.mx">Mi editorial</a>
</p>
```

Como siempre, salve el archivo y verifíquelo en su navegador, en la Fig. 3.12 se puede apreciar el resultado esperado del ejercicio de enlaces.

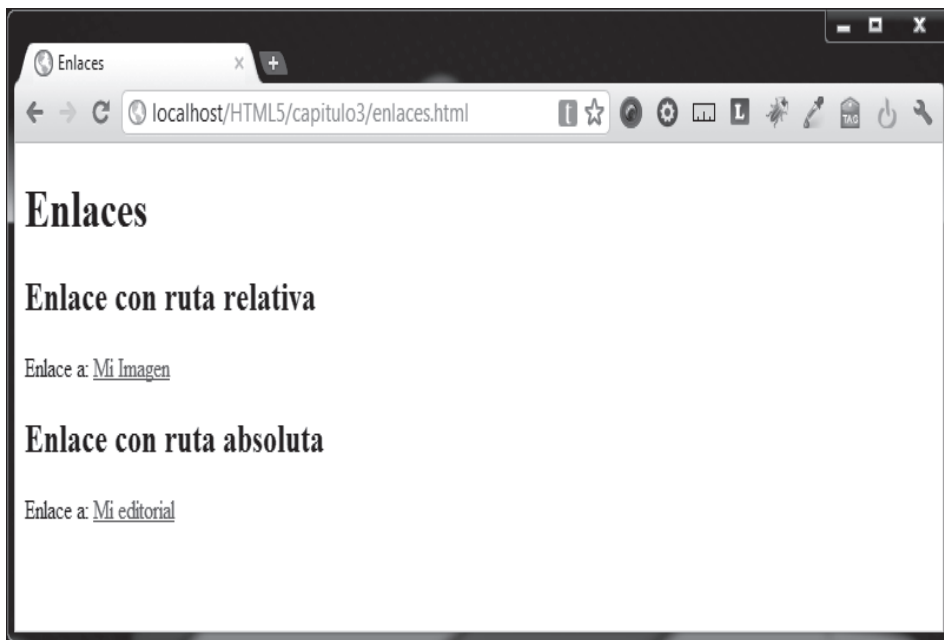


Fig. 3.12 Página con enlaces

Apoyo en la



En los materiales adicionales encontrará el archivo *enlaces.html*.

3.7 Formularios (controles clásicos)

Todo el mundo se ha visto en la necesidad de llenar alguna solicitud de empleo, o algún documento para completar algún trámite escolar o legal en papel, a estos documentos en papel se les llama formas o formularios. Desde siempre HTML ha ofrecido la manera de obtener datos de los usuarios que consumen nuestras páginas, a este recurso de la misma manera en que las versiones en papel, se les llama formularios. Usted no puede hacer nada con estos datos con sólo HTML, es aquí donde son necesarias las tecnologías de páginas de servidor (ASP, PHP, etc.) y JavaScript, sin embargo, es importante saber cómo trabaja cada elemento de los formularios. En este punto se abordará los controles HTML clásicos para los formularios (válidos en HTML5) y posteriormente los nuevos controles de formulario que nacen con HTML5.

A diferencia de la estructura de los puntos anteriores, en este punto se planteará primero el código completo de una página que hace uso de los elementos para formularios y posteriormente se hará el análisis pertinente de cada etiqueta.

Formulario clásico

localhost/HTML5/capitulo3/formularios.html

Formulario con controles clásicos

Botones

Campo de texto:

Area de texto:

Campo para contraseña:

Controles de selección

Lista de selección simple (tipo combo): Lista de selección simple:
Gato
Roedor

Lista de selección múltiple:
Gato
Roedor

Lista de botones radio: ☐ Perro ☐ Gato ☐ Roedor

Lista de cajas (checkbox): ☐ Chico ☐ Mediano ☐ Grande

Botones

Otros

Imagen: Aquí un otro control oculto (tipo hidden)

Fig. 3.13 Formulario clásico.

Ésta es la página más elaborada que se ha manejado hasta este momento. Observe cómo se ve en el navegador en la Fig. 3.13, a continuación se analizará parte por parte su código.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulario clásico</title>
    <meta charset="iso-8859-1" />
  </head>
  <body>
    <h1>Formulario con controles clásicos</h1>
    <form>
      <fieldset>
        <legend>Botones</legend>
        <p>
          <label>Campo de texto:</label>
          <input type="text" id="txt_ejemplo" value="Puede colocar
texto" />
        </p>
        <p>
          <label>Área de texto:</label>
          <textarea id="txtar_ejemplo" rows="4" cols="60">Puede
colocar texto</textarea>
        </p>
        <p>
          <label>Campo para contraseña:</label>
          <input type="password" id="psw_ejemplo" value="contraseña"
/>
        </p>
      </fieldset>
      <fieldset>
        <legend>Controles de selección</legend>
        <p>
          Lista de selección simple (tipo combo):
          <select name="mascotal">
```

```
<option>Perro</option>
<option>Gato</option>
<option>Roedor</option>
</select>
```

Lista de selección simple:

```
<select name="mascota2" size="2">
  <option>Perro</option>
  <option>Gato</option>
  <option>Roedor</option>
</select>
```

Lista de selección múltiple:

```
<select name="mascota3" size="2" multiple>
  <option>Perro</option>
  <option>Gato</option>
  <option>Roedor</option>
</select>
</p>
<p>
```

Lista de botones radio:

```
<input type="radio"
      id="perro"
      name="tipo_mascota"
      value="1" />
<label for="perro">Perro</label>
<input type="radio"
      id="gato"
      name="tipo_mascota"
      value="2" />
<label for="gato">Gato</label>
<input type="radio"
      id="roedor"
      name="tipo_mascota"
```

```
        value="3" />
        <label for="roedor">Roedor</label>
    </p>
    <p>
        <label>Lista de cajas (checkbox):</label>
        <input type="checkbox"
            id="chico"
            name="tam_chico"
            value="1" />
        <label for="chico">Chico</label>
        <input type="checkbox"
            id="mediano"
            name="tam_mediano"
            value="2" />
        <label for="mediano">Mediano</label>
        <input type="checkbox"
            id="grande"
            name="tam_grande"
            value="3" checked="checked" />
        <label for="roedor">Roedor</label>
    </p>
</fieldset>
<fieldset>
    <legend>Botones</legend>
    <p>
        <input type="submit"
            id="envía"
            value="Enviar información" />
        <input type="reset"
            id="borra"
            value="Borrar información" />
        <input type="button"
            id="boton"
            value="Botón" />
    </p>
</fieldset>
```


3.7.1 Estructura del formulario

La estructura general del formulario utiliza las siguientes etiquetas:

- **Etiqueta con cierre `<form>`.** Ésta es la etiqueta que contiene a todo el formulario. Todos los elementos que formen parte de un formulario deben ser colocadas dentro de las etiquetas `<form>...</form>`. Esta etiqueta puede utilizar varios atributos, pero los más utilizados son `action`, que indica adónde enviar los datos del formulario para que sean procesados (este proceso normalmente se auxilia de páginas activas de servidor) y el otro atributo comúnmente usado es `method` que especifica la manera en que se enviarán los datos en el protocolo HTTP.
- **Etiqueta con cierre `<fieldset>`.** Es usada para agrupar juntos una serie de elementos de un formulario. No es una etiqueta requerida en la estructura de formulario, pero es útil para organizar mejor los elementos visualmente. Por default dibuja una línea de contorno alrededor del grupo.
 - Etiqueta con cierre `<legend>`. Coloca un texto que sirve de leyenda para el `fieldset`.
 - Etiqueta con cierre `<label>`. Es una etiqueta se utiliza para colocar texto referente a un elemento de formulario, pero en realidad no hace nada notable sin su atributo `for`. Si se utiliza el atributo `for` (el cual debe ser igual al `id` de un elemento de formulario) entonces el `<label>` queda amarrado con dicho elemento, de manera que si el usuario hace clic sobre el texto dentro de `<label>` el foco lo obtendrá el elemento de formulario referido en el atributo `for`.

3.7.2 Campo de texto

Muchos de los controles de formulario están basados en el uso de la etiqueta `<input>`. Esta etiqueta usa el atributo `type` para señalar qué tipo de elemento debe colocar en la página. El tipo de control para obtener datos del usuario más utilizado es sin duda el campo de texto simple.

El proceso para agregar un control de etiqueta `<input>` es simple:

1. Se coloca la etiqueta vacía `<input>`.
2. Escriba el atributo `type` con su respectivo valor, en este caso `text`.
3. Añada el atributo `id`, el cual permite colocar un identificador al control, esto es de suma importancia, ya que posibilita distinguir este control de cualquier otro, por lo tanto no se debe repetir el valor de este atributo en ningún otro control. JavaScript y CSS se auxilian de este valor constantemente.
4. Opcionalmente puede agregar valor por default utilizando el atributo `value`. Cualquier texto colocado en este atributo será el valor por default para el elemento en el formulario.

Después de seguir estos pasos el campo de texto simple aparecerá en el navegador. El usuario al hacer clic en el campo colocará el cursor y podrá escribir dentro de él, tenga en cuenta que el campo de texto simple es para información corta y concreta. En nuestro formulario el código del control es el siguiente:

```
<input type="text"
      id="txt_ejemplo"
      value="Puede colocar texto" />
```

3.7.3 Área de texto

Hay momento en donde se requiere de varias líneas de texto para escribir un dato largo, dado que el campo de texto simple es para información breve, el control área de texto se vuelve muy conveniente. Éste no es un control de tipo `<input>` por lo que la sintaxis es algo diferente, ya que se usa la etiqueta con cierre `<textarea>`.

Para hacer un área de texto:

1. Escriba las etiquetas `<textarea>` y su cierre `</textarea>`.
2. Especifique el número de líneas que desea que el área de texto pueda contener. Si agrega más líneas el área de texto será más grande.
3. Indique el número de columnas que convenga. La cantidad de columnas es medida en caracteres.

Este elemento en el formulario se escribió así:

```
<textarea id="txtar_ejemplo" rows="4" cols="60">
Puede colocar texto
</textarea>
```

3.7.4 Campo de contraseña (password)

En muchos formularios es común requerir datos personales que no es conveniente que estén a la vista de cualquiera, como por ejemplo, contraseña entre otros. Para ello se debe crear un campo que no muestre los datos que se escriben en él. Para lograr esto se utiliza la etiqueta `<input>` con el atributo `type` con valor `password`, tal y como está en el código de nuestro formulario:

```
<input type="password"
      id="psw_ejemplo"
      value="contraseña" />
```


Nota

Usar este tipo de control oculta visualmente su valor, pero por sí mismo no encripta, ni oculta su valor al enviar la información. Por lo que una persona hábil puede obtener esta información si usted no se auxilia de técnicas y tecnologías de seguridad más avanzadas.

3.7.5 Lista de selección simple (combobox)

Las listas de selección simple como combobox son muy comunes y de gran ayuda para los formularios en los que se requiera saber un dato preciso y excluyente de otros, de manera que se limite a opciones planteadas previamente por nosotros. Este tipo de elemento implícitamente nos ofrece la ventaja de poder evitar errores de escritura por parte del usuario y predecir exactamente los posibles valores que podemos obtener.

Las listas de selección u opciones se obtienen gracias a la etiqueta `<select>` y a su cierre, las opciones que contienen cada lista de selección simple se escriben utilizando la etiqueta con cierre `<option>`.

Los pasos para crear este control en una página son:

1. Colocar las etiquetas contenedoras de las lista `<select>` y `</select>`.
2. Asignar un identificador único a la lista con el atributo `id`.
3. Escribir dentro del ámbito de `<select>` una etiqueta `<option>` con su cierre para cada opción de su lista.
4. Dar a cada `<option>` un valor, que será el valor enviado por el usuario después de seleccionar y enviar la información. Para esto escriba el atributo `value` y asigne el valor deseado.
5. Colocar el texto que el usuario verá por cada opción. Sólo coloque el texto entre la etiqueta de inicio `<option>` y la de cierre `</option>`.

El código de este control se observa así:

```
<select name="mascota1">
<option>Perro</option>
<option>Gato</option>
<option>Roedor</option>
</select>
```

3.7.6 Lista de selección múltiple (listbox)

Es posible que en lugar de tener una lista de selección simple donde sólo se puede ver una opción a la vez, si se desea ver más de una opción a la vez, se seguirán los mismo pasos que se utilizan para la lista de selección múltiple, únicamente se agregará el atributo `size` a la etiqueta `<select>` y se le asignará un valor numérico correspondiente a la cantidad de opciones que desea ver al mismo tiempo. Visualmente lo que se obtendrá es que en lugar de un menú desplegable aparecerá simplemente una caja con las opciones y un scroll lateral si la cantidad de opciones lo amerita. En el formulario está escrito de la siguiente manera:

```
<select name="mascota2" size="2">
<option>Perro</option>
<option>Gato</option>
<option>Roedor</option>
</select>
```

3.7.7 Lista de selección múltiple (listbox multiple selection)

Si desea seleccionar más de una opción al mismo tiempo, sólo agregue otros pasos a las listas de selección anteriores [3.7.5 Lista de selección simple (combobox) y 3.7.6 Lista de selección múltiple (listbox)]. Únicamente se requiere colocar el atributo `multiple` a la etiqueta `<select>` sin ningún otro valor.

El código del formulario es:

```
<select name="mascota3" size="2" multiple>
<option>Perro</option>
<option>Gato</option>
<option>Roedor</option>
</select>
```

3.7.8 Lista de botones radio (radio buttons)

Otro tipo de lista para la etiqueta `<input>` son las listas de botones radio. Esto es, cómo tener respuestas de selección múltiple, en donde sólo una respuesta es la correcta, por lo tanto, este tipo de control siempre va en grupos de 2 o más miembros en la lista y por lo menos un elemento siempre debe estar seleccionado.

Los pasos generales para crear este tipo de lista son:

1. Inicie escribiendo la etiqueta `<input>`.

2. Asigne al atributo `type` el valor de `radio`.
3. Repita los pasos 1 y 2 tantas veces como opciones en la lista de botones desee y dé un valor al atributo `id` único.
4. Coloque a todos los botones (etiquetas `<input>`) que haya creado, el mismo valor para el atributo `name` (esto los hará botones miembros de la misma lista).
5. Especifique el valor a cada miembro de la lista usando el atributo `value` igual al valor deseado (`value="suvalor"`).
6. Ordene su código de manera que visualmente sea evidente que todos los botones son miembros de la misma lista.
7. Seleccione el botón `radio`, el cual estará seleccionado por default, para lograr esto agregue el atributo `checked` con el valor `checked` (`checked="checked"`).
8. Asigne una `<label>` asociada con el atributo `for` a la opción correspondiente con un texto descriptivo en cada botón para ayudar al usuario a distinguir entre sus opciones.

El código en su formulario ejemplo es:

```
<input type="radio"
      id="perro"
      name="tipo_mascota"
      value="1" />
<label for="perro">Perro</label>
<input type="radio"
      id="gato"
      name="tipo_mascota"
      value="2" />
<label for="gato">Gato</label>
<input type="radio"
      id="roedor"
      name="tipo_mascota"
      value="3" checked="checked" />
<label for="roedor">Roedor</label>
```

Nota

Recuerde que la etiqueta `<label>` comparte el foco con el `<input>` asociado para facilitar el clic al usuario.

3.7.9 Lista de cajas (checkboxes)

Se encontrará casos en donde cierta información requiere una respuesta positiva o falsa. El control apropiado para enfrentar esta situación son las cajas de selección, en donde el usuario marcará o demarcará con un clic en la caja correspondiente a la opción deseada.

Las cajas de selección a menudo forman grupos para aparecer en listas, como es nuestro caso, pero actúan independientemente una de otra, de manera que se puede dar el caso de una caja solitaria. Éstos son los pasos para construir nuestra lista de cajas:

Inicie escribiendo la etiqueta `<input>`.

1. Dé al atributo `type` el valor de "checkbox".
2. Repita los pasos 1 y 2 tantas veces como opciones en la lista de botones desee y dé un valor al atributo "id" único.
3. Especifique el valor a cada miembro de la lista usando el atributo `value` igual al valor deseado (`value="suvalor"`).
4. Asigne una `<label>` asociada con el atributo `for` a la opción correspondiente con un texto descriptivo en cada botón para ayudar al usuario a distinguir entre sus opciones.

El código en el formulario es:

```
<input type="checkbox"
      id="chico"
      name="tam_chico"
      value="1" />
<label for="chico">Chico</label>
<input type="checkbox"
      id="mediano"
      name="tam_mediano"
      value="2" />
<label for="mediano">Mediano</label>
<input type="checkbox"
      id="grande"
      name="tam_grande"
      value="3" checked="checked" />
<label for="roedor">Roedor</label>
```

3.7.10 Botón para enviar información

El botón para enviar información o submit, normalmente se utiliza para enviar la información coleccionada por todos los campos del formulario al lado del servidor. Alguna tecnología de servidor como ASP o PHP entre otras procesa esta información. Este control es otro valor del atributo `type` igualado a `submit` de la versátil etiqueta `<input>`, el texto para el atributo `“value”` funciona como leyenda dentro del botón. En el formulario el código utilizado es:

```
<input type="submit"
      id="envía"
      value="Enviar información" />
```

3.7.11 Botón para borrar información del formulario

Éste es un botón muy sencillo, cuando el usuario hace clic en él, todos los campos del formulario borran su valor actual y regresan a su estado original. Trabaja con la etiqueta `<input>` con el el valor de `type` igual a `reset`, el texto para el atributo `value` funciona como leyenda dentro del botón, el código es:

```
<input type="reset"
      id="borra"
      value="Borrar información" />
```

3.7.12 Botón simple

El botón simple tiene el mismo aspecto que los dos anteriores y tiene la función de trabajar del lado cliente, a diferencia del botón para enviar información que envía información al servidor. Usualmente este botón trabaja con JavaScript para disparar alguna acción, el texto para el atributo `value` funciona como leyenda dentro del botón. En el formulario de ejemplo:

```
<input type="button"
      id="botón"
      value="Botón" />
```

El código anterior muestra la forma tradicional de hacer un botón simple, pero existe una forma alternativa que tiene más sentido, ya que el botón simple no captura información como sugeriría el uso de `<input>`, sino que simplemente ejecuta una acción que el usuario convoca. Esta nueva forma es:

```
<button type="button">
  Botón
</button>
```

Por el momento no hay restricción de cuál manera debe usar, aunque cuando aprenda a usar CSS es muy probable que prefiera la segunda opción.

3.7.13 Imagen (botón)

Este tipo de botón trabaja de la misma manera que el botón simple, sin embargo, la diferencia radica en que en lugar de trabajar con la apariencia por default del botón simple, utiliza en su lugar una imagen indicada por usted. El atributo `type` en `<submit>` contiene el valor `image`, se utiliza el atributo `src` que indicará la ruta donde se encuentra la imagen que desee usar y el atributo `alt` para un texto alternativo. Así es como se usa:

```
<input type="image"
      id="imagen"
      src="img_boton.jpg"
      alt="imagen como botón" />
```

3.7.14 Notas adicionales sobre botones

La versátil etiqueta `<input>` permite crear botones como se ha hecho en el formulario de ejemplo y como se ha explicado, esta forma es la manera tradicional de hacer botones en HTML y es válida en HTML5, pero también existe una nueva manera de hacer el mismo tipo de botones. Si lo medita un poco, los botones no son un campo donde se coloque información, es decir, no son un dato de entrada como lo serían el resto de los tipos `<input>`, es por eso que existe la etiqueta con cierre `<button>`, la manera de usarla es muy intuitiva, para sustituir nuestro botón simple escriba el código de la siguiente manera:

```
<button type="button" id="boton">
  Botón
</button >
```

3.7.15 Campo oculto

El campo oculto es una etiqueta `<input>` con el valor de `type` igual a `hidden`. Gracias a este atributo se envía al programa de gestión de datos, además de los datos enviados por el usuario, datos predefinidos por nosotros mismos, invisibles para el usuario. Estos datos pueden ser útiles para ayudar al programa servidor en la gestión de los datos enviados por el formulario.

```
<input type="hidden"
      id="oculto"
      value="Borrar información" />
```

3.8 Validar páginas HTML5

Seguir todo el esquema que se ha visto en estos primeros capítulos le dará la posibilidad de hacer páginas con buenas bases. Sin embargo, debo ser honesto con usted, en el momento en el que decide aprender a programar utilizando cualquier tecnología, sin bien ha abierto un mundo de posibilidades, también ha abierto la puerta a una gran cantidad de errores potenciales dentro de su código. ¡No se aflija!, esto es parte del reto que le presenta la programación, y para ayudarle en el caso de páginas hay herramientas que le ayudarán a minimizar estos errores.

La W3C, consciente de esta situación, ha creado una herramienta útil para validar nuestras páginas y ayudarnos a detectar errores. La herramienta es llamada *W3 validator* y está disponible en:

<http://validator.w3.org>

Esta herramienta es de gran ayuda, pero no puede encontrar todos los errores y sólo funciona si cuenta con conexión a Internet. Adicionalmente los mensajes de error no siempre son suficientemente descriptivos o exactos. A pesar de esto la herramienta es útil y por fortuna no es la única, a medida que progrese en sus habilidades como programador, usted mismo encontrará más alternativas para ayudarle en la creación de sus páginas, utilizando, por ejemplo, editores más avanzados de código entre otras cosas.

Actividades para el lector

Realice un documento que utilice un formulario que incluya por lo menos 4 de los elementos de formularios aprendidos y coloque una imagen y un hipervínculo en cualquier parte del mismo documento.

RESUMEN

En este capítulo se ampliaron los conocimientos acerca de HTML conociendo:

- La diferencia entre HTML y HTML5.
- El mejor manejo de la estructura visual de contenido HTML.
- La creación básica de formularios.
- La manera de verificar la correcta construcción de una página con el servicio de verificación de etiquetas de la W3C.

Autoevaluación

1. ¿Qué es XHTML?
2. ¿Qué son las listas HTML?
3. ¿Qué etiquetas HTML componen una tabla?
4. ¿En qué casos puede ser útil un formulario HTML?
5. ¿Qué herramienta proporciona la W3C para verificar las etiquetas de un documento HTML5?

EVIDENCIA

☐

Realizó un formulario utilizando 4 controles de formulario, una imagen e hipervínculo.

☐

En el formulario solicitado incluyó todos los elementos solicitados y algunos más de acuerdo a sus necesidades o creatividad.

REFERENCIAS

Bibliografía

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.

Wempen, Faithe (2011). HTML5: Step by Step, 1a. ed., O'Reilly, Canadá.



Páginas Web recomendadas

<http://en.wikipedia.org/wiki/HTML>

<http://dev.w3.org/html5/spec/Overview.html>

<http://diveintohtml5.org/>

<http://www.w3schools.com/html5/default.asp>

Respuestas sugeridas a las preguntas de autoevaluación

1. XHTML es una versión que utiliza una sintaxis estricta basada en XML, fue creada pensando originalmente en sustituir a HTML.
2. Son etiquetas HTML que permiten hacer un listado de elementos, como podrían ser los elementos de un temario o un índice.
3. Las etiquetas involucradas para hacer una tabla son `<table>`, `<tr>`, `<td>` y `<th>`, todas con su respectivo cierre.
4. Cuando desea que el usuario proporcione alguna información.
5. En la dirección <http://validator.w3.org/> la W3C proporciona su validador de etiquetado HTML5.

Los nuevos elementos HTML5

4

Reflexione y responda las preguntas siguientes:

¿Qué es la semántica en un documento HTML?

¿Qué etiquetas apoyan la filosofía semántica de HTML5?

¿Cómo insertar imágenes vectoriales con elemento `<canvas>` de HTML5?

¿Cómo insertar audio y video con elementos nativos de HTML5?

Contenido

Los nuevos elementos HTML5

4.1 Nueva semántica

4.2 Elementos estructurales

4.3 Elementos estructurales complementarios

4.4 Elementos semánticos en línea

4.5 Media

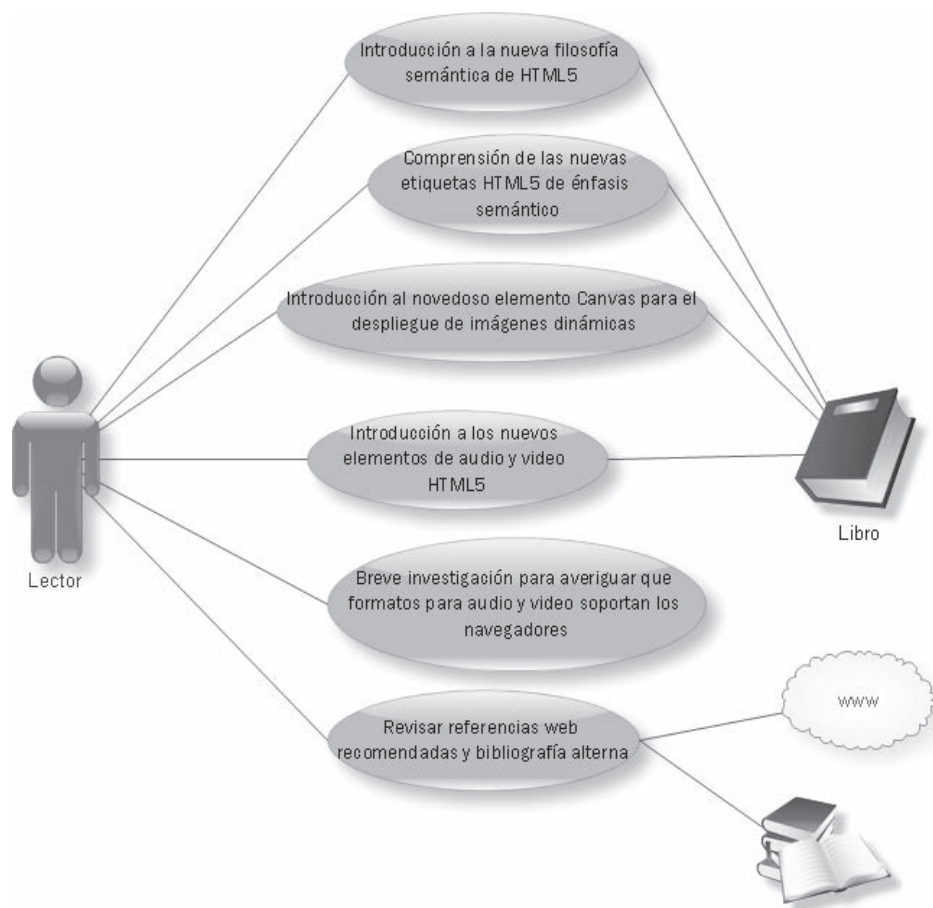
Expectativa

Conocer la importancia que brinda HTML5 a la semántica de los documentos HTML.

Conocer los nuevos elementos para insertar gráficos, sonidos y videos en páginas HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Utilizar los nuevos elementos semánticos de HTML5.
- Comenzar a comprender el potencial del elemento <canvas>.
- Enriquecer sus páginas con audio y video con los nuevos elementos HTML5.



4.1 Nueva semántica

Existen buenas razones para aspirar a páginas más semánticas, por ejemplo, muchos rastreadores (programas que buscan, ordenan e indexan información de la Web) o lectores de pantalla para débiles visuales utilizan las cabeceras (*headers*) para navegar documentos, es por esto que más que el formato visual importa la semántica de nuestro contenido, en otras palabras, a ese tipo de programas no les importa tanto el tamaño o color de nuestro título principal ni del subtítulo, pero sí les importa saber cuál es cuál para interpretar un documento correctamente.

La semántica da sentido o significado a algo, en HTML es usar etiquetas que le den un mejor significado a nuestras páginas. La mayoría de las nuevas etiquetas semánticas no dan una apariencia visible particular, en lugar de esto, describen el significado de una parte del código en el contexto de una página Web.

Anteriormente los desarrolladores de páginas usaban exhaustivamente la etiqueta `<div>` y/o el atributo `class` para tratar de separar y organizar en secciones el contenido de una página. Dado que el navegador cuando usa versiones anteriores de HTML “no entiende” lo que se hizo con esta práctica, aunque poco semántico es (o era) válido hacerlo. Pero ahora se cuenta con nuevos elementos propios de HTML5 para hacer más descriptivas nuestras páginas y permitir con esto que el contenido en ellas sea identificado, clasificado, interpretado y manejado de forma más precisa por navegadores, lectores, indexadores, etcétera.

Nota

Recuerde que al momento de escribir este libro HTML5 aún no es oficialmente un estándar y que el navegador utilizado para los ejemplos es Google Chrome, por lo tanto, puede darse el caso de que otros navegadores no soporten aún alguna etiqueta.

4.2 Elementos estructurales

Como se mencionó, durante muchos años se han utilizado `<div>` y `class` para definir (casi por regla general) un *header*, un *footer*, *navs* y otras entidades muy comunes en el desarrollo de cualquier sitio Web. Finalmente llegó el momento de incorporar formalmente a todas esas entidades como parte del lenguaje y actualmente son parte de la nueva especificación de HTML5.

4.2.1 article

La etiqueta con cierre `<article>` es como una página pequeña dentro de una página. Debe usarse para definir contenido autónomo e independiente, proveniente de una fuente externa de información y predecible de ser reutilizada de modo aislado.

```

<article>
<header>
  <h1>Bienvenido(a) a la Tienda de Mascotas!</h1>
</header>
  <p>Nuestra misión es facilitarle el camino para hacer de una de
nuestras mascotas un gran compañero para usted y toda su familia,
disfrute la visita a nuestro sitio y esperamos servirle pronto en
nuestra tienda
  </p>
  <p>Acuda a nuestra tienda para conocer a nuestras mascotas y a
nuestro extenso catálogo de juguetes y accesorios disponibles para
su diversión y cuidado<sup>1</sup></p>
</article>

```

4.2.2 aside

Cuando se desea colocar información en un fragmento de la página que está relacionado con el contenido principal, pero debe estar separado de él, se utiliza la etiqueta con cierre `<aside>`. Comúnmente a `<aside>` se le ubica como una barra lateral. Tenga en cuenta que si desea colocar una barra lateral porque el diseño visual lo indica, no significa necesariamente que deba utilizar `<aside>`. Unas preguntas simples que puede utilizar para determinar si es conveniente usar o no son: ¿Si elimino el contenido dentro de `<aside>`, el contenido principal se ve reducido?, ¿el contenido que pienso colocar en `<aside>` tiene que ver con información respecto a la estructura o autoría de la página? Si la respuesta es negativa en ambas preguntas, entonces puede usar el elemento `<aside>`.

Déjeme advertirle que varios navegadores soportan la etiqueta, pero no se han puesto de acuerdo en cómo presentarla.

```

<aside>
<h3>Visítenos!</h3>
<p><ins>Nuestra Dirección:</ins><br/>
  Calle: San Ciprián No. 45<br/>
  Colonia: Centro<br/>
  México D.F.<br/>
  C.P. 07640<br/>
</p>
<p><sup>1</sup>Venta de comida, productos y accesorios para perros,
  gatos, pájaros, roedores, reptiles y peces.
</p>
</aside>

```

4.2.3 footer

La etiqueta `<footer>` se utiliza para representar el pie de página (en contexto de una página Web). Se posiciona por supuesto en la parte inferior de la página, usualmente se utiliza este elemento para colocar información de derechos de autor, información de contacto o créditos.

Para usted que tiene experiencia en el desarrollo de páginas Web, tendrá sentido enterarse que `<footer>` sustituye al habitual `<div id="footer"></div>`, tan utilizado en versiones anteriores de HTML.

```
<footer>
  <p>&copy;&nbsp;2011 todo los derechos reservado</p>
</footer>
```

4.2.4 header

El elemento con cierre `<header>` se utiliza como tope o cabecera visible de una página. No confunda `<header>` con la etiqueta `<head>` que contiene el título y los metadatos de nuestras páginas. En `<header>` usualmente se tienen etiquetas de encabezado `<h1>` a `<h6>` o alguna etiqueta ``.

La etiqueta `<header>` no está limitada a cabecera de una página, también puede usarse dentro de una etiqueta `<article>` y `<section>` (este elemento se abordará en el punto 4.2.6) para indicar cabeceras dentro de éstas, si usted coloca un `<h1>` en el encabezado principal sólo de ese artículo o sección. Este elemento está pensado para sustituir al uso común de `<div id="header"></div>`.

```
<header>
<h1>Bienvenido(a) a la Tienda de Mascotas!</h1>
</header>
```

4.2.5 nav

La etiqueta `<nav>` junto con su cierre es utilizada para contener información sobre la navegación del sitio Web que se construyó. Este elemento usualmente utiliza enlaces con la etiqueta `<a>`, pero éstos deben de ser enlaces sólo para la navegación principal del sitio y no para enlaces externos. El elemento `<nav>` ha sido creado buscando sustituir a `<div id="nav"></div>`.

```
<nav>
<h3>Menú Principal</h3>
  <a href="#">Inicio</a>
  <a href="#">Gatos</a>
  <a href="#">Perros</a>
7
</nav>
```

4.2.6 section

El nuevo elemento con cierre `<section>` intenta albergar contenido específico sobre algún tema, es muy similar a usar la etiqueta `<div>`, pero `<div>` no tiene un significado semántico y no informa sobre el tipo de contenido alojado en la etiqueta. El elemento `<section>` se usa de forma explícita para agrupar contenido relacionado.

Considere la pregunta ¿está todo el contenido incluido relacionado entre sí?, si la respuesta es positiva entonces puede utilizar `<section>`.

```
<section>

  <p>Acuda a nuestra tienda para conocer a nuestras mascotas y a
  nuestro extenso catálogo de juguetes y accesorios disponibles para
  su diversión y cuidado<sup>1</sup></p>

</p>
</section>
```

Nota

Hasta el momento el soporte para `<section>` por parte de los navegadores es reducido. Los elementos `<div>` y `<article>` pueden ser buenos sustitutos.

Apoyo en la



La página *principal.html* (la tienda de mascotas) ha sido modificada para implementar cada uno de los ejemplos expuestos hasta ahora, puede descargarla para analizarla con detalle.

4.3 Elementos estructurales complementarios

Hasta ahora se ha visto los elementos que brindan el esquema general a nuestras páginas, pero existen aún varios elementos más que ayudan a dar una mayor profundidad semántica a ellas. A continuación se expone algunas de las más utilizadas.

4.3.1 address

La etiqueta `<address>` contiene la información de contacto del autor de una página, sección o artículo. Si `<address>` está dentro de un elemento `<article>` o `<section>` se está indicando que el autor referido lo es sólo del contenido de ese artículo o sección. Otras etiquetas pueden ser colocadas dentro de este elemento, como pueden ser enlaces, por ejemplo.

El elemento `<address>` ya existía en la versión HTML4, pero sólo podía hacer referencia a una página entera, mientras que ahora, como ya se mencionó, se puede referir a secciones de contenido más pequeñas.

```
<address>
    Escrito por Su Nombre<br/>
    112 Mercer St.<br/>
    Princeton, NJ<br/>
</address>
```

4.3.2 hgroup

La etiqueta con cierre llamada `<hgroup>` permite agrupar en un bloque un título y un subtítulo. Dentro de un elemento `<hgroup>` tan sólo se puede introducir etiquetas de encabezado (`<h1>`-`<h6>`) de forma que no permite ningún otro tipo de etiqueta en su interior. Cuando utiliza `<hgroup>` para delimitar encabezados, se está indicando cuáles están relacionados, pero sobre todo se señala cuál es el encabezado principal del grupo el cual será el único elemento tomado en cuenta para el esquema total del documento, es decir, si usted continúa con más encabezados (`<h2>`-`<h6>`) dentro de un `<hgroup>` estos últimos serán tomados en cuenta para el grupo, pero no para el resto del documento.

```
<hgroup>
<h1>Sección para perros</h1>
    <h2>Encuentra lo necesario para tus amigos caninos</h2>
</hgroup>
```

4.3.3 menu

La etiqueta `<menu>` se asocia con la etiqueta `<command>` utilizada para añadir varios tipos de menús. El atributo `“type”` es el que nos permite seleccionar el tipo de menú, con los siguientes valores posibles:

- `list`. Muestra los comandos de elemento del menú como si fueran una lista.
- `context`. Se muestra como menú contextual, es decir, emerge cuando se hace clic con el botón derecho del ratón.
- `toolbar`. Los comandos se verán como una lista de herramientas.


```
<menu>

  <command label="opción 1" onclick="alert('opción 1')">Clic
Aquí!</command>

  <command label="opción 2" onclick="alert('opción 2')">Clic
Aquí!</command>

  <command label="opción 3" onclick="alert('opción 3')">Clic
Aquí!</command>

</menu>
```

Ahora se tiene un inconveniente, la especificación HTML5 dice que el elemento `<menu>` está diseñado para trabajar con `<command>`, pero muy recientemente sólo los navegadores Internet Explorer y Safari soportan `<command>`, más adelante en este capítulo se mostrará cómo solucionar este problema.

4.4 Elementos semánticos en línea

Esta categoría no existe de manera oficial en la especificación de HTML5, sin embargo, se utilizaba en versiones anteriores para describir a aquellas etiquetas breves que no suelen anidar a muchas otras etiquetas o a ninguna en lo absoluto, o que contienen un contenido sumamente breve, y que no afectan a nivel de estructura de página. Éste es exactamente el sentido que le daré a este punto para ayudarlo a entrar rápidamente a HTML5.

4.4.1 command

La etiqueta `<command>` con su respectivo cierre trabaja con elemento `<menu>`, para añadir un elemento al menú, pero también puede trabajar en cualquier punto de una página para funcionar como un atajo de teclado. Este elemento cuenta con los atributos siguientes:

- **Atributo** `icon`. Indica la ruta de una imagen que representará al comando con un ícono.
- **Atributo** `checked`. Si el comando es de tipo caja de selección (*checkbox*) o botón radio (*radio button*) indica si el comando está marcado o no.
- **Atributo** `disabled`. Establece si el comando está activado o no.
- **Atributo** `label`. Le da un texto al comando, el cual será mostrado en la página.
- **Atributo** `title`. Despliega una leyenda con un texto como descripción del comando.

- **Atributo type.** Indica el comportamiento del comando, éste puede ser el correspondiente a una caja de selección (*checkbox*), comando simple (*command*) o un botón radio.

Los atributos globales para el manejo de eventos funcionan con `<command>`, el más utilizado es el atributo `onclick`, el cual especifica una línea simple de código JavaScript para ejecutar cuando se hace clic.

En al punto 4.3.3 menu, se señaló que hasta el momento esta etiqueta sólo es soportada por Internet Explorer y Safari; una técnica para solucionar este inconveniente es usar una lista en lugar de `<command>` como se muestra:

```
<menu>
  <li onclick="alert('opción 1')">opción 1</li>
  <li onclick="alert('opción 2')">opción 2</li>
  <li onclick="alert('opción 3')">opción 3</li>
</menu>
```

4.4.2 details y summary

La etiqueta con cierre `<details>` funciona como un espacio desplegable y replegable. Trabaja junto con la etiqueta `<summary>` y su respectivo cierre donde ésta funciona como texto en forma de título sensible, el cual al hacer clic en él, despliega u oculta contenido colocado mediante otra etiqueta.

```
<details>
  <summary>Mi contenido</summary>
  <p>El contenido replegable</p>
</details>
```

4.4.3 dfn

Se usa la etiqueta con cierre `<dfn>` si en el contexto del contenido desea definir un término. Simplemente hay que rodear al término, no a su significado, con las etiquetas `<dfn>` y `</dfn>`.

```
<p><dfn>HTML:</dfn>
  Lenguaje de marcado para la elaboración de páginas Web.
</p>
```

4.4.4 figure y figcaption

Utilizamos la etiqueta `<figure>` con su cierre si decidimos que es semánticamente importante enfatizar cualquier tipo de imagen, no se afecta de ninguna manera la presentación visual de ésta. Opcionalmente puede utilizar un título descriptivo para la imagen en cuestión con la etiqueta `<figcaption>` y su cierre como se muestra a continuación.

```
<figure>
  <figcaption>Dibujo de una pelota</figcaption>
  
</figure>
```

4.4.5 wbr

La etiqueta `<wbr>` y su cierre indican dónde podría estar bien un salto de línea en una palabra demasiado larga. Esto aplica cuando el navegador ajusta el texto al espacio disponible en el navegador, si encuentra al elemento `<wbr>` lo interpreta como una posible oportunidad para hacer el salto de línea.

```
<p>
  Para explotar el poder de HTML5 debe aprender Java<wbr>Script y
  CSS.
</p>
```

Apoyo en la



El autor ha creado un archivo llamado *perros.html*, el cual implementa todos los elementos de los puntos 4.3 y 4.4 y varios del 4.2. Está disponible para su descarga en los materiales adicionales.

4.5 Media

Ésta es la característica más esperada de HTML5 y probablemente la más divertida desde el punto de vista de desarrollo y aún más para el usuario. Ahora HTML5 soporta la inserción de audio y video sin necesidad de ningún componente externo e incluye también soporte para gráficas vectoriales a través del elemento `<canvas>` y SVG.

4.5.1 audio

El elemento `<audio>` y su cierre nos permite tener audio embebido en nuestras páginas. Cualquier texto entre las etiquetas `<audio>` y `</audio>` será desplegado en cualquier navegador que no soporte el elemento de audio, opcionalmente también puede colocar en lugar de texto un enlace para descargar el

archivo de audio o embeber un reproductor Flash. El elemento `<audio>` cuenta con los atributos siguientes:

- **Atributo** `autoplay`. Indica si el archivo de audio debería ser automáticamente reproducido o no.
- **Atributo** `controls`. Especifica si los controles de reproducción deberían ser o no desplegados.
- **Atributo** `loop`. Establece si la reproducción del audio debería ser reiniciada cada vez que llega al final.
- **Atributo** `preload`. Especifica si el audio debería ser cargado cuando la página comienza a cargar.
- **Atributo** `src`. Indica la dirección URL del archivo de audio a reproducir. A pesar de la existencia de este atributo es mejor utilizarlo en la etiqueta vacía `<source>`, ya que ésta nos permite colocar varias alternativas del origen del audio.

Aunque `<audio>` ya es soportado por los principales navegadores, los fabricantes aún no están completamente de acuerdo con el formato de audio que debería ser utilizado, y por lo tanto soportan diferentes formatos que se debaten principalmente entre el estándar libre *Ogg* y *Mp3*. Por fortuna, esto no es un gran problema si usted simplemente coloca una versión de su archivo para cada formato utilizando `<source>`, el navegador reproducirá el primer formato que le sea compatible.

```
<audio controls="controls">
<source src="ladridos.ogg" />
<source src="ladridos.mp3" />
Su navegador no soporta el elemento audio
</audio>
```

4.5.2 canvas

La etiqueta con cierre `<canvas>` (lienzo) es quizá el elemento que puede arrojar resultados más llamativos para el usuario. Cuando utiliza `<canvas>` el navegador reserva una parte de la ventana para el uso de gráficos, los cuales serán dibujados justo ahí. El elemento en sí es fácil de implementar en términos de HTML, pero su verdadera manipulación se hace totalmente por medio del uso de JavaScript.

```
<canvas id="miCanvas" width="350" height="300">
    El elemento canvas requiere soporte HTML5
</canvas>
```

Cualquier cosa que escriba entre `<canvas>` y `</canvas>` solamente será interpretado por navegadores que no soportan aún la nueva etiqueta. Ahora que se adicionó un `<canvas>` también se incluirá un botón para llamar a una función JavaScript llamada `dibuja()` cuando se hace clic en él.

```
<button type="button" onclick="dibuja();">
    Dibuja!
</button>
```

La función JavaScript `dibuja()` es la que en realidad hará algo interesante. Primero debe referenciar el elemento `<canvas>` para adquirir su contexto (la interfaz de programación de aplicación o API).

```
function dibuja(){
    var micanvas = document.getElementById('miCanvas');
    var context = micanvas.getContext('2d');
    context.strokeStyle = '#8eee14';
    context.fillStyle = '#ff8f43';
    context.strokeRect (20, 30, 100, 50);
    context.strokeRect (410, 30, 100, 50);

    for (i=1; i<=200; i+=2){
        context.strokeStyle = '#8eee14';
        context.strokeRect (20+i, 30+i, 100+i, 50+i);
        context.strokeRect (510-(i+(100+i)), 30+i, 100+i, 50+i);
    }
}
```

La Fig. 4.1 muestra el resultado que despliega el navegador.

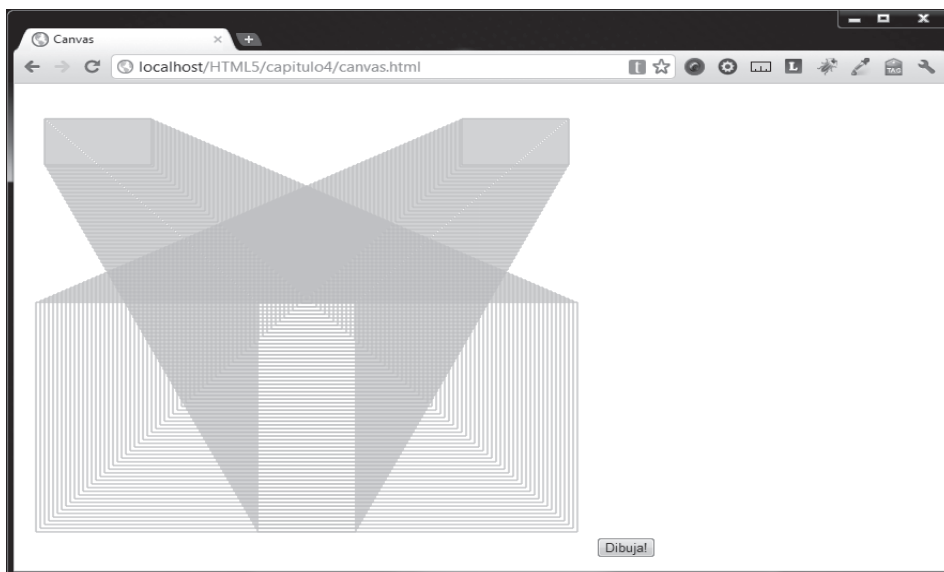


Fig. 4.1 Página con función JavaScript que dibuja rectángulos dentro de `<canvas>`.

Apoyo en la



En los materiales adicionales encontrará el archivo *canvas.html*

La API 2D tiene muchos métodos para trabajar con ella, de hecho por su importancia, el capítulo final de este libro estará completamente dedicado al elemento `<canvas>`, de tal manera que este punto es sólo una pequeña introducción, pero si tiene curiosidad por ver todos los métodos disponibles, la especificación oficial la encuentra en el enlace siguiente:

<http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html>

Nota

Por el momento el único contexto disponible es el contexto bidimensional (2d) pero se planea incluir soporte para gráficos 3d en el futuro.

4.5.3 embed

La etiqueta vacía `<embed>` es la etiqueta general para embeber cualquier tipo de media en una página. En teoría esta etiqueta permite embeber cualquier tipo de media que no es soportada nativamente por ninguna etiqueta. Utiliza el atributo `"type"` para especificar el tipo de contenido que vamos a embeber auxiliándose del tipo MIME del contenido.

Tome en cuenta que en la práctica no importa que utilice la etiqueta `<embed>` con todos sus atributos de manera correcta, si el cliente no cuenta con el plug-in apropiado, su contenido no será desplegado. Utilice esta etiqueta preferentemente sólo como método de respaldo a etiquetas más confiables como `<audio>` y `<video>`.

Por otro lado, si desea utilizar contenido Flash en el contexto de HTML5, `<embed>` es la mejor manera de incluir este tipo de contenido.

```
<embed src="flashcontent01.swf"
      type="application/x-shockwave-flash"
      width="550"
      height="300" />
```

Apoyo en la



En los materiales adicionales encontrará los archivos:
embed.html y *flashcontent01.swf*.

4.5.4 svg

La etiqueta con cierre `<svg>` es otra opción disponible en HTML5 que permite construir gráficos vectoriales en nuestra página. Utiliza una estructura XML, es decir, también utiliza etiquetas, pero éstas son propias de `<svg>` y no se pueden utilizar fuera del ámbito de éste; estas etiquetas propias tienen atributos para permitir posición y apariencia, pero se puede utilizar JavaScript y CSS sin ningún problema. La característica más popular de `<svg>` es que permite escalar imágenes sin perder calidad. El siguiente código dibuja un cuadrado:

```
<!DOCTYPE html>
<html>
  <head>
    <title>SVG</title>
    <meta charset="iso-8859-1" />
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg"
        width="150"
        height="150">
      <rect width="90" height="90"
          x="30" y="30"
          style="fill:#0000ff;fill-opacity:0.75;stroke:#000000" />
    </svg>
  </body>
</html>
```

Este código mostraría un cuadro en el navegador como muestra la Fig. 4.2.

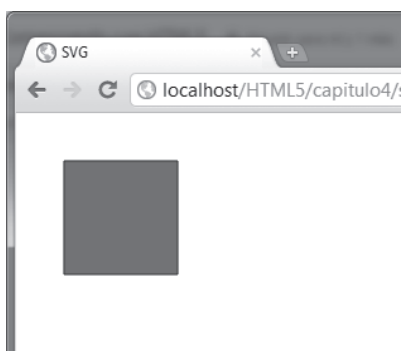


Fig. 4.2 Ejemplo de figura dibujada con `<svg>`.



En los materiales adicionales encontrará el archivo *svg.html*

Nota

El detalle del elemento `<svg>` escapa del enfoque de este libro, pero si desea profundizar puede consultar el sitio oficial W3C acerca del tema: <http://www.w3.org/Graphics/SVG/>

4.5.6 video

La etiqueta con cierre `<video>` es otro elemento que ha causado gran expectativa y ha estado involucrado en una interesante polémica (animada por el cofundador de Apple, Steve Jobs). He puesto especial esfuerzo en simplificar y al mismo tiempo ser lo más explícito posible en este tema, dado que puede ser un poco confuso para las persona que van comenzando.

Cuando vemos un video en un reproductor, normalmente lo que vemos son las imágenes (cuadros) del video y su respectivo audio, pero no usamos dos archivos separados para cada uno, si no que en su lugar solamente tenemos un solo archivo, quizá en formato *Mp4*, *AVI* o *MOV* entre otros. Pero éstos son sólo formatos contenedores, simplemente definen cómo almacenar la información de video y audio en un solo archivo. Cuando usted ve un video, su reproductor hace básicamente 3 cosas:

1. Determina qué formato contenedor se está utilizando para entonces encontrar qué video y audio están disponibles dentro y proceder a determinar cuál es la “herramienta” correcta para decodificar la información, a esta herramienta se le conoce como *codec*.
2. Procede a decodificar (usando un *codec*) la información de video para presentar la secuencia de imágenes (o cuadros) que lo componen en pantalla.
3. Procede a decodificar (usando un *codec*) la información de audio para enviarla a sus bocinas.

Ahora bien, existe polémica (y problemas) respecto a qué formatos contenedores y por ende qué *codec* de audio y video deberían ser usados, debido a que las compañías dueñas de los navegadores tienen diferentes posturas acerca de cuál debería ser el formato soportado para desplegar video, y esto a veces trae problema con la relación navegadores, tipos *MIME* y servidores Web. Sin embargo, podemos hablar de los principales contendientes hasta el momento:

- **Mp4** (Codec de Video: H.264, Codec de Audio: AAC): Los tres formatos son propietarios y por lo tanto tienen restricciones de patente.
- **Ogg** (Codec de Video: Theora, Codec de Audio: Vorbis): Éste es un formato y sus codecs son completamente libres.

- WebM (Codec de Video: VP8, Codec de Audio: Vorbis): Éste es un formato más nuevo, y sus codecs son de código abierto. VP8 fue un formato propietario hasta que Google compró a la compañía dueña y liberó el *codec*.

La etiqueta `<video>` trabaja de manera muy similar a la etiqueta `<audio>`. Si va a utilizar `<video>` utilice la etiqueta `<source>` para proporcionar diferentes alternativas al archivo origen de video y como respaldo le sugiero que utilice `<embed>` para un reproductor Flash. Cualquier texto entre las etiquetas `<video>` y `</video>` será desplegado en cualquier navegador que no soporte el elemento.

```
<video controls="controls">
  <source src="siberian_husky.ogg" />
  <source src="siberian_husky.mp4" />
  Su navegador no soporta la etiqueta video
</video>
```

Actividades para el lector

- 1.- Investigue cuáles de los navegadores más utilizados soportan el audio en formato Mp3 y el video en formato Mp4.
- 2.- Haga una lista con los navegadores que no soporten estos formatos, en otra lista indique si presentan alguna alternativa.
- 3.- Cree un documento donde utilice los elementos estructurales `<header>`, `<footer>`, `<nav>`, `<section>` y los que desee agregar.
- 4.- En su documento dentro de `<section>` coloque por lo menos tres tipos diferentes de elementos semánticos.

RESUMEN

En este capítulo se ha aprendido más acerca de la filosofía de la nueva especificación y de los nuevos elementos que permiten insertar gráficos y multimedia:

- Énfasis en la semántica en HTML5.
- Etiquetas para precisar mejor la estructura de nuestro contenido.
- Introducción al elemento para la creación y manipulación de imágenes dinámicas.
- Introducción a los nuevos elementos multimedia para audio y video.

Autoevaluación

1. ¿Qué es semántica en términos de HTML?
2. ¿Qué etiquetas se utilizan para marcar semánticamente la cabecera y el pie de un documento HTML5?
3. ¿En qué etiqueta se puede agrupar semánticamente a las etiquetas de encabezado `<h1>`-`<h6>`?
4. ¿Qué lenguajes se requieren para dibujar en el elemento `<canvas>`?
5. ¿Qué es un codec?

EVIDENCIA

☐

Diferencia entre los navegadores que soportan Mp3 y Mp4, de aquellos que no lo hacen, indica los formatos alternativo que pueden soportar.

☐

Utiliza en sus documentos los elementos `<header>`, `<footer>`, `<nav>` y `<section>`.

☐

Emplea elementos multimedia en sus páginas web.

REFERENCIAS

Bibliografía

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.

Fulton, Steve y Fulton, Jeff (2011). HTML5 Canvas 1a. ed., O'Reilly, EUA.



Páginas Web recomendadas

<http://diveintohtml5.org/>

<http://dev.w3.org/html5/spec/Overview.html>

<http://www.w3schools.com/html5/default.asp>

Respuestas sugeridas a las preguntas de autoevaluación

1. Es el uso apropiado de elementos HTML que ayudan a dar significado a la estructura del código en una página.
2. Las etiquetas `<header>` y `<footer>`.
3. En la etiqueta `<hgroup>`.
4. JavaScript.
5. Es la herramienta que permite interpretar o decodificar correctamente el formato utilizado para incluir información de video y/o audio.

Controles nuevos en formularios HTML5

5

Reflexione y responda las preguntas siguientes:

¿Qué controles nuevos existen para formularios?

¿En qué casos se utilizan los controles nuevos?

¿Cómo se comportan los controles nuevos?

¿Qué atributos nuevos comparten los controles nuevos para formulario?

Contenido

Formularios HTML5 (controles nuevos)

Introducción

5.1 Un vistazo a los controles nuevos

5.2 Elementos independientes

5.3 Elementos tipo input

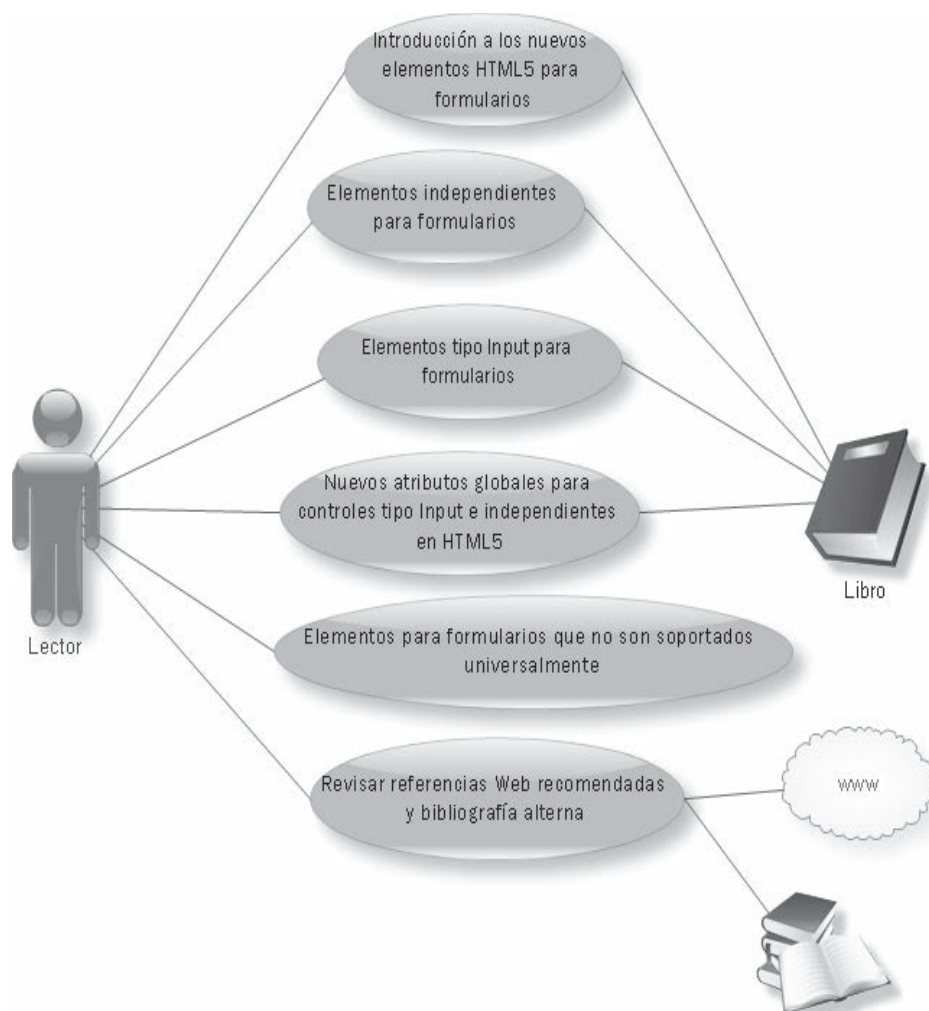
5.4 Nuevos atributos

Expectativa

Conocer y emplear la gama de posibilidades que ofrecen algunos de los controles de formularios de la nueva especificación HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Utilizar los nuevos controles para formulario de HTML5.
- Distinguir los nuevos elementos de formulario, independientes al tipo input.



INTRODUCCIÓN

Los formularios son algo muy utilizado en la Web, todo el mundo los ha usado alguna vez; en el capítulo 3, se dio un repaso de ellos con sus controles clásicos y su funcionamiento general dentro de las etiquetas `<form>` y `</form>`, ahora que ya se tiene una noción acerca del tema, se puede abordar en forma más técnica.

Aunque en este punto se ha avanzado, la verdad es que en el contexto de HTML5 no se conoce aún ni la mitad de sus nuevas capacidades, hay más de una docena de nuevos controles con interesantes características que antes sólo se podían lograr con aquellos controles desarrollados por terceros.

Nota

Opera 11 es, por el momento, el navegador más avanzado en lo que se refiere al uso de formularios, soportando todos los controles nuevos de la especificación.

5.1 Un vistazo a los controles nuevos

Formulario HTML5

localhost/HTML5/capitulo5/formulario_html5.html

Formulario con los nuevos controles HTML5

Controles Independientes

Security: 2048 (Grado elevado) (este control requiere trabajar con otro control de manera paralela)

meter:

progress:

Controles tipo Input

email: *Coloque dato inválido y utilice el botón "Enviar"

number:

range:

search: campo de búsqueda

tel: (este control requiere trabajar con algún script de cliente o servidor)

url: *Coloque dato inválido y utilice el botón "Enviar" (En el campo email debe haber un dato correcto y ya validado)

Enviar

Fig. 5.1 Formulario con los nuevos controles HTML5.

No todos los navegadores soportan todos los tipos nuevos de control para formulario, con excepción de Opera 11 que los soporta todos. Sin embargo, relativamente poca gente usa el navegador Opera, por lo que únicamente se tratará aquellos que son soportados por el navegador Chrome en su versión 14, pero una vez que el lector tenga mayor experiencia, lo invito a que pruebe las capacidades de otros navegadores.

En la Fig. 5.1 notará los controles que se analizan visualizados en el navegador, si se tiene alguna experiencia previa en el desarrollo de formularios HTML, de inmediato se dará cuenta que a simple vista algunos sugieren cosas interesantes.

El código de esta página es el siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Formulario HTML5</title>
    <meta charset="iso-8859-1" />
  </head>
  <body>
    <h1>Formulario con los nuevos controles HTML5</h1>
    <form id="formulario1">
      <fieldset form="formulario1" name="fieldset1">
        <legend>Controles Independientes</legend>
        <p>
          <label>Securtiy:</label>
          <keygen name="security" keytype="rsa" challenge="letmein"
/>
          <label>(este control requiere trabajar con otro control de
manera paralela)</label>
        </p>
        <p>
          <label>meter:</label><br/>
          <meter value="2" min="0" max="10">2 de 10</meter><br/>
          <meter value="0.6">60%</meter>
        </p>
        <p>
          <label>progress:</label><br/>
          <progress value="25" max="100">
            </progress>
        </p>
      </fieldset>
      <fieldset form="formulario1" name="fieldset2">
```

```

        <legend>Controles tipo Input</legend>
        <p>
            email: <input type="email" name="user_email"
            autofocus="autofocus" />
            <label>*Coloque dato inválido y utilice el botón
            "Enviar"</label>
        </p>
        <p>
            number: <input type="number" name="no_number" min="1"
            max="10" />
        </p>
        <p>
            range: <input type="range" name="no_range" min="1"
            max="10" />
        </p>
        <p>
            search: <input type="search" name="user_search"
            placeholder="campo de búsqueda" />
        </p>
        <p>
            tel: <input type="tel" name="user_mobile" />
            <label>(este control requiere trabajar con algún script
            de cliente o servidor)</label>
        </p>
        <p>
            url: <input type="url" name="user_url" />
            <label>*Coloque dato inválido y utilice el botón "Enviar"
            (En el campo email debe haber un dato correcto y ya
            validado)</label>
        </p>
    </fieldset>
    <input type="submit" />
</form>
</body>
</html>

```

Apoyo en la



En los materiales adicionales encontrará el archivo:
formulario_html5.html.

5.2 Elementos independientes

Los nuevos elementos se clasifican básicamente en aquellos que pertenecen a la etiqueta `<input>` y aquellos que no dependen de ésta para ser utilizados, a estos últimos son a los que se puede llamar independientes o referirlos simplemente por su nombre individual. Los controles independientes pueden ser utilizados fuera del contexto de un formulario en otro punto de sus páginas.

5.2.1 fieldset

Este control `<fieldset>` no es nuevo en realidad, sin embargo, la especificación HTML5 le asignó nuevos atributos (`disabled`, `form` y `name`). La etiqueta ayuda a agrupar en un bloque algunos elementos, en su forma común coloca un marco que delimita el contenido agrupado, y alternativamente puede emplear la etiqueta `<legend>` para añadir una etiqueta al bloque. La descripción más precisa de los nuevos atributos es:

- **Atributo** `disabled`. Desactiva todos los controles dentro del bloque.

Nota

El atributo `disabled` está en la especificación, al 30/10/2011 no funciona con Chrome.

- **Atributo** `form`. Indica específicamente a qué formulario pertenece el `<fieldset>` colocando el ID del formulario como valor del atributo.
- **Atributo** `name`. Establece el nombre del `<fieldset>`.

El formulario de ejemplo se utiliza así:

```
<fieldset form="formulario1" name="fieldset1">
<legend>Controles Independientes</legend>
.
.
.
</fieldset>
```

5.2.2 keygen

El control `<keygen>` genera una clave de encriptación que hace referencia a algún dato para enviarlo bajo ese esquema de encriptación al servidor. Este control es sumamente debatido y su futuro es incierto, pero es soportado al momento por Chrome, Firefox y Opera.

- **Atributo** `keytype`. Indica el algoritmo de inscripción que utilizará.
- **Atributo** `challenge`. Es una cadena que se envía junto con la clave de encriptación (la cadena es construida en términos de DER encode).

El ejemplo en el formulario es así:

```
<keygen name="security" keytype="rsa" challenge="letmein" />
```

5.2.3 label

El elemento `<label>` no es nuevo, pero debido a la filosofía semántica de HTML5 se utiliza mucho más, a esto se le suma que puede ayudar en el momento de usar CSS para darle formato a los textos de manera más sencilla. Su atributo `for` establece la relación entre `<label>` y el elemento al que pertenece, si así se desea.

La única diferencia con el `<label>` de la versión anterior de HTML es que ahora se ha añadido el atributo `form` para indicar explícitamente a qué formulario pertenece.

En el formulario es utilizado como se muestra:

```
<label>
(esteste control requiere trabajar con otro control de manera
paralela)
</label>
```

5.2.4 meter

El control `<meter>` proporciona una medida visual escalar dentro de un cierto rango, o bien el valor porcentual de un valor total. Aunque `<meter>` se utiliza continuamente en formularios, se puede utilizar en cualquier punto de una página donde se requiera. La etiqueta `<meter>` soporta los argumentos siguientes:

- **Atributo `form`.** Especifica a qué formulario pertenece el control `<meter>`
- **Atributo `high`.** Indica a partir de qué punto de la medida total podría considerarse un valor como alto.
- **Atributo `low`.** Indica hasta qué punto de la medida total podría considerarse un valor como bajo.
- **Atributo `max`.** Máximo valor posible, por default su valor es 10.
- **Atributo `min`.** Mínimo valor posible, por default su valor es 0.
- **Atributo `optimum`.** Especifica cuál es el mejor valor en el control `<meter>`.
- **Atributo `value`.** Establece el valor actual de `<meter>`.

En el formulario se utilizarán 2 ejemplos:

```
<meter value="2" min="0" max="10">2 de 10</meter><br/>
<meter value="0.6">60%</meter>
```

5.2.5 progress

El control `<progress>` ayuda al usuario señalando qué tanto de alguna tarea ha sido realizado. Actualmente este control es sólo soportado por los navegadores Chrome y Opera, y lo presentan como una barra de progreso. Este elemento es controlado utilizando código JavaScript.

Aunque su uso es muy común en formularios, puede utilizarse en cualquier parte de una página. Los atributos disponibles son:

- **Atributo** `max`. Especifica el valor máximo total en el control.
- **Atributo** `value`. Define el valor actual de progreso en el control.

En el formulario se usa:

```
<progress value="25" max="100"></progress>
```

5.3 Elementos tipo input

Los formularios HTML utilizan constantemente al versátil elemento `<input>`. Como ya se ha visto en el punto 3.7 esta etiqueta permite crear botones, cajas de selección y campos de texto, entre otros, cambiando el valor del atributo `input`, pero ahora en HTML5 la variedad de posibilidades se ha incrementado, permitiéndo crear nuevos tipos de controles.

Es justo aclarar una vez más, que el soporte para estos nuevos controles no ha sido adoptado por todos los navegadores, e incluso en aquellos que ya han aceptado determinado control, pueden tener su propia interpretación y presentarlo con variaciones con respecto a otros navegadores.

Nota

Cuando un navegador no soporta cierto tipo de control, usualmente se coloca un control de tipo texto simple en su lugar.

5.3.1 email

El tipo `email` se presenta como un campo de texto estándar que debe contener un correo electrónico, de lo contrario, cuando llegue el momento de enviar la información del formulario se hará la validación de este campo y el navegador informará de alguna manera si el texto colocado no es válido. En nuestro formulario ejemplo:

```
<input type="email" name="user_email" autofocus="autofocus" />
```

5.3.2 number

El tipo `number` está hecho específicamente para permitir la entrada de datos numéricos. Usualmente se presenta acompañado con algún tipo de elemento visual para introducir sólo números, en el caso de Chrome, con dos pequeña flechas que señalan hacia arriba y abajo para incrementar y decrementar el valor respectivamente. Los atributos que se pueden manejar al usar el tipo `number` son:

- **Atributo** `max`. Especifica el valor máximo permitido.
- **Atributo** `min`. Especifica el valor mínimo permitido.
- **Atributo** `step`. Indica el intervalo entre valores que se permite (si `step="2"` entonces los valores permitidos podrían ser -2, 0, 2, 4, etcétera).
- **Atributo** `value`. El valor por default.

En el formulario:

```
<input type="number" name="no_number" min="1" max="10" />
```

5.3.3 range

El tipo `range` se utiliza para permitir la entrada de valores que deben estar dentro de cierto rango definido de números. El tipo `range` se presenta comúnmente como elemento desplazable a manera de barra desplazable o “slider”. Los atributos con los que se puede contar son:

- **Atributo** `max`. Especifica el valor máximo permitido.
- **Atributo** `min`. Especifica el valor mínimo permitido.
- **Atributo** `step`. Indica el intervalo entre valores que se permite (si `step="2"` entonces los valores permitidos podrían ser -2, 0, 2, 4, etcétera).
- **Atributo** `value`. El valor por default.

En el ejemplo dentro del formulario:

```
<input type="range" name="no_range" min="1" max="10" />
```

5.3.4 search

El `<input>` de tipo `search` es utilizado como un campo de texto que sirve para hacer búsquedas, ya sean internas o que utilicen un servicio de búsqueda, como Google. Aunque normalmente luce como un campo de texto simple, algunos navegadores le dan alguna funcionalidad adicional, por ejemplo, Chrome y Safari agregan una pequeña X que permite borrar el contenido.

```
<input type="search" name="user_search" placeholder="campo de  
búsqueda" />
```

5.3.5 tel

El tipo `tel` es un campo de texto que sirve para introducir números telefónicos. A diferencia de los tipos `email` y `url`, el navegador no hace una validación automática al enviar la información del formulario, por lo que requerirá de JavaScript y el atributo `pattern` para validar la correcta información en este campo.

```
<input type="tel" name="user_mobile" />
```

5.3.6 url

La finalidad del tipo `url` es permitir la entrada de direcciones Web. Si el usuario envía la información del formulario se hará una validación automática sobre este campo, verificando la existencia del prefijo `http://`, si la validación resulta negativa el navegador lo informará de alguna manera.

```
<input type="url" name="user_url" />
```

Actividades para el lector

Realice un documento con un formulario donde emplee al menos 3 elementos independientes y cuatro elementos tipo `input`.

5.4 Nuevos atributos

La especificación HTML5 no sólo llegó con nuevos controles de formulario, sino que también incluye algunas ventajas extra que se pueden aplicar en cualquiera de estos controles. Expongo sólo los atributos que son soportados por el navegador Chrome 14, pero existen algunos más que aún no son soportados universalmente.

5.4.1 autofocus

Este atributo `autofocus` especifica qué elemento obtendrá primero el foco del usuario en el formulario al cargarlo, es decir, indicará en cuál control podrá comenzar a introducir información el usuario. Por supuesto, sólo tiene sentido si se usa `autofocus` únicamente en un control del formulario. En el formulario de ejemplo se utiliza `autofocus` en el campo tipo `email`:

```
<input type="email" name="user_email" autofocus="autofocus" />
```

5.4.2 pattern

El atributo `pattern` se utiliza para especificar una expresión regular que servirá para validar el texto introducido en un elemento `<input>` de tipo `text`, `search`, `url`, `telephone`, `email` o `password`. Este atributo debería ser utilizado sólo si una validación común no funciona. El análisis profundo acerca del manejo de este atributo escapa del objetivo de este libro, ya que requiere de conocimiento de JavaScript, si desea saber más al respecto consulte bibliografía JavaScript y el objeto `RegExp`.

5.4.3 placeholder

El atributo `placeholder` permite colocar una cadena de texto dentro de los campos de texto de cualquier tipo, la cadena se mostrará mientras que éstos no tengan ningún texto introducido por el usuario. Tan pronto como el usuario haga clic en el control, el texto desaparecerá. En el ejemplo del formulario se utiliza `placeholder` en el campo tipo `search`:

```
<input type="search" name="user_search" placeholder="campo de
busqueda" />
```

Actividades para el lector

En el documento que realizó en la actividad anterior manipule el atributo correspondiente para que el cuarto elemento del formulario creado obtenga el foco al cargar la página.

Actividades para el lector

- 1.- Investigue y haga una lista de todos los controles HTML5 para formularios.
- 2.- Indique en una lista por separado cuáles controles no son soportados por Google Chrome.
- 3.- Detalle la diferencia entre las dos listas anteriores para reconocer cuáles controles son soportados por Google Chrome.

RESUMEN

A lo largo de este capítulo se ha hecho referencia a las nuevas características de los controles para formularios en HTML5 que consisten en:

- Los nuevos elementos independientes, es decir, aquellos que no son un elemento de tipo input.
- Elementos de tipo input disponibles en HTML5.
- Nuevos atributos para los controles de formulario.

Autoevaluación

1. ¿Se puede utilizar algún control nuevo de formulario, fuera de un formulario?
2. ¿Cuál es la diferencia entre los controles de tipo `number` y `range`?
3. ¿Cuál es la diferencia entre un control de tipo input y otro que no lo es?
4. ¿Qué atributo debería usar si desea comenzar a llenar un formulario por un campo específico?

EVIDENCIA

☐

Realizó un documento con un formulario donde empleo al menos 3 elementos independientes y cuatro elementos tipo input.

☐

Manipuló el documento para obtener el foco del usuario al cargar la página.

☐

Investigó y enumeró los controles para formularios HTML5., detecto cuales no son soportados por Google Chrome.

☐

Reconoce cuáles controles para formularios HTML5 sí son soportados por Google Chrome.

REFERENCIAS

Bibliografía

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.



Páginas Web recomendadas

<http://diveintohtml5.org/>

<http://dev.w3.org/html5/spec/Overview.html>

<http://www.w3schools.com/html5/default.asp>

Respuestas sugeridas a las preguntas de autoevaluación

1. Sí, los elementos se pueden usar fuera de la etiqueta `<form>`.
2. El control de tipo `number` acepta sólo números y no necesariamente tiene un límite superior e inferior, mientras que `range` sí tiene límites inferior y superior y funciona mediante un scroll.
3. Los controles tipo `input` son una implementación del elemento `<input>` y el atributo `type`, los independientes son elementos que son una etiqueta en sí mismos.
4. El atributo `autofocus`.

Primeros pasos con CSS

6

Reflexione y responda las preguntas siguientes:

¿Qué es CSS?

¿Cómo controlar la apariencia de su página?

¿Cómo agrupar elementos para controlar su apariencia?

¿Cómo controlar la posición de los elementos de una página?

Contenido

Primeros pasos con CSS

Introducción

6.1 Un vistazo a hojas de estilo o CSS (Cascading Style Sheets)

6.2 Utilizar estilos a nivel local

6.3 Utilizar estilos a nivel documento

6.4 Utilizar estilos a nivel de sitio (link)

6.5 Utilizar clases y el identificador (class e id)

6.6 Manipular la apariencia de una página

6.7 Agrupación de elementos (span y div)

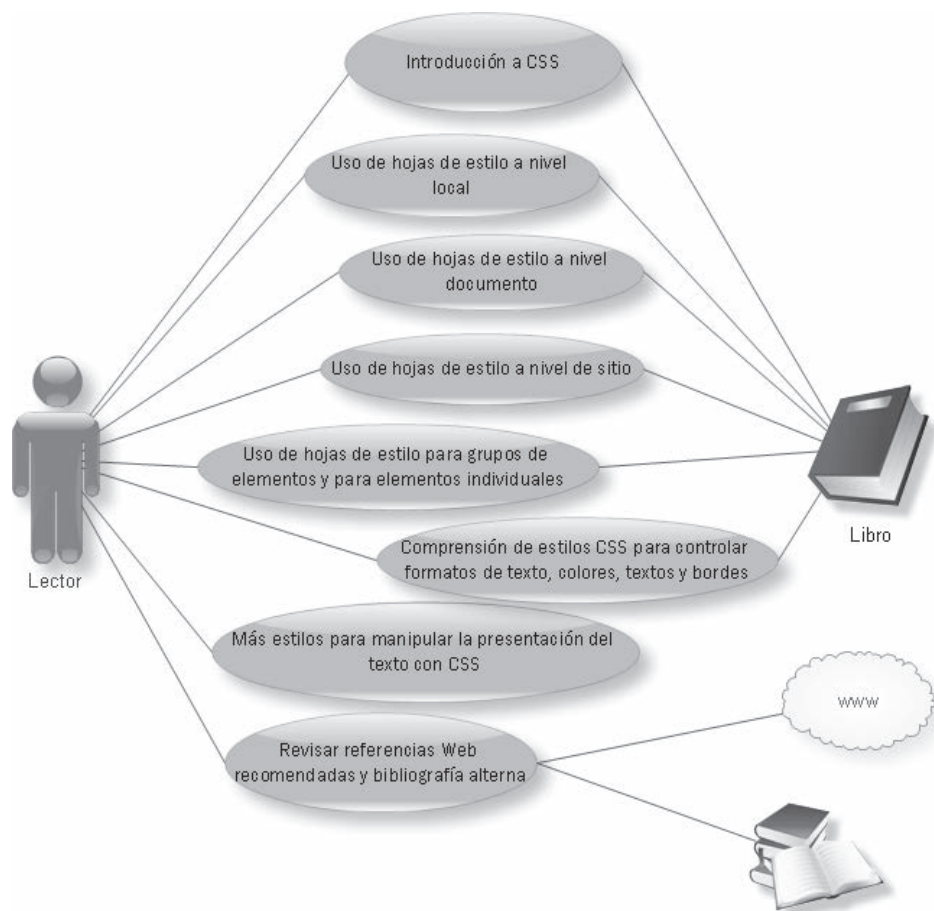
6.8 Posicionamiento

Expectativa

Conocer cómo separar el contenido de sus páginas HTML5 del código, para controlar su aspecto con CSS.

Después de estudiar este capítulo, el lector será capaz de:

- Utilizar los aspectos básicos de las CSS.
- Controlar la apariencia de páginas HTML con estilos locales y estilos externos.
- Comprender la filosofía de posicionamiento de elementos HTML con CSS.



INTRODUCCIÓN

El HTML original era mucho más enfocado al contenido como tal que a la forma en que se presentaba, y la Web lucía de esta manera, es decir, la estética y la organización visual del contenido no era algo importante en los primeros días de la Web. Aunque esta situación era comprensible e incluso adecuada en algún momento, las necesidades fueron cambiando, los dueños de los navegadores se dieron cuenta de la situación y añadieron etiquetas y atributos propios que proporcionaban nuevas capacidades, pero al mismo tiempo se creó cierta complejidad y desorden.

Ahora con la nueva especificación HTML5, se intenta regresar a la idea original en la cual HTML se concentraba en el contenido. Las etiquetas que estaban dedicadas a dar formato, posición o color en HTML han sido discontinuadas, así es, etiquetas como `<center>`, `` y ``, entre otras, ya no están disponibles. Pero no se preocupe, como algunos programadores Web ya sabrán, ahora se dispone (oficialmente) con un lenguaje completamente dedicado a darle un buen aspecto a sus páginas, el lenguaje es llamado CSS (*Cascading Style Sheets*).

Si usted está familiarizado con CSS no le afecta de gran manera si decide no estudiar este capítulo, pero si está iniciándose en el mundo del desarrollo Web, es el momento de comenzar y familiarizarse con el lenguaje.

6.1 Un vistazo a hojas de estilo o CSS (*Cascading Style Sheets*)

El funcionamiento de las hojas de estilo o CSS consiste en definir, mediante cierta sintaxis, la apariencia y presentación de nuestras páginas a diferentes niveles:

- **A nivel local.** Una etiqueta en concreto, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta. Esto es muy importante, ya que ofrece potencia en nuestra programación, se puede definir, por ejemplo, varios tipos de párrafos: en rojo, en azul, con márgenes, sin ellos.
- **A nivel de página usando la cabecera.** En un documento HTML o página, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- **A nivel de archivos externos.** Los estilos pueden especificarse desde un archivo separado, y referenciados desde cualquier página, esto proporciona la posibilidad de definir la forma de todo un sitio de una sola vez.

En los próximos puntos de este capítulo se analizará más a fondo estas tres diferentes maneras de usar las hojas de estilo.

Aunque lo anterior pone en evidencia varias ventajas de las CSS, no son las únicas, también es posible controlar la posición y las distancias de muchos de los elementos en una página con mucha mayor precisión, al grado de poder optar por diferentes unidades de medida como pulgadas, centímetros, puntos y píxeles.

6.2 Utilizar estilos a nivel local

El uso de CSS permite hacer que sólo cierta parte de una página muestre un estilo determinado, esto se logra utilizando a dicho estilo en alguna determinada etiqueta. Por ejemplo, se puede definir un párrafo entero en color rojo y otro en color azul. Para ello se maneja el atributo “style”, que es soportado por la mayoría de las etiquetas del HTML. Considere la página siguiente:

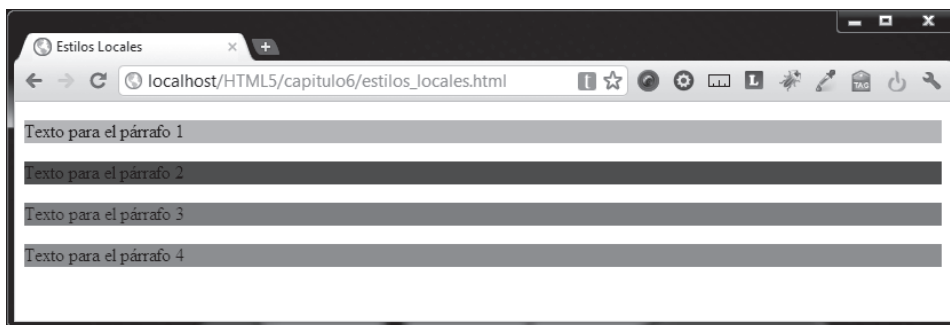


Fig. 6.1 Página con diferentes estilos sobre párrafos con CSS.

A continuación se detalla el código de la página mostrada en la figura 6.1.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos Locales</title>
    <meta charset="iso-8859-1" />
  </head>
  <body>
    <p style="background-color: aqua">
      Texto para el párrafo 1
    </p>
    <p style="background-color: blue">
      Texto para el párrafo 2
    </p>
    <p style="background-color: fuchsia">
```

```
        Texto para el párrafo 3
    </p>
    <p style="background-color: gray">
        Texto para el párrafo 4
    </p>
</body>
</html>
```

Nótese que se utiliza el atributo “style” en cada párrafo, se hace uso de la propiedad `background-color` para indicar el color que se desea de fondo, de manera que lo único que se ve afectado es un párrafo de manera individual.

Usted puede usar esta técnica para afectar una determinada etiqueta, pero no es lo más conveniente, ya que el abuso podría llevarlo a tener que hacer un mantenimiento demasiado laborioso, es decir, si deseara modificar el estilo de sus etiquetas tendría que hacerlo una por una, en cada archivo `.html` donde se encuentren, y esto es justo lo que CSS trata de evitar, sin embargo, la técnica puede ser útil en situaciones particulares.

Dado que los colores no tienen mucho sentido en blanco y negro, puede descargar el código de este ejemplo y verlo en su navegador.



Para apreciar los colores del ejemplo descargue de los materiales adicionales el archivo: `estilos_locales.html`.

6.3 Utilizar estilos a nivel documento

Podemos definir estilos en la sección de la etiqueta `<head>` de un documento, dichos estilos se pueden aplicar a toda la página. Es una manera muy cómoda y potente de darle forma a un documento, ya que estos estilos serán aplicados en toda la página y nos ahorrará la necesidad de aplicar estilo individual a muchas etiquetas HTML para afectar a todo el documento. Si desea cambiar los estilos de la página, se hará de una sola vez en la cabecera de la misma, sin tocar en lo absoluto a las etiquetas HTML.

El siguiente es un ejemplo de cómo se utiliza CSS en el ámbito de `<head>` y `</head>`. Por ahora lo más importante es la comprensión de conceptos básicos, no se preocupe si no entiende a detalle el código CSS en el ejemplo, a medida que avance en el libro y practique irá desarrollando su habilidad para comprender y escribir código.

El primer elemento que se debe agregar para incluir estilos que afecten todo un documento en la cabecera es la etiqueta con cierre `<style>` dentro del ámbito de `<head>`, en `<style>` se colocarán los diferentes estilos que desee. Los estilos se definirán indicando a qué etiqueta desea agregar alguno y enseguida se especifica las reglas de ese estilo entre los símbolos de llave (`{}`). Observe el ejemplo siguiente:

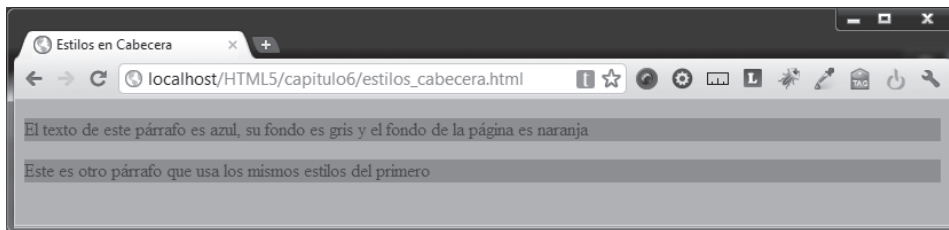


Fig. 6.2 Página que utiliza estilos CSS en la cabecera del documento `<head>`.

El código para la página que se muestra en la Fig. 6.2 es el siguiente:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos en Cabecera</title>
    <meta charset="iso-8859-1" />
    <style>
      body {
        background-color:orange;
      }
      p {
        color:blue;
        background-color:grey;
      }
    </style>
  </head>
  <body>
    <p>
      El texto de este párrafo es azul, su fondo es
      gris y el fondo de la página es naranja
    </p>
    <p >
      Éste es otro párrafo que usa los mismos estilos
      del primero
    </p>
  </body>
</html>
```

Observe que a diferencia del ejemplo del punto 6.2, en donde especifica el estilo a cada etiqueta, en este ejemplo ambos párrafos existentes usan el mismo estilo especificado una sola vez en la cabecera de la página y no fue necesario tocar en ningún momento el código HTML dentro de la sección `<body>`. Éste es la belleza de CSS, hace el código HTML mucho más limpio controlando lo concerniente a la apariencia de las páginas en otro lugar.



Para apreciar los colores del ejemplo descargue de los materiales adicionales el archivo: *estilos_cabecera.html*.

6.4 Utilizar estilos a nivel de sitio (link)

Una de las características más potentes de la programación con hojas de estilos consiste en poder definir y cambiar los estilos para todo un sitio Web de una sola vez. Esto se consigue creando un archivo donde únicamente se coloca las declaraciones de estilos de la página y enseguida enlace todas las páginas que componen un sitio con ese archivo. De esta forma, todas las páginas comparten una misma declaración de estilos y, por lo tanto, si cambia algo en esa declaración, también cambiarán todas las páginas. Con las ventajas añadidas de que se ahorra en líneas de código HTML (lo que reduce el peso del documento) y se evita la molestia de definir una y otra vez los estilos con el HTML, como se comentó en el punto anterior. Analice la página en la Fig. 6.3:



Fig. 6.3 Página que utiliza estilos CSS desde un archivo externo.

Ponga especial atención en el código de esta página y no en que el elemento `<style>` no existe, ni absolutamente ningún código CSS.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos en Cabecera</title>
    <meta charset="iso-8859-1" />
    <link rel="stylesheet" type="text/css"
href="estiloExterno01.css" />
```



```
</head>
<body>
  <h1>Esta página utiliza un Estilo Externo</h1>
  <p>
    El texto de este párrafo es azul, su fondo es gris y el
    fondo de la página es naranja
  </p>
  <p >
    Éste es otro párrafo que usa los mismos estilos del
    primero
  </p>
</body>
</html>
```

Esta página es muy similar a la del ejemplo del punto 6.3, la diferencia interesante es que claramente tiene un estilo, pero la información no está directamente escrita en la página, en su lugar se utiliza la etiqueta vacía `<link>`.

La etiqueta vacía `<link>` dentro de la sección `<header>` permite asociar otro archivo con la página actual, su uso más común es precisamente para incluir estilos. Los atributos utilizados cuando se usan hojas de estilo son los siguientes:

- **Atributo** `href`. Especifica la dirección URL del archivo que se desea vincular.
- **Atributo** `rel`. Indica la naturaleza de relación entre el documento actual y el archivo vinculado por la URL. Cuando el archivo vinculado contiene cascadas de estilo, el valor es `stylesheet`.
- **Atributo** `type`. Especifica el tipo MIME del archivo vinculado, para las hojas de estilo el valor es `text/css`.

El estilo en un archivo externo puede usarse una y otra vez en múltiples páginas, y éstas reflejarán cualquier cambio en la hoja de estilos.

El archivo que contiene los estilos tiene extensión `.css` y puede modificarse con cualquier editor de texto plano, el código no requiere de la etiqueta `<style>`.

Aquí el código CSS del archivo que se utiliza en el ejemplo:

```
body {
  background-color: orange;
}
p {
  color:blue;
  background-color: grey;
}
```



Para apreciar los colores del ejemplo descargue de los materiales adicionales los archivos: `estilos_externos.html` y `estiloExterno01.css`.

Actividades para el lector

Para practicar sus conocimientos cree 3 documentos en ellos manipule el color del texto de un párrafo a rojo, utilice la declaración de estilo a nivel local, documento y sitio, una vez terminados modifique sus documentos a color azul.

6.5 Utilizar clases y el identificador (class e id)

Como se ha mencionado, CSS es muy práctico porque permite añadir o modificar un estilo para elementos HTML rápidamente. Se ha visto ejemplos donde se añade un estilo a algunas etiquetas en particular, o le fueron asignados estilos a la etiqueta `<p>`. Pero ¿qué pasaría si quisiera seguir afectando a los párrafos de la misma manera, excepto a uno o a dos?, este caso se muestra en la página muestra en la Fig. 6.4:



Fig. 6.4 Página con estilos referenciados usando los atributos `class` e `id`.

Si se ha contestado que podría resolver este problema empleando la técnica de estilos a nivel local, tiene usted razón, pero si le queda la sensación de una piedra en el zapato, permítame felicitarlo porque comienza a pensar como ¡programador! Resolver esto con estilos locales resultaría ineficiente y poco elegante. Para resolver esto se cuenta con los atributos `id` y `class` que permiten aplicar estilos a una etiqueta en particular o a un grupo de ellas.

El código HTML y CSS de la página que se muestra en la Fig. 6.4 es:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos con Identificadores y Clases </title>
    <meta charset="iso-8859-1" />
    <link rel="stylesheet" type="text/css"
href="estiloExterno02.css" />
  </head>
  <body>
    <h1>Esta página utiliza un Estilo Externo con
Identificadores y Clases
    </h1>
    <p id="elegante">
      Este párrafo usa el estilo elegante para los párrafos
    </p>
    <p>
      Este párrafo usa el estilo general para los párrafos
    </p>
    <p class="blancas" >
      Este párrafo utiliza un estilo de letras blancas
    </p>
    <p >
      Este párrafo usa el estilo general para los párrafos
    </p>
    <p class="blancas" >
      Este párrafo utiliza un estilo de letras blancas
    </p>
  </body>
</html>
```

Si tiene experiencia construyendo páginas Web no le resulta nuevo el atributo “id”, el cual sirve para dar un identificador único a una etiqueta HTML, en este caso se usa `id="elegante"` para el primer párrafo, el cual tiene su propio estilo dedicado. Después en el tercer y quinto párrafos se utiliza el atributo `class="blancas"`, en este caso ambos párrafos comparten el mismo estilo, y finalmente los dos párrafos restantes hacen uso del estilo general indicado para los párrafos.

El código siguiente corresponde al archivo vinculado con el ejemplo *estiloExterno02.css*, observe detenidamente el código y deduzca qué pasa en él.

```
body {
    background-color: orange;
}
p {
    color:blue;
    background-color: gray;
}
#elegante {
    color:black;
    background-color: yellow;
}
.blancas {
    color:white;
    background-color: green;
}
```

Después de observar el código HTML y su correspondiente CSS, es posible afirmar que lo que se ha agregado son algunos indicadores para distinguir entre tres tipos de párrafos:

- **Párrafos comunes.** Estos párrafos no requieren ninguna característica especial. Una vez vinculado el estilo, el código de estos párrafos queda intacto.
- **Párrafos identificados.** Estos párrafos usan el atributo `id` con un valor único, este atributo está disponible para prácticamente todas las etiqueta HTML, de manera que en este ejemplo, el estilo puede hacer referencia a un elemento en particular gracias a su valor en `id`.
- **Párrafos dentro de una clase.** Para dar aún mayor flexibilidad se puede asignar valor al atributo `"class"` y a cualquier elemento HTML. Esta propiedad permite indicar que el elemento HTML que la utiliza es miembro de ella. A diferencia del atributo `"id"` se puede tener tantos elementos como se desee utilizando la misma clase. El estilo por supuesto afecta a todos los miembros de una misma clase.

Para hacer uso de un `id` como referencia en la hoja de estilos, debe anteponer el signo `"#"` al nombre del valor, como se hace en el ejemplo con el `id` con valor `elegante`:

```
#elegante {
    color:black;
    background-color: yellow;
}
```

Utilice el carácter punto (.) antes del nombre de la clase para hacer referencia al valor del atributo “class”, como se hizo en el ejemplo:

```
.blancas {  
    color:white;  
    background-color: green;  
}
```

- Utilice el nombre de la etiqueta HTML cuando desee afectar todos los elementos de ese tipo.
- Use el atributo “class” cuando quiera afectar sólo a un grupo de elementos que son o no del mismo tipo.
- Use el atributo id cuando quiera afectar a un elemento individualmente.



Para apreciar los colores del ejemplo descargue de los materiales adicionales los archivos:
estilos_idclass.html y *estiloExterno02.css*.

6.6 Manipular la apariencia de una página

La cantidad de cosas que usted puede manipular con CSS es bastante grande, CSS utiliza una buena cantidad de propiedades para diferentes fines que ameritarían un libro entero, es por eso que sólo ejemplificaré algunas de las situaciones más comunes en la implementación de hojas de estilo.

Con las hojas de estilo es posible utilizar colores, modificar el formato de un texto, añadir y manejar bordes de diferentes elementos, usar imágenes como fondos y manejar la estructura y posición visual de una página.

6.6.1 Manipular texto

El texto está presente en la mayoría de las páginas y es quizá el elemento más importante en la Web, es por esa razón que CSS ofrece muchas características para manipular la apariencia del texto de los documentos. Las siguientes son algunas de las propiedades clásicas utilizadas en CSS:

- **Propiedad** color. Especifica el color del texto.
- **Propiedad** letter-spacing. Incrementa o decrementa la distancia entre los caracteres de un texto.
- **Propiedad** text-align. Especifica la alineación horizontal de un texto.
- **Propiedad** text-decoration. Indica la decoración de un texto. Los valores más utilizados son none, underline, overline y line-through.
- **Propiedad** vertical-align. Especifica la alineación vertical de un texto.

En la Fig. 6.5 se muestra un número de efectos de texto que puede lograr utilizando CSS y los atributos expuestos:

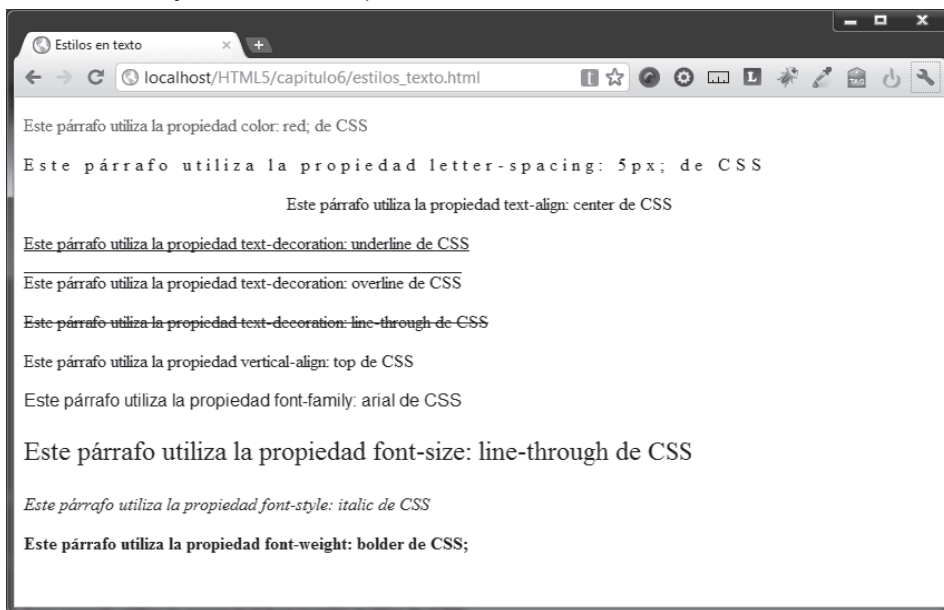


Fig. 6.5 Algunos efectos de texto que puede lograr utilizando CSS.

A continuación se muestra el código de esta página:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos en texto</title>
    <meta charset="iso-8859-1" />
    <link rel="stylesheet" type="text/css"
href="estiloExterno03.css" />
  </head>
  <body>
    <p class="color">
      Este párrafo utiliza la propiedad color: red; de
CSS
    </p>
    <p class="letter_spacing">
      Este párrafo utiliza la propiedad letter-
spacing: 5px; de
CSS
    </p>
    <p class="text-_lign">
```

```

        Este párrafo utiliza la propiedad text-align:
center de CSS
    </p>
    <p class="underline">
        Este párrafo utiliza la propiedad text-
decoration: underline de CSS
    </p>
    <p class="overline">
        Este párrafo utiliza la propiedad text-
decoration: overline de CSS
    </p>
    <p class="through">
        Este párrafo utiliza la propiedad text-
decoration:
        line-through de CSS
    </p>
    <p class="vertical_align">
        Este párrafo utiliza la propiedad vertical-
align: top de CSS
    </p>
    <p class="font_family">
        Este párrafo utiliza la propiedad font-family:
arial de CSS
    </p>
    <p class="font_size">
        Este párrafo utiliza la propiedad font-size:
line-through de CSS
    </p>
    <p class="font_style">
        Este párrafo utiliza la propiedad font-style:
italic de CSS
    </p>
    <p class="font_weight">
        Este párrafo utiliza la propiedad font-weight:
bolder de CSS;
    </p>
</body>
</html>

```

El código HTML de esta página es muy sencillo, es esencialmente un grupo de párrafos con texto, lo más notable es que cada párrafo es miembro de una clase en particular, para esto por supuesto se hace uso del atributo “class”.

En el archivo *estiloExterno03.css* se implementa todas las propiedades de hojas de estilo expuestas en este punto.

```
.color {  
    color: red;  
}  
.letter_spacing {  
    letter-spacing: 5px;  
}  
.text-_lign {  
    text-align: center;  
}  
.underline{  
    text-decoration: underline;  
}  
.overline {  
    text-decoration: overline;  
}  
.through {  
    text-decoration: line-through;  
}  
.vertical_align{  
    vertical-align: top;  
}  
.font_family{  
    font-family: arial;  
}  
.font_size{  
    font-size: 150%;  
}  
.font_style{  
    font-style: italic;  
}  
.font_weight{  
    font-weight: bolder;  
}
```

Nota

Existen aún más propiedades para manipular texto, si desea conocerlas todas consulte la bibliografía dedicada a CSS o bien puede consultar la página oficial: <http://www.w3.org/Style/CSS/>

Apoyo en la



Para apreciar los colores del ejemplo descargue de los materiales adicionales los archivos: *estilos_texto.html* y *estiloExterno03.css*

6.6.2 Manipular colores

En los ejemplos realizados hasta este capítulo se han manipulado esencialmente colores, utilizando nombre de colores en inglés como orange, green, etc., a veces se requiere tener una variedad más amplia de colores, pero CSS sólo cuenta con 147 colores diferentes si se hace de esta manera.

¿Recuerda cuando usted asistía al jardín de niños o a alguna escuela de educación básica?, ahí muy probablemente en algún momento le dieron algunos pigmentos de color rojo, verde y azul, los cuales mezcló para obtener nuevos colores. En el mundo de las computadoras (incluyendo su monitor y las hojas de estilo) las cosas suceden con la misma base. Usted tiene que decidir qué cantidad de rojo, verde y azul utilizará para crear un color, de manera simple, manejará un porcentaje para cada uno de los colores básicos, por ejemplo: 50, 50, 0.

Sin embargo, los equipos de cómputo trabajan distinto, en el porcentaje el rango va usualmente de 0 a 100, pero en nuestro caso el rango es de 0 a 255, cada valor de color (rojo, verde y azul) toma dos dígitos que representan su color en numeración hexadecimal (base 16) que va de 00 (total ausencia de color) a FF (máximo de brillo). Al final lo que se obtiene se conoce como notación hexadecimal de un color.

No se angustie, todo esto suena mucho más difícil de lo que es en realidad, además hoy en día hay una gran cantidad de herramientas que le ayudan en la creación de colores. En la Fig. 6.6 se muestra un analizador de color, que es una página desarrollada para que el usuario pruebe los colores que puede crear.



Fig. 6.6 Página analizador_color.html.

La página *analizador_color.html* tiene un funcionamiento muy simple, utiliza un control `range` para cada color primario, cuando mueve los sliders (`range`) el color de fondo de la página cambia y el formato hexadecimal de ese color se muestra en el campo de texto en la parte superior.

Utilizando cualquier valor que resulte en el campo de texto, puede sustituir cualquiera de los nombres de colores que ha usado. Si quisiera usar una notación hexadecimal para el color del texto y el fondo de un párrafo, el código sería así:

```
p {  
    color: #FF0000;  
    background-color: #FFAD00;  
}
```

Observe que debe utilizar el símbolo (#) justo antes del número hexadecimal, la ventaja de usar la notación hexadecimal es que se obtiene la posibilidad de manejar muchos más colores que los reconocidos por su nombre en inglés en CSS. No tiene que aprender de memoria ninguna notación, se puede auxiliar de muchas herramientas para obtener el color que desea.



En los materiales adicionales encontrará el archivo:
analizador_color.html.

6.6.3 Manipular Bordes

Hasta el momento se ha visto cómo darle color al interior de nuestros elementos, pero las hojas de estilo también hacen posible cambiar la apariencia de los bordes de aquellos elementos HTML que tienen uno, como por ejemplo los párrafos y las tablas, entre otros. Para dar estilo a un borde se emplean las propiedades disponibles en CSS, las más utilizadas son:

- **Propiedad** `border`. Permite especificar todas las propiedades de un borde en una sola declaración.
- **Propiedad** `border-color`. Especifica el color de un borde por sus cuatro lados.
- **Propiedad** `border-style`. Especifica el estilo de un borde por sus cuatro lados.
- **Propiedad** `border-width`. Especifica el grosor del borde por sus cuatro lados.

La Fig. 6.7 muestra el uso de las propiedades `border`, `border-color`, `border-style`, `border-width`, así como diferentes estilos que puede utilizar.

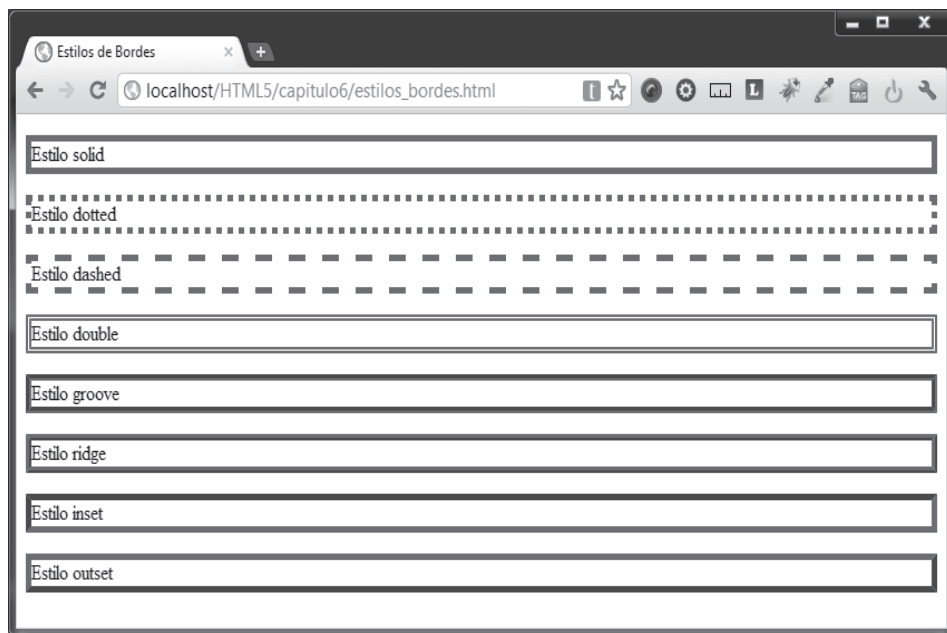


Fig. 6.7 Página que muestra diferentes estilos de bordes.



En los materiales adicionales encontrará el archivo: *estilos_bordes.html*.

Existen más propiedades para manipular con más detalle el estilo de un borde, por ejemplo puede manipular los lados de manera independiente con `border-left`, `border-right`, `border-top`, `border-bottom`.

Actividades para el lector

Investigue a profundidad todas las propiedades para dar estilo a los bordes.

6.7 Agrupación de elementos (span y div)

Las etiquetas `` y `<div>` se usan para agrupar otros elementos y así ayudar a estructurar un documento. Efectivamente, son elementos HTML y no CSS, pero en la práctica se utilizan ambos elementos, máxime cuando se tiene la intención de combinarlos con hojas de estilo; es por esto que los explico en este capítulo. Debido a esta relación tan común, a menudo se verá utilizando también los atributos “class” e “id”, se expusieron en el punto 6.5. Ahora se revisará el uso de los elementos (`` y `<div>`).

6.7.1 span

El elemento con cierre `` es lo que se denomina un elemento neutro que no provee ningún cambio visual en sí, pero con CSS `` actúa como un gancho hacia cierto texto o cierta parte de un documento. Cuando algo está enganchado con `` puede aplicar estilos sobre eso o manipularlo con JavaScript.

Ejemplo

Nos interesa resaltar ciertas partes del contenido de un texto:

```
<blockquote  
  cite="http://www.proverbia.net/citasautor.asp?autor=327">  
  <span>Todos</span> somos muy <span>ignorantes</span>. Lo que ocurre  
  es que no todos ignoramos las <span>mismas cosas</span>.  
</blockquote>
```

Ahora se utilizará `` en las partes de texto que se quiere resaltar, sólo resta agregar el estilo adecuado.

```
span {  
  color:red;  
}
```

La Fig. 6.8 muestra el resultado en el navegador.



Fig. 6.8 Página que utiliza `` con estilos CSS.

Se podría haber usado las propiedades `class` o `id`, pero recuerde que usted como desarrollador es el que decide cuándo será apropiado usarlas.

6.7.2 div

Mientras que `` se usa dentro de un elemento a nivel de una etiqueta tipo bloque (`<p>`), como se vio en el ejemplo anterior, `<div>` se usa para agrupar uno

o más elementos, siendo el mismo `<div>`, el bloque que envuelve a éstos. Además de esta diferencia, la agrupación con `<div>` funciona de manera similar.

Ejemplo

Dos listas de razas caninas, divididas según su primera letra.

```
<div id="letraA">
  <ul>
    <li>Akita</li>
  </ul>
</div>

<div id="letraB">
  <ul>
    <li>Beagle</li>
    <li>Black Cocker Spaniel</li>
    <li>Bloodhound</li>
    <li>Boston Terrier</li>
    <li>Boxer</li>
  </ul>
</div>

<div id="letraC">
  <ul>
    <li>Chihuahua</li>
    <li>Chow Chow</li>
    <li>Cocker Spaniel</li>
  </ul>
</div>
```

En la parte de estilos se tiene:

```
#letraA {
background: blue;
}

#letraB {
background: red;
}

#letraC {
background: green;
}
```

Puede ver la apariencia de la página en la Fig. 6.9.

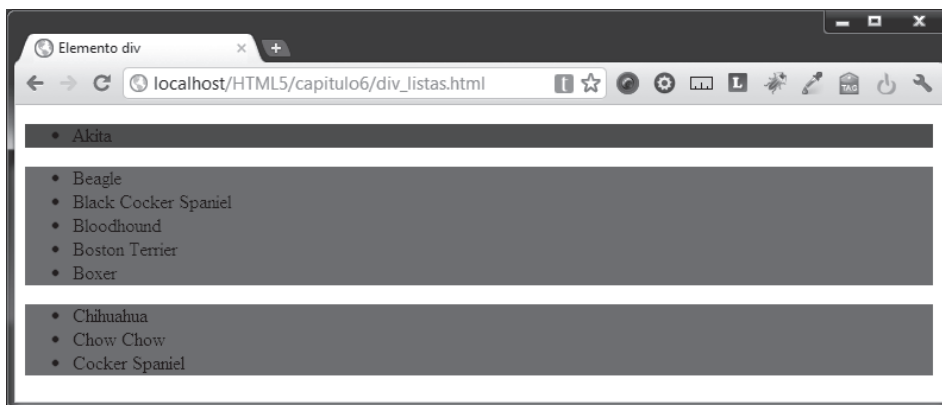


Fig. 6.9 Uso de `div` en el ejemplo: Dos listas de razas caninas.



En los materiales adicionales encontrará el archivo:
ejemplo div listas.html.

Nota

Se ha utilizado estilos simples tanto con `<div>` como con ``.
El lector puede hacer estilos más complejos, el mecanismo es el mismo.

Actividades para el lector

- 1.- Cree un documento con cinco elementos `<div>`. Utilice clases para asignar a tres de ellos fondo verde. Con identificadores determine para los restantes rojo y azul respectivamente.
- 2.- En un nuevo documento utilice ``, emplee estilos para asignar a 5 palabras color naranja.

6.8 Posicionamiento

Durante mucho tiempo la distribución de la estructura de las páginas ha sido una de las cosas más problemáticas de HTML. Los desarrolladores utilizan diversos trucos para lograr posicionar los elementos en el lugar correcto. Sin embargo, ahora las hojas de estilo facilitan la vida proporcionando una serie de alternativas para colocar los elementos donde se desea.

6.8.1 Elementos flotantes (propiedades `float` y `clear`)

El posicionamiento flotante funciona definiendo la relación entre elementos, más que indicando de manera explícita en qué lugar aparecen. Los elementos se pueden hacer flotar a la derecha o a la izquierda usando la propiedad `float`. Es decir, un elemento con su contenido flota bien a la derecha o bien a la izquierda de un documento.

Por ejemplo, si quisiera texto con ajuste de línea alrededor de una imagen, se podría utilizar el siguiente código:

```
<div id="imagen">

</div>
<p>Rhoncus nec porttitor est rhoncus, turpis, ultricies cras...</p>
```

Para conseguir que la imagen flote a la izquierda y el texto se ajuste a su alrededor, sólo hay que definir la propiedad “float” con el valor “left” y el ancho de la caja definida por <div> que rodea la imagen:

```
#imagen {
float: left;
width: 200px;
}
```

La implementación de este código se muestra en la Fig. 6.10.



Fig. 6.10 Página con propiedad float.



En los materiales adicionales encontrará el archivo: *estilos_flotante.html*.

Otra manera de implementar la propiedad “float” es la creación de columnas en un documento. Para crear tales columnas tendrá que estructurar las co-

lumnas deseadas en el código HTML con la etiqueta <div>, como se muestra a continuación:

```
<div id="columna1">
<p>Rhoncus nec porttitor est rhoncus, turpis, ultricies cras...</p>
</div>
<div id="columna2">
<p>Rhoncus nec porttitor est rhoncus, turpis, ultricies cras...</p>
</div>
<div id="columna3">
<p>Rhoncus nec porttitor est rhoncus, turpis, ultricies cras...</p>
</div>
```

Enseguida se establece el ancho de las columnas que puede ser un valor fijo o un porcentaje, como en este caso, y se coloca el valor “float” en los estilos:

```
#columna1 {
float:left;
width: 33%;
}

#columna2 {
float:left;
width: 33%;
}

#columna3 {
float:left;
width: 33%;
}
```

La Fig. 6.11 muestra el resultado de este código.

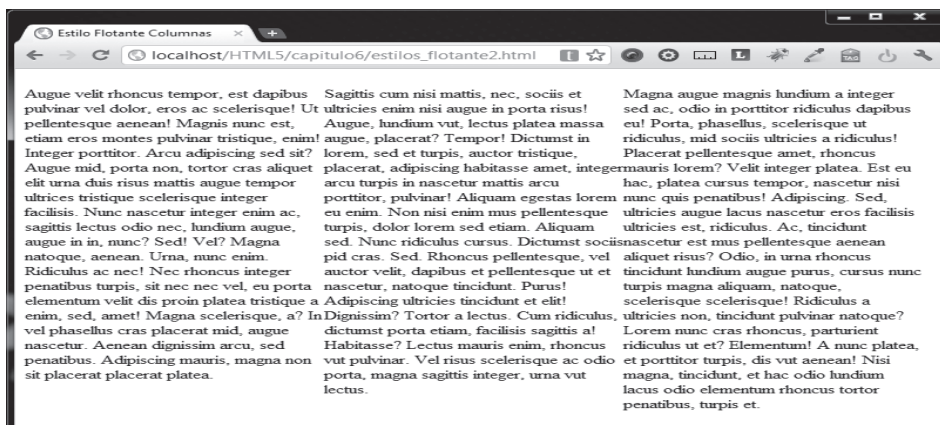


Fig. 6.11 Creación de columnas en un documento con la propiedad “float”.



En los materiales adicionales encontrará el archivo: *estilos_flotante2.html*.

La propiedad “clear” se usa para controlar cómo se comportarán los elementos que siguen a los elementos flotantes de un documento. Por defecto, los elementos siguientes se mueven hacia arriba para rellenar el espacio disponible que quedará libre al flotar una caja hacia un lado. En el ejemplo con la foto de nuestro amigo canino, el texto se desplaza en forma automática hacia arriba junto a la imagen.



Fig. 6.12 Texto en un documento con la propiedad “clear”.

La propiedad “clear” puede tomar los valores *left*, *right*, *both* o *none*. El principio consiste en que, si “clear”, por ejemplo, se fija en “both” para una caja, el borde del margen superior de esta caja siempre estará debajo del borde del margen inferior para las posibles cajas flotantes que vengan de arriba. Para comprender mejor esto véase la Fig. 6.12. Respecto al código HTML y el primer ejemplo, lo único que cambió fue que en esta ocasión se hizo uso del atributo “class” con el valor “sinflotar”:

```
<p class="sinflotar">Rhoncus nec porttitor est rhoncus, turpis, ult...</p>
```

Al código CSS se le añadió el estilo con la propiedad “clear” para impedir que el texto flotara junto a la imagen.

```
.sinflotar {
clear: both;
}
```



En los materiales adicionales encontrará el archivo: *estilos_clear.html*

6.8.2 Posicionamiento absoluto

Cuando se quiere posicionar un elemento de manera totalmente independiente, fuera del esquema flotante o de la relación con otro elemento, con un control más preciso, se puede hacer indicando de manera explícita la posición exacta donde se colocará el elemento.

El mecanismo es simple, cada página es como un gran eje de coordenadas donde la posición inicial es la esquina superior izquierda, de manera que se utilizan tres simples propiedades de CSS:

- **Propiedad** `position`. Especifica el tipo de posicionamiento a utilizar. En este caso el valor será “absolute”.
- **Propiedad** `top`. Para un posicionamiento absoluto determina la distancia entre la parte superior de la página y la parte superior del elemento afectado.
- **Propiedad** `left`. Para un posicionamiento absoluto determina la distancia entre la parte izquierda de la página y la parte izquierda del elemento afectado.

La Fig. 6.13 muestra un ejemplo de posicionamiento absoluto.



Fig. 6.13 Ejemplo de posicionamiento absoluto.

El código HTML es el siguiente:

```
<h1>Ejemplo de Posicionamiento Absoluto</h1>
<p >
Un bicho en la página!!!!Un bicho en la página!!!!Un bicho en la
página!!!!Un bicho en la página!!!!Un bicho en la página!!!!Un
bicho en la página!!!!Un bicho en la página!!!!Un bicho en la
página!!!!Un bicho en la página!!!!Un bicho en la página!!!!Un
bicho en la página!!!!Un bicho en la página!!!!Un bicho en la
página!!!!Un bicho en la página!!!!Un bicho en la
```

```

página!!!!Un bicho en la página!!!!Un bicho en la página!!!!Un
bicho en la página!!!!Un bicho en la página!!!!Un bicho en la
página!!!!Un bicho en la página!!!!Un bicho en la página!!!!Un
bicho en la página!!!!Un bicho en la página!!!!Un bicho en la
página!!!!Un bicho en la página!!!!
</p>
<div id="imagen">

</div>

```

El código CSS es el que hace el trabajo:

```

#imagen {
position: absolute;
    top: 100px;
    left: 280px;
}

```



En los materiales adicionales encontrará el archivo:
estilos_posabsoluta.html.

6.8.3 posicionamiento relativo

Para posicionar un elemento de forma relativa, la propiedad “position” se establece con el valor “relative”. La diferencia entre posicionamiento absoluto y relativo consiste en cómo se calcula la posición.

La posición se calcula desde la posición original en el documento. Esto significa que el elemento se mueve hacia la derecha, izquierda, arriba o abajo de donde estaba colocado originalmente. De este modo, el elemento sigue teniendo efecto en los elementos que lo rodean. Las propiedades serían para este caso de la siguiente manera:

- **Propiedad position.** Especifica el tipo de posicionamiento a utilizar. En este caso el valor será “relative”.
- **Propiedad top.** Para un posicionamiento relativo determina la distancia entre la parte superior del elemento afectado en su posición original y la parte superior del elemento afectado en su nueva posición.
- **Propiedad left.** Para un posicionamiento relativo determina la distancia entre la parte izquierda del elemento afectado en su posición original y la parte izquierda del elemento afectado en su nueva posición.

La Fig. 6.14 muestra el mismo ejemplo anterior en su versión relativa.

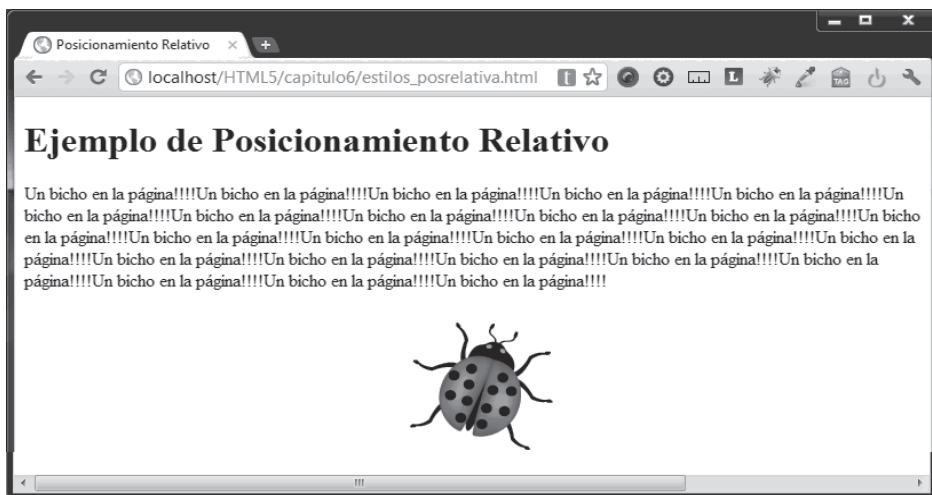


Fig. 6.14 Ejemplo de posicionamiento relativo.

El único cambio en el código con respecto al ejemplo anterior es el valor “relative” para la propiedad “position”:

```
#imagen {  
  position: relative;  
  top: 100px;  
  left: 280px;  
}
```

Existen muchas más cosas que se pueden hacer con las hojas de estilo de las que puedo abordar es este capítulo introductorio. Para profundizar más consulte las bibliografías disponibles dedicadas a CSS. El siguiente capítulo abordará algunas de las nuevas capacidades de la especificación CSS en su nueva versión, conocida como CSS3.



En los materiales adicionales encontrará el archivo:
estilos_posrelativa.html.

Actividades para el lector

- 1.- Construya un documento con dos elementos <div>, el primero con texto de color azul que flote a la izquierda del segundo <div> que contiene texto rojo.
- 2.- Elabore un documento con texto a 4 columnas utilizando <div>.
- 3.- Investigue y documente en forma de lista, los estilos para controlar la apariencia del texto que no se hayan mencionado en este capítulo, incluya la descripción de sus atributos y la versión de CSS a que pertenece.

RESUMEN

En este capítulo se ha introducido al lector a las reglas básicas de las hojas de estilo CSS (Cascading Style Sheets).

Conoció de manera general:

- Objetivos generales de CSS.
- Trabajo conjunto de HTML y CSS.
- Métodos para controlar el aspecto de elementos particulares, agrupados y generales con CSS.
- La manera de posicionar elementos HTML con CSS.

Autoevaluación

1. ¿Qué es CSS?
2. ¿Qué atributo de etiqueta se debe utilizar si requiere usar un estilo directamente en la etiqueta HTML?
3. ¿En qué sección de una página HTML se debe colocar estilos si requiere que tenga efecto sobre todo el documento?
4. ¿Qué atributo de etiqueta se debe usar si requiere afectar sólo algunos elementos con un estilo?
5. ¿Qué atributo de etiqueta debería usar si desea afectar únicamente a un elemento HTML?

EVIDENCIA

☐

Creó documentos donde manipuló el color del texto utilizando declaraciones de estilo a nivel local, documento y sitio.

☐

Investigó y documentó las propiedades para dar estilo a los bordes.

☐

Utilizó clases e identificadores para asignar fondos.

☐

Utilizó `` y estilos para asignar colores a palabras.

☐

Utilizó diversos tipos de posicionamiento, con elementos `<div>`.

☐

Investigó y documentó la descripción de los estilos para afectar el texto de un documento, incluyó la versión de CSS a la que pertenece.

REFERENCIAS

Bibliografía

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.

Hogan, Brian (2011). HTML5 and CSS3. Develop with Tomorrow's Standards Today, 1a. ed., Pragmatic Bookshelf, EUA.



Páginas Web recomendadas

<http://diveintohtml5.org/>

<http://dev.w3.org/html5/spec/Overview.html>

<http://www.w3schools.com/html5/default.asp>

Respuestas sugeridas a las preguntas de autoevaluación

1. Es un lenguaje que se utiliza para definir la presentación de un documento HTML.
2. El atributo `style`.
3. En la sección `<head>` y `</head>`.
4. El atributo `class`.
5. El atributo `id`.

Nuevo y mejorado CSS3

7

Reflexione y responda las preguntas siguientes:

¿Qué es lo nuevo en la versión CSS3?

¿Qué es lo que ha mejorado con CSS3?

¿Hay alguna limitante de compatibilidad entre CSS3 y navegadores?

¿Cómo se usan los nuevos elementos visuales?

¿Qué hay de nuevo con textos y fuentes?

¿Cuáles son las nuevas herramientas de selección?

Contenido

Nuevo y mejorado CSS3

Introducción

7.1 Motores de renderizado y CSS3

7.2 Nuevas notaciones para color

7.3 Bordos

7.4 Gradientes

7.5 Transformaciones

7.6 Transiciones

7.7 Transparencia

7.8 Texto y fuentes descargables

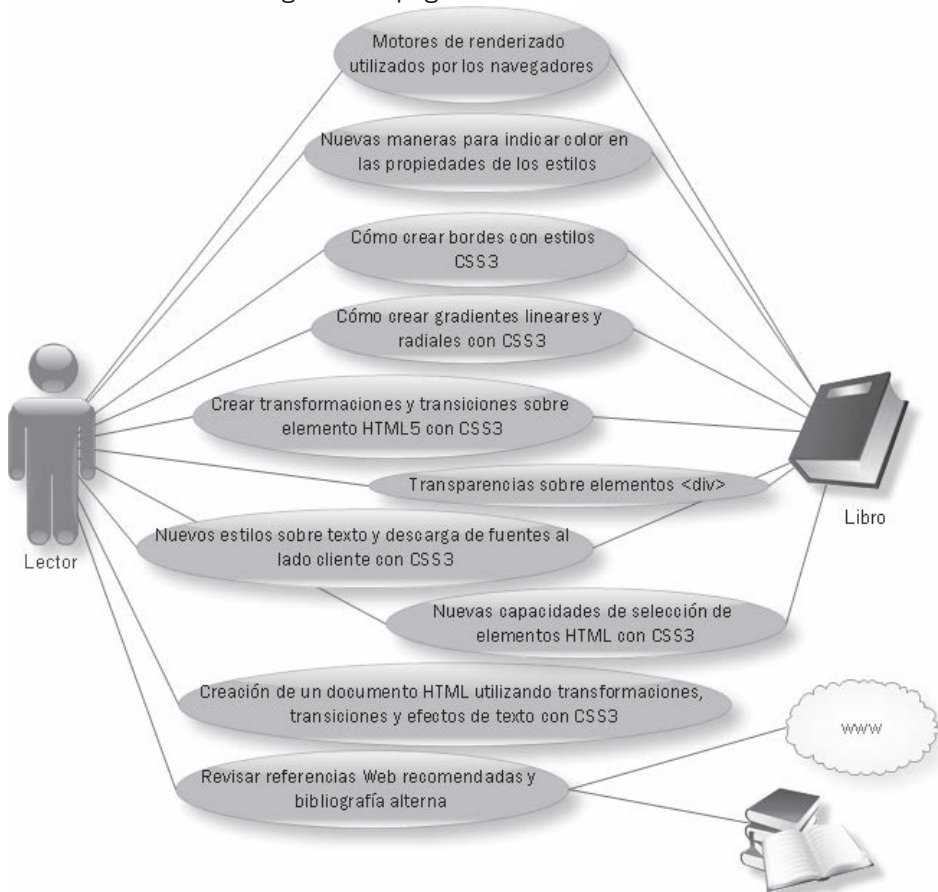
7.9 Herramientas de selección

Expectativa

Conocer las capacidades de la nueva especificación Cascading Style Sheet Level 3. Obtener las bases para mejorar la eficacia y eficiencia en la construcción y diseño de documentos HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Conocer las nuevas y mejoradas características en CSS3.
- Estar al tanto de las limitantes entre CSS3 y los navegadores.
- Hacer más eficiente la construcción y diseño de páginas HTML5.
- Ampliar la variedad de posibilidades para manipular la estética visual y la estética de código en sus páginas HTML5.



INTRODUCCIÓN

En el capítulo 6: “Primeros pasos en CSS”, se tuvo una introducción a las capacidades de las hojas de estilo o CSS, ahora se hablará de la nueva versión de las hojas de estilo, conocida como CSS Level 3 o solamente CSS3. Junto con la creación de la nueva especificación HTML5 también se desarrolla la nueva especificación para CSS3 con capacidades nuevas y mejorando otras ya existentes, esto se escucha muy bien, sin embargo, no todo es color de rosa, al igual que HTML5, CSS3 aún no es una especificación completa ni oficial, por lo que no todas sus características funcionan en todos los navegadores, o bien puede haber variaciones en el comportamiento de algunas características.

En el caso particular de la especificación CSS3, ésta no es una sola especificación general, sino que se divide en módulos. Cada módulo añade nuevas capacidades o extiende algunas ya existentes en especificaciones anteriores. La nueva especificación preserva compatibilidad con versiones anteriores.

El avance en el desarrollo de la especificación en cada módulo va cambiando al momento de escribir este libro (octubre del 2011), por lo que hay módulos con diferente nivel de avance. La manera más veraz de seguir el desarrollo de este lenguaje es verificando en la página oficial:

<http://www.w3.org/Style/CSS/current-work>

7.1 Motores de renderizado y CSS3

Los fabricantes de navegadores no suelen construir éstos completamente desde cero hoy en día, aunque algunos lo han hecho en el pasado, por lo que se auxilian de motores de renderizado (*rendering engines*) creados previamente, es decir, se basan en código de programación ya construido que permite visualizar elementos en el navegador que lo utiliza. Esto tiene importancia para CSS3 porque debido a que no es un estándar terminado, los motores de renderizado aún se están mejorando o actualizando y se tienen en fase de experimentación varias propiedades CSS3 y lo indican utilizando un prefijo, es importante señalar esto, ya que hay mucha información y ejemplos en libros y en línea que utilizan estos prefijos constantemente:

- **Prefijo -webkit-.** Indica que es una versión de prueba para las propiedades CSS, optimizada para los navegadores basados en el motor de renderizado WebKit. Chrome y Safari (incluyendo las versiones para iPhone y iPod) se basan en este motor.

- **Prefijo -o-.** Indica que es una versión de prueba para las propiedades CSS, optimizada para el motor de Opera.
- **Prefijo -moz-.** Indica que es una versión de prueba para las propiedades CSS, optimizada para el motor de renderizado Mozilla. Este motor es principalmente usado por Firefox, pero hay otros navegadores basados en el motor de Mozilla que están ahí afuera.
- **Prefijo -ms-.** Indica que es una versión de prueba para las propiedades CSS, optimizada para el motor de renderizado Microsoft para Internet Explorer. Son pocas las situaciones donde se usa, ya que la mayor parte de las veces Explorer simplemente soporta o no alguna propiedad.

7.2 Nuevas notaciones para color

Si usted leyó el punto 6.6.2: “Manipulando colores”, en el capítulo anterior o tiene experiencia con CSS, no será nuevo para usted el hecho de que los colores utilizan valores RGB en diferentes notaciones para definir colores.

Ahora CSS3 ofrece nuevas notaciones que no sólo permiten hacer lo mismo que ya hacían las anteriores, sino que también ofrecen nuevas características y facilidades para las personas con experiencia en diseño gráfico y/o edición de imágenes.

7.2.1 Notación RGBA

Para definir un color RGBA, se deben especificar cuatro valores, el primero para el color rojo (R), el segundo para verde (G), el tercero para azul (B) y el cuarto y novedoso en CSS que corresponde al *canal alfa*, es decir, el nivel de transparencia, por ejemplo:

```
background: rgba(255, 125, 0, 0.5);
```

Los tres primeros valores son números en sistema decimal entre 0 y 255, o bien pueden ser porcentajes de 0% a 100%, los cuales corresponden con los valores de rojo, verde y azul. El cuarto valor es un número entre 0 (completamente transparente) y 1 (totalmente opaco).

7.2.2 Notación HSL Y HSLA

La notación HSL debe su nombre a las primeras letras de *hue* (tono), *saturation* (saturación) y *lightness* (brillo), este formato no es el favorito de los programadores, pero tiene mucho sentido para las personas relacionadas con el diseño gráfico y programas de edición de imágenes.

Hue (tono) representa el grado en el círculo de color (de 0 a 360 grados), 0 o 360 es rojo, 120 es verde, 240 es azul. *Saturation* (saturación) es un porcentaje donde 0% es una sombra de gris y 100% es el color total. *Lightness* (brillo) también es un porcentaje donde 0% es negro y 100% es blanco, ejemplo:

```
background: hsl(120,65%,75%);
```

Probablemente ya se ha imaginado cómo es la notación HSLA, ésta es el mismo modelo HSL, agregando el canal alfa para indicar el nivel de transparencia:

```
background: hsla(120,65%,75%,0.3);
```

7.3 Bordes

Si tiene alguna experiencia utilizando bordes o leyó el punto 6.6.3: “Manipulando bordes”, quizá ha podido concluir que utilizar estilos para manipular la presentación de un borde, es una buena manera de mejorar la estructura visual de sus páginas.

CSS3 mejora esta posibilidad y ahora permite brindar características que anteriormente implicaban mucho trabajo artesanal, como esquinas redondeadas, sombras, imágenes en los bordes (en caso exclusivo de Firefox incluso gradientes en los bordes).

A continuación verá una página con ejemplos de los estilos para borde CSS3 y su código, posteriormente se hará el análisis de cada borde, tenga presente que algunos navegadores usan aún el prefijo experimental respectivo y pueden presentar algunas diferencias en su comportamiento.

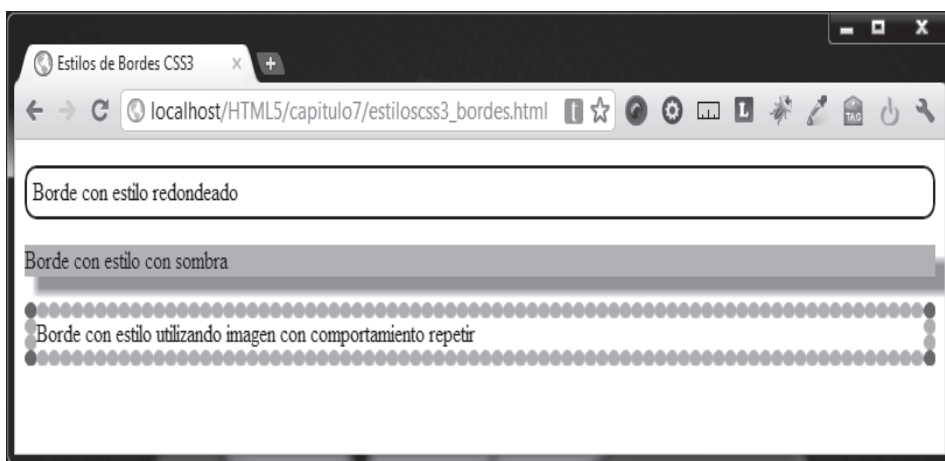


Fig. 7.1 Implementación de los nuevos estilos de borde comunes en CSS3.



En los materiales adicionales encontrará el archivo:
estiloscss3_bordes.html

Considere a continuación el código de este ejemplo para enseguida analizarlo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Estilos de Bordes CSS3</title>
    <meta charset="iso-8859-1" />
    <style>
      #rounded{
        border:2px solid;
        border-radius:10px;
        padding: 5px;
      }
      #shadow{
        background-color:orange;
        box-shadow: 10px 10px 5px #888888;
        -webkit-box-shadow: 10px 10px 5px #888888; /* Safari y
Chrome */
      }
      #image_repeat{
        border-width:10px;
        -moz-border-image:url(imagen_borde.png) 30 repeat; /*
Firefox */
        -webkit-border-image:url(imagen_borde.png) 30 repeat; /*
Safari y Chrome */
        -o-border-image:url(imagen_borde.png) 30 repeat; /* Opera */
        border-image:url(imagen_borde.png) 30 repeat;
      }
    </style>
  </head>
  <body>
    <p id="rounded" >
      Borde con estilo redondeado
    </p>
    <p id="shadow" >
      Borde con estilo con sombra
    </p>
    <p id="image_repeat" >
      Borde con estilo utilizando imagen con comportamiento
repetir
  </body>
</html>
```

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

7.3.1 Bordes redondeados

Crear bordes redondeados es algo laborioso con CSS2 y CSS1, también hay otros métodos, como la manipulación de celdas en tablas para lograr este mismo efecto, pero estas técnicas resultan laboriosas y confusas para algunas personas cuando comienzan a desarrollar, además de requerir crear imágenes para cada esquina. Con el uso de la especificación CSS3 esto se vuelve mucho más sencillo y puede aplicarse a una sola esquina o a todas ellas, detalles como el ancho y el color se pueden alterar con gran facilidad. El código para lograr este efecto es:

```
#rounded{
    border:2px solid;
    border-radius: 10px;
    padding: 5px;
}
```

Su sintaxis es muy simple, `border-radius` puede tomar hasta cuatro argumentos, uno para cada esquina en orden de las manecillas del reloj, en este caso un solo valor indica que aplica el mismo para las 4 esquinas. También es posible redondear esquinas individualmente con `border-top-left-radius`, `border-top-right-radius`, etc., esto puede ser útil en algunos casos, sin embargo, tome en cuenta que algunos navegadores pueden requerir aún el prefijo experimental.

7.3.2 Bordes sombreados

Añadir sombra a un elemento, sin usar CSS3, es algo complicado, ya que es necesario hacer un buen trabajo de manipulación de imágenes, del uso de colores y de espacios. CSS3 facilita el proceso, y aunque aún hay que calcular espacios y distancias, el trabajo con imágenes ya no existe. El código CSS3 que se utilizó en el ejemplo:

```
#shadow{
    background-color:orange;
    box-shadow: 10px 10px 5px #888888;
    -webkit-box-shadow: 10px 10px 5px #888888; /* Safari y Chrome
*/
}
```

Las sintaxis que se utiliza:

```
box-shadow: sombra-h sombra-v difuminado tamaño color
comportamiento;
```

- **sombra-h.** Parámetro requerido que establece la posición horizontal de la sombra.
- **sombra-v.** Parámetro requerido que establece la posición vertical de la sombra.
- **difuminado.** Indica la distancia del efecto de difuminado.
- **tamaño.** Indica el tamaño de la sombra.
- **color.** Indica el color de la sombra.
- **comportamiento.** Establece si la imagen se proyectará desde el elemento hacia afuera con el valor `outset` o del elemento hacia adentro con el valor `inset`.

7.3.3 Bordes con imagen

CSS3 permite usar una imagen como elemento para formar un borde. El mecanismo es poderoso, pero puede ser confuso, se detectan las orillas de una imagen y se separan en partes para crear las esquinas y orillas del borde, basándose en las orillas y esquinas detectadas de la imagen. El código utilizado en el ejemplo para este estilo es el siguiente:

```
#image_round{
    border-width:10px;
    -moz-border-image:url(imagen_borde.png) 30 repeat; /* Firefox */
    -webkit-border-image:url(imagen_borde.png) 30 repeat; /* Safari y
Chrome */
    -o-border-image:url(imagen_borde.png) 30 repeat; /* Opera */
    border-image:url(imagen_borde.png) 30 repeat;
}
```

La sintaxis para utilizar este estilo es:

```
border-image: imagen arriba derecha abajo izquierda comportamiento1
comportamiento2;
```

- **imagen.** Establece la ruta de la imagen base para llenar el borde.
- **arriba, derecha, abajo e izquierda.** Indican en dónde se harán los cortes de la imagen base, desde sus filos superior, derecho, inferior e izquierdo externos hasta el centro de la misma, y de esta manera utilizar los

fragmentos cortados de la imagen para rellenar las esquinas, las partes laterales y las partes superior e inferior del borde. Si indica un solo valor se tomará como si los 4 valores fueran iguales (en el ejemplo se indicó sólo 30, lo que es igual a 30 30 30 30).

- **comportamiento1 y comportamiento2.** La imagen base original y los cortes resultantes de la misma, rara vez coinciden con lo largo y alto del borde, por lo que básicamente tiene las opciones de estirar la imagen o repetirla hasta rellenar el largo o el alto del borde, para esto puede utilizar el valor `repeat` para repetir la imagen, o bien, el valor `stretch` para estirla. Si indica sólo un valor como comportamiento, se tomará como el mismo valor para *comportamiento1* y *comportamiento2* (en el ejemplo sólo se utilizó un `repeat`, lo cual indica que es `repeat` para ambos parámetros).

La idea puede ser confusa al principio, pero siga estos sencillos pasos, experimente un poco y pronto dominará la lógica de este estilo:

1. **Prepare la imagen base.** La imagen debe ser previamente preparada para su uso como imagen base de un borde. Utilice una imagen cuyas dimensiones sean cerradas, sencillas y que formen un cuadro preciso, si hay huecos en el diseño utilice imágenes con transparencia o colores sólidos de fondo. En el ejemplo se utiliza una imagen de 90 x 90 píxeles con círculos exactamente de 30 x 30 para poder dividir fácilmente los cortes. En la Fig. 7.2 se muestra la imagen utilizada en el ejemplo.



Fig. 7.2 Imagen base para borde con imagen utilizada en el ejemplo.

2. **Especifique el ancho del borde.** Es necesario que indique el ancho del borde con la propiedad `border-width`, ya que el corte de la imagen base será escalado para adaptarse a este tamaño automáticamente.
3. **Indique el tamaño que se va a utilizar en cada esquina y lado de la imagen.** Para cada lado y esquina del nuevo borde de nuestro elemento, indique el tamaño de los cortes en la imagen base que desea usar, recuerde que cada línea de corte es del filo hacia adentro y en el orden de las agujas del reloj, es decir, arriba, derecha, abajo e izquierda (véase la Fig. 7.3). Se pueden expresar en porcentajes o sin ninguna unidad de medida para expresar píxeles exactos, es decir, por ejemplo 30 30 30 30).

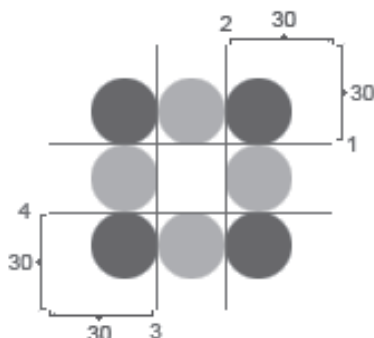


Fig. 7.3 Líneas de corte imaginarias para obtener el borde del ejemplo.

4. **Señale el comportamiento deseado.** La última parte es indicar el comportamiento, determina cómo se van a escalar y cortar los bordes de los lados y superior e inferior de nuestro elemento. Los posibles valores:
 - `stretch` (estirar)
 - `repeat` (repetir)
 - `round` (repetir y ajustar)

Nota

El valor `round` también repite la imagen, pero previene un corte abrupto ajustando automáticamente la imagen que se está repitiendo, pero los navegadores aún no hacen diferencia con `repeat` en la práctica.

Si se especifica sólo un valor, es como poner dos veces el mismo (por ejemplo sólo una vez `repeat`. Entonces será `repeat` tanto para los lados superior e inferior como para los lados izquierdo y derecho). El valor por defecto es `stretch`.

7.4 Gradientes

Un gradiente consiste en una secuencia de al menos dos colores, como por ejemplo, imagine la imagen del cielo donde se va de un azul muy claro a un azul muy oscuro, podría hablar entonces de un gradiente que va de un color a otro. Los tipos de gradientes más comunes son los gradientes *linear* (donde el cambio de un color a otros fluye a lo largo de una línea) y *radial* (donde un color parte de un punto central y los siguientes colores se alejan circularmente de ese punto).

Los gradientes se utilizan constantemente en las páginas, pero hasta el momento la manera de hacer un gradiente requería de crear una imagen en algún programa editor de imágenes y después colocar esa imagen en algún fondo (si se utilizaban hojas de estilo se usaba el atributo `background-image`).

Con CSS3 ya no es necesario crear ninguna imagen, lo puede hacer directamente en el navegador con código de hojas de estilo. Como ya sabe, CSS3, al no ser un estándar terminado no es utilizado de la misma manera en todos los navegadores, aún utilizan prefijos experimentales, aunque la manera de implementar los gradientes hoy en día es prácticamente igual.

A continuación se analizan las diferentes sintaxis para crear un gradiente lineal.

Sintaxis para Chrome y Safari:

```
background: -webkit-linear-gradient ( orientación, color-ini stop-ini, color-N stop-N );
```

Sintaxis para Firefox:

```
background: -moz-linear-gradient ( orientación, color-ini stop-ini, color-N stop-N );
```

Sintaxis para Opera:

```
background: -o-linear-gradient ( orientación, color-ini stop-ini, color-N stop-N );
```

- **orientación.** La orientación del efecto gradiente, los valores pueden ser `top`, `bottom`, `left`, `right` (aunque también puede usar el ángulo, por ejemplo, `90deg`).
- **color-ini.** El color con el que inicia el gradiente.
- **stop-ini.** A partir de qué punto se debe comenzar el efecto de desvanecimiento del color inicial (%).
- **color-N.** El color con el que continúa el gradiente.
- **stop-N.** Desde qué punto se debe detener el efecto de desvanecimiento del color actual para continuar con color sólido (%).

La Fig. 7.4 muestra una página que presenta varios ejemplos de gradiente con la descripción del estilo que están usando.



En blanco y negro no se puede apreciar el efecto gradiente con sus respectivos colores, por esta razón descargue el archivo `estiloscss3_gradientelinear.html` y pruébelo en su navegador.

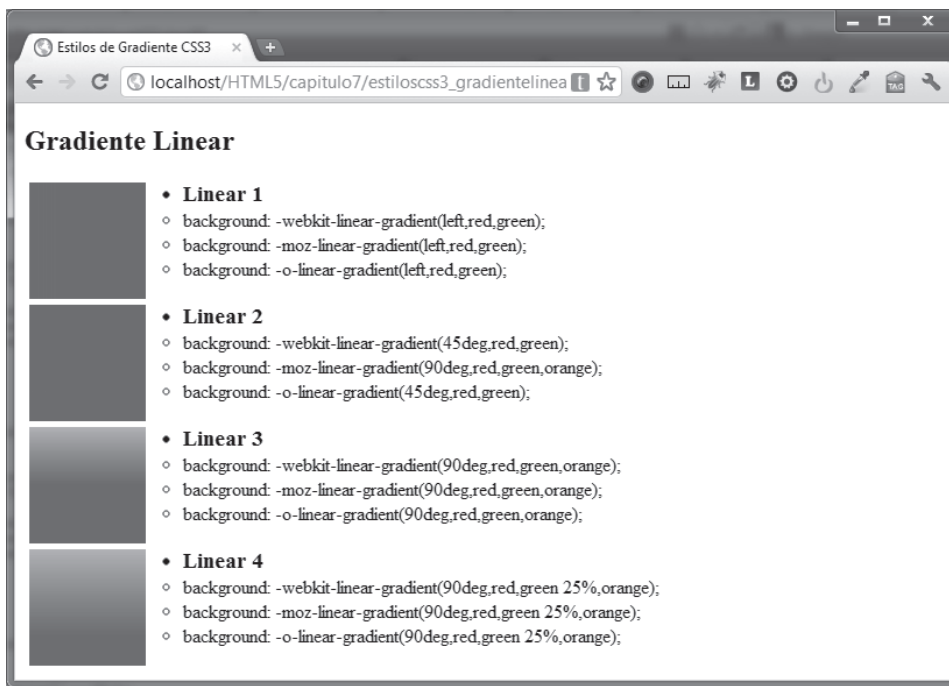


Fig. 7.4 Ejemplos de gradiente linear.

Para crear ahora un gradiente radial analice las sintaxis que se exponen a continuación:

Sintaxis para Chrome y Safari:

```
background: -webkit-radial-gradient( [posición,] [forma || tamaño,]
color-ini [stop-ini], color-N [stop-N] );
```

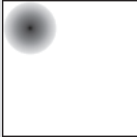
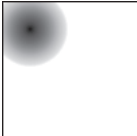
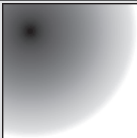

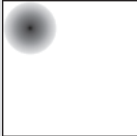

En la sintaxis para Firefox la única diferencia es el prefijo -moz-:

```
background: -moz-radial-gradient( [posición,] [forma || tamaño,]
color-ini [stop-ini], color-N [stop-N] );
```

Nota

Por ahora no es posible hacer gradientes radiales con Opera utilizando estilos de la misma forma.

- **posición.** Parámetro opcional que define la posición X, Y del punto central de inicio para el gradiente y el primer color, se pueden usar unidades, porcentajes y los valores `right`, `left`, `top`, `bottom` o `center`. La posición es relativa al contenedor del gradiente.
- **forma.** Parámetro opcional que indica si la forma del gradiente será un círculo (valor `circle`) o una elipse (valor `ellipse`).
- **tamaño.** Parámetro opcional que toma uno de los siguientes valores: `closest-side`, `closest-corner`, `farthest-side`, `farthest-corner`, `contain` o `cover`, todos estos valores hacen referencia al contenedor. Dado que los valores no son demasiado explícitos por sí mismos, observe los ejemplos para cada caso en la siguiente tabla:

EJEMPLO	DESCRIPCIÓN
	closest-side El filo del gradiente se prolongará sólo hasta llegar al lado más cercano con respecto al punto central establecido en el parámetro posición, en este caso, posición: 20px 20px.
	closest-corner El filo del gradiente se prolongará sólo hasta llegar a la esquina más cercana con respecto al punto central establecido en el parámetro posición (en este caso posición: 20px 20px).
	farthest-side Esto hace exactamente lo opuesto al primer valor, provocando que el filo del gradiente busque el lado más lejano al punto central indicado con el parámetro posición.
	farthest-corner El filo del gradiente se prolongará hasta llegar a la esquina más lejana con respecto al punto central establecido en el parámetro posición.
	contain Indica que el gradiente se creará sólo en los límites del elemento contenedor (para este caso se obtiene un efecto igual que closest-side).
	cover Significa que el gradiente continúa hasta cubrir el área entera del contenedor (para este caso se logra un efecto igual que farthest-corner).

La Fig. 7.5 muestra la implementación completa de estos parámetros en una página.



En los materiales adicionales encontrará el archivo *estiloscss3_gradienteradialsize.html*

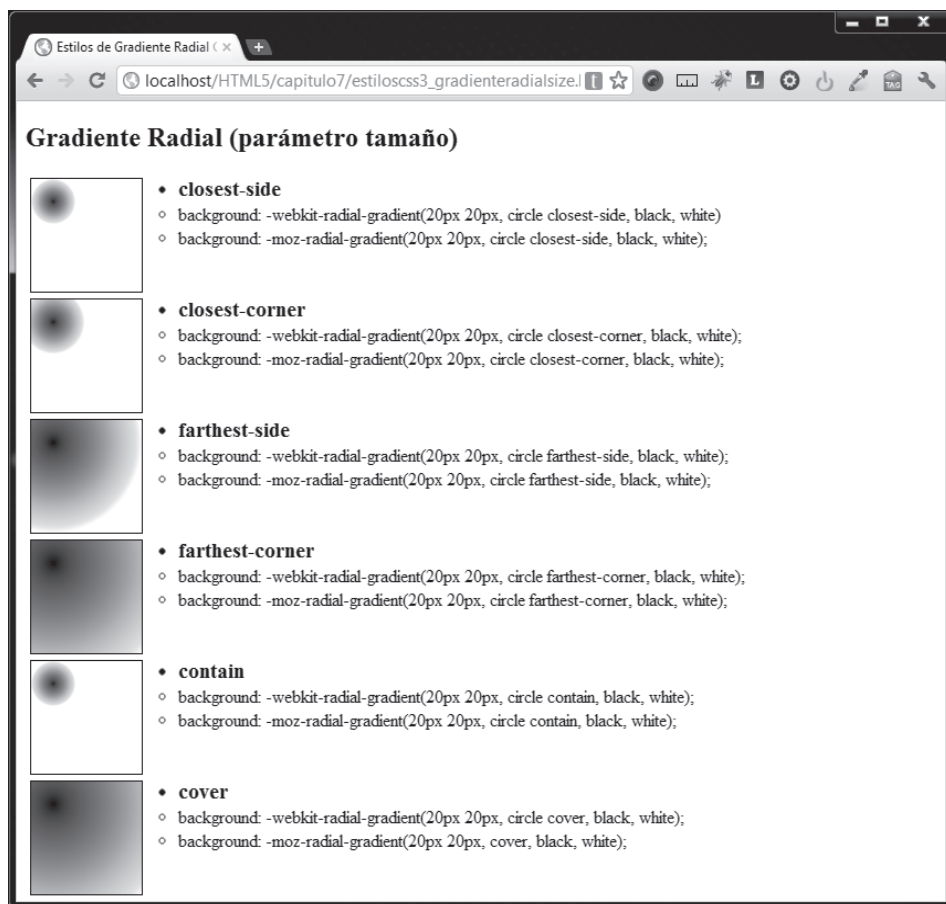


Fig. 7.5 Implementación del parámetro **tamaño** para gradiente radial.

- **color-ini.** El color con el que inicia el gradiente desde el centro.
- **stop-ini.** Hasta qué punto se debe comenzar el efecto de desvanecimiento del color inicial (%).
- **color -N.** Hasta qué punto se debe comenzar el efecto de desvanecimiento del color inicial (%).

- **stop-N.** Desde qué punto se debe detener el efecto de desvanecimiento del color actual para continuar con color sólido (%).

En la Fig. 7.6 se muestra una serie de ejemplos para generar el efecto de gradiente radial.

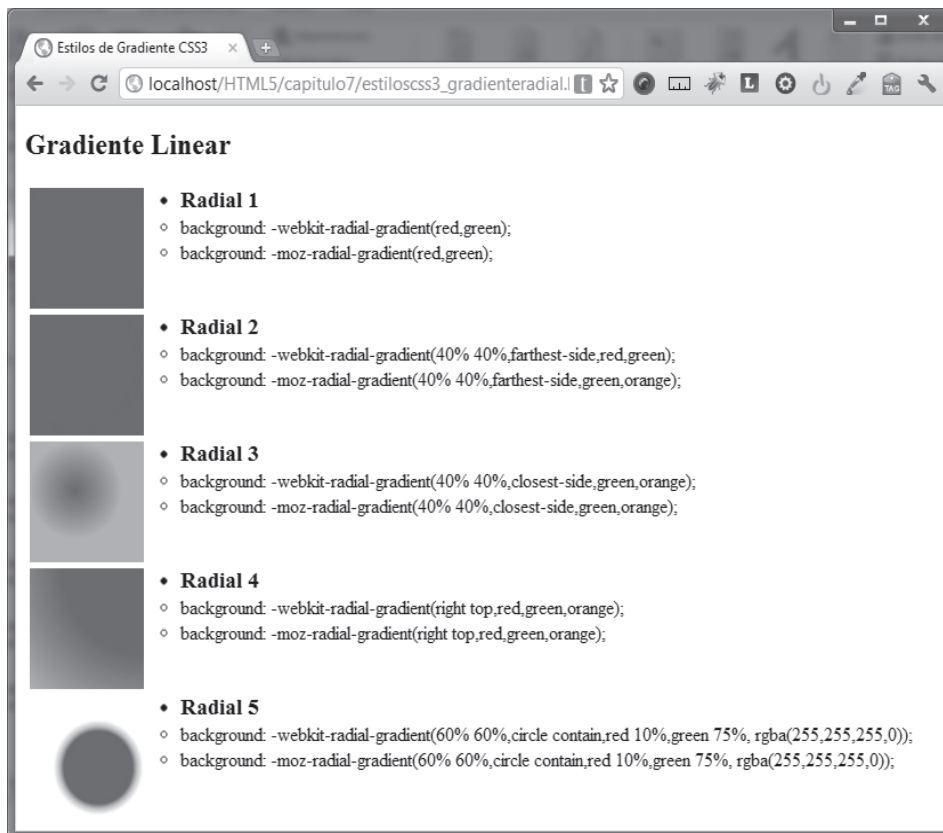


Fig. 7.6 Ejemplos de gradiente radial.



En blanco y negro es posible que el efecto no se aprecie completamente, por lo que para visualizar mejor puede descargar el archivo [estiloscss3_gradienteradial.html](#)

Actividades para el lector

Elabore un documento HTML5 en el utilice tres elementos `<div>`, asigne bordes y fondos con gradiente diferente utilice estilos CSS3. Su documento deber soportar los WebKit y Mozilla engines.

7.5 Transformaciones

El efecto de una transformación en CSS3 consiste en aplicar un cambio de apariencia geométrica a cualquier elemento. Esto proporciona una notable capacidad de control visual que anteriormente no era posible en lo que respecta a HTML o a las hojas de estilo.

El atributo utilizado para crear transformaciones matemáticas sobre un elemento contenedor `<div>` es `transform`. Cuando desea hacer uso de `transform` requiere de proporcionar uno o más de los siguientes parámetros (si usa más de uno, sólo hay que separarlos con espacios):

- **Atributo** `rotate`. Rota un elemento alrededor de su centro, su único parámetro requiere el ángulo de rotación, por ejemplo, 45 grados es `45deg`.
- **Atributo** `scale`. Cambia el tamaño del elemento a lo largo y a lo ancho, el valor que se asigne es un número que indica cuántas veces más grande será la escala. Existen dos atributos que son una variación alternativa de `scale` para escalar un elemento solo en un eje, `scalex` y `scaley` sólo afectarán al eje `x` o `y` individualmente.
- **Atributo** `skew`. Permite sesgar un elemento a lo largo de sus ejes. Utiliza el parámetro `X` y el parámetro `Y` utilizando grados para indicar el ángulo de sesgo sobre los ejes. Las variaciones `skewx` y `skewy` de este atributo se utilizan para usar un sesgo en ejes individuales.
- **Atributo** `translate`. Mueve un elemento de su posición inicial. Utiliza los parámetros `X` e `Y` utilizando las unidades de medida estándares de CSS.

Para ilustrar lo planteado observe el siguiente código HTML:

```
<div id="caja1">Caja 1 (sin transform)</div>
<div id="caja2">Caja 2</div>
<div id="caja3">Caja 3</div>
<div id="caja4">Caja 4</div>
<div id="caja5">Caja 5</div>
```

Observe que el código HTML contiene varios elementos `<div>` que son esencialmente iguales. En el código CSS puede establecer una presentación común para algunas propiedades como sigue:

```
div {
width: 150px;
height: 100px;
float: left;
margin-right: 2em;
border: 3px solid;
}
```

Ahora los estilos para las diferencias que interesan, que en este caso son las transformaciones:

```
#caja2{
float: left;
border-color: red;
-webkit-transform: rotate(45deg);
-moz-transform: rotate(45deg);
-o-transform: rotate(45deg);
-ms-transform: rotate(45deg);
}
#caja3{
float: left;
border-color: green;
-webkit-transform: scale(2) translatey(100px);
-moz-transform: scale(2) translatey(100px);
-o-transform: scale(2) translatey(100px);
-ms-transform: scale(2) translatey(100px);
}
#caja4{
float: left;
border-color: blue;
-webkit-transform: skew(7deg);
-moz-transform: skew(7deg);
-o-transform: skew(7deg);
-ms-transform: skew(7deg);
}
#caja5{
float: left;
border-color: black;
-webkit-transform: translate(25px,50px);
-moz-transform: translate(25px,50px);
-o-transform: translate(25px,50px);
-ms-transform: translate(25px,50px);
}
```

En la Fig. 7.7. Puede ver cómo luce el resultado que lanza este código en el navegador.

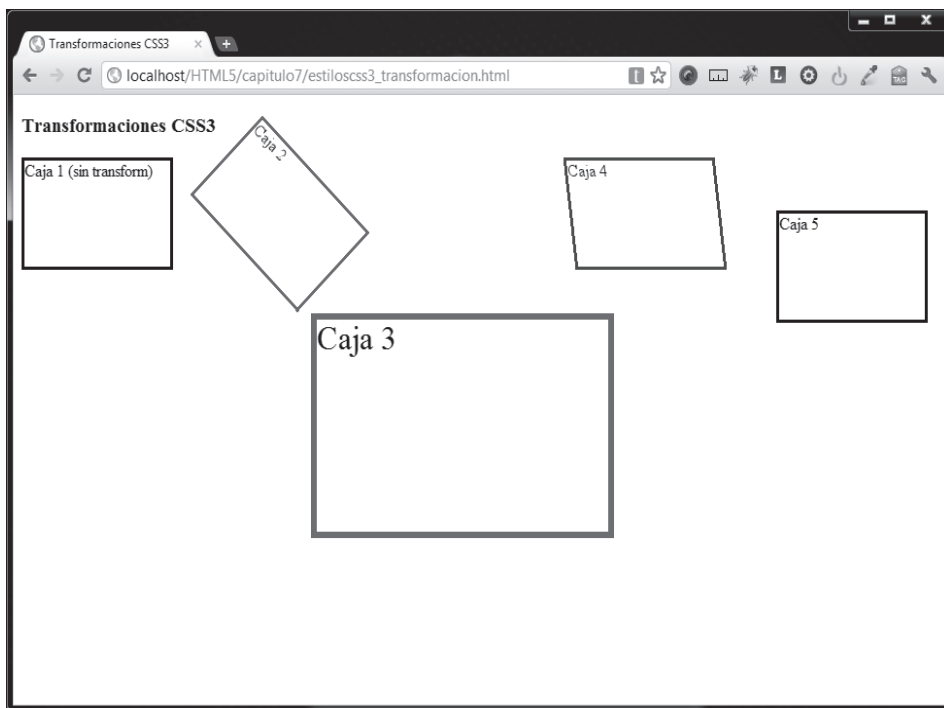


Fig. 7.7 Ejemplos de transformaciones con CSS3.



Para comprobar el código en el navegador puede acudir a los apoyos del libro y descargar el archivo *estiloscss3_transformacion.html*

7.6 Transiciones

Mientras que las transformaciones son una herramienta fuerte e interesante para el desarrollador, la posibilidad de animar los mismos efectos es algo aún más emocionante, es en este momento donde la propiedad `transition` entra en acción.

Si ha probado en el navegador el ejemplo del punto anterior, habrá notado que al momento de cargar la página los efectos ya han sido aplicados, es decir, el usuario sólo ve el resultado final de la transformación. La propiedad `transition` permite ver el cambio “al vuelo” cuando el usuario dispara algún evento.

El mecanismo es simple, observe el ejemplo para dar una transición sencilla a un elemento `<p>`:

<p>Este texto hace una transición de color</p>

El código CSS es claro en su intención:

```
p {  
  color: black;  
  font-size: 200%;  
  -webkit-transition: color 1s ease-in;  
  -moz-transition: color 1s ease-in;  
  -o-transition: color 1s ease-in;  
}  
  
p:hover {  
  color: yellow;  
}
```

Apoyo en la



En los materiales adicionales encontrará el archivo *estiloscss3_transicion.html*

Más allá de la propiedad `transition` lo que sucede es breve y simple. El estado normal del párrafo sólo indica que el color del texto dentro es negro y su tamaño se incrementa a un 200%. Se define lo que debe hacer el párrafo cuando el puntero del mouse se coloca encima, utilizando la pseudoclase `hover`, y se indica que cambie su color a amarillo, cuando el mouse deja de estar encima del párrafo el texto que contiene regresa a negro. Cuando se agrega la propiedad `transition` lo único que sucede es que el cambio de color del texto no es instantáneo, sino que el cambio de negro a amarillo y de amarillo a negro es gradual.

La sintaxis de `transition` para Chrome y Safari:

```
-webkit-transition: tipo-animación [duración] [sincronización]  
[retraso]
```

La sintaxis de `transition` para Firefox:

```
-moz-transition: tipo-animación [duración] [sincronización]  
[retraso]
```

La sintaxis de `transition` para Opera:

```
-o-transition: tipo-animación [duración] [sincronización] [retraso]
```

Nota: Internet Explorer aún no soporta transiciones (25/10/2011).

- **Atributo tipo-animación.** Es el tipo de animación que se utilizará. El valor por default es *all*, pero hay varios otros valores que se pueden utilizar como *color*, *length*, *width*, *percentage*, *opacity* y *number*. Esto ha cambiado un poco, en el ir y venir de la especificación, por lo que ante la duda pruebe con *all*.
- **Atributo duración.** La duración de la animación en segundos, por ejemplo 3 segundos es 3s.
- **Atributo sincronización.** Para lograr que la animación ocurra a velocidad constante utilice *linear*. Si desea un movimiento más gradual que se acelere o disminuya la velocidad al final de la animación, puede usar los valores *ease*, *ease-in*, *ease-out*, *ease-in-out*.
- **Atributo retraso.** Establece un tiempo de retraso en segundos y la animación no comenzará hasta que este tiempo transcurra.

La animación con transiciones es una característica muy nueva en la especificación CSS3, por lo que algunos elementos pueden ser animados de esta manera y otros no, esto sumado a la fase de experimentación de los navegadores lo lleva a usted a tener que hacer experimentación para determinar si el elemento es viable o no de usar el atributo *transition*.

Nota

Existen propiedades individuales de transición para controlar las diferentes partes de una animación, pero las más utilizadas son como se ha descrito en este punto.

Ahora observe y estudie cómo se comportan las cajas del ejemplo del punto anterior (7.5 Transformaciones) pero agregando transiciones, la fracción de código CSS ahora luce de esta manera:

```
#caja2{
    float: left;
    border-color: red;
    -webkit-transition: all 1s ease-in;
    -moz-transition: all 1s ease-in;
    -o-transition: all 1s ease-in;
    -ms-transition: all 1s ease-in;
}
#caja2:hover{
    -webkit-transform: rotate(45deg);
```

```
-moz-transform: rotate(45deg);
-o-transform: rotate(45deg);
-ms-transform: rotate(45deg);
}
#caja3{
  float: left;
  border-color: green;
  -webkit-transition: all 1s ease-in;
  -moz-transition: all 1s ease-in;
  -o-transition: all 1s ease-in;
  -ms-transition: all 1s ease-in;
}

#caja3:hover{
  -webkit-transform: scale(2) translatey(100px);
  -moz-transform: scale(2) translatey(100px);
  -o-transform: scale(2) translatey(100px);
  -ms-transform: scale(2) translatey(100px);
}
#caja4{
  float: left;
  border-color: blue;
  -webkit-transition: all 1s ease-in;
  -moz-transition: all 1s ease-in;
  -o-transition: all 1s ease-in;
  -ms-transition: all 1s ease-in;
}

#caja4:hover{
  -webkit-transform: skew(7deg);
  -moz-transform: skew(7deg);
  -o-transform: skew(7deg);
  -ms-transform: skew(7deg);
}

#caja5{
  float: left;
  border-color: black;
  -webkit-transition: all 1s ease-in;
  -moz-transition: all 1s ease-in;
  -o-transition: all 1s ease-in;
```

```

    -ms-transition: all 1s ease-in;
}
#caja5:hover{
    -webkit-transform: translate(25px,50px);
    -moz-transform: translate(25px,50px);
    -o-transform: translate(25px,50px);
    -ms-transform: translate(25px,50px);
}

```

Apoyo en la



En los materiales adicionales encontrará el archivo *estiloscss3_transicion.html*

7.7 Transparencia

El efecto de transparencia es algo que se puede controlar en CSS3 con el uso de la propiedad `opacity`. Sólo usa un argumento que puede tener un valor numérico de 0 (total transparencia) a 1 (total opacidad).

El nivel de opacidad se puede utilizar sobre prácticamente cualquier elemento, como pueden ser imágenes, textos, secciones completas dentro de un elemento `<div>`, etcétera.

La Fig. 7.8 muestra imágenes dentro de un `<div>` con diferentes grados de opacidad (o transparencia) en cada uno de ellos.

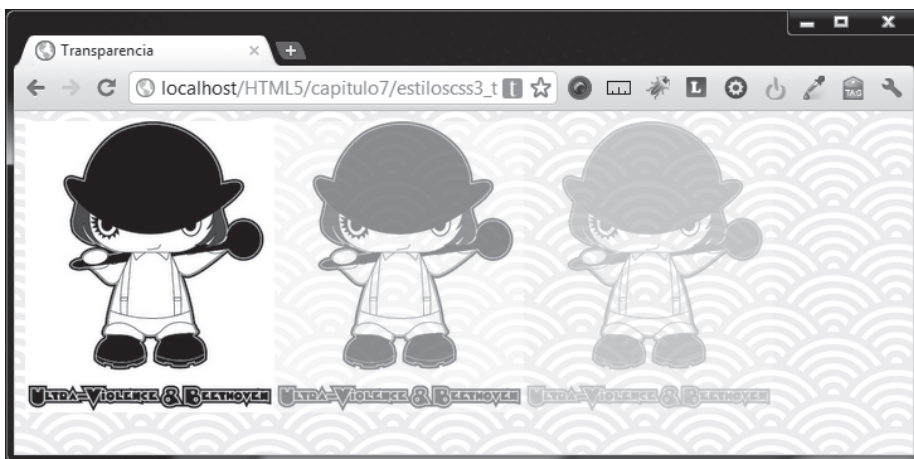


Fig. 7.8 Ejemplo de transparencia con CSS3.

El código es de muy fácil implementación, en la parte HTML implica sólo unos `<div>` que utilizan por dentro a la etiqueta ``:

```
<div id="img1"></div>
<div id="img2"></div>
<div id="img3"></div>
```

El código CSS3 establece una imagen de fondo, y enseguida se procede a aplicar estilo a cada <div> utilizando el atributo `opacity`:

```
body{
    background: url(bg_texture.gif);
}

div {
    float: left;
}
#img1{
    opacity: 1;
}
#img2{
    opacity: .5;
}
#img3{
    opacity: .2;
}
```

Apoyo en la



En los materiales adicionales encontrará el archivo
estiloscss3_transparencia.html

7.8 Texto y fuentes descargables

El formato de texto y la fuente que se utiliza siempre ha sido una limitante de las páginas basadas en HTML, ya que en muchas ocasiones los desarrolladores se ven en situaciones donde diseñadores gráficos, clientes o personas ajenas al desarrollo Web desean que un texto luzca de determinada manera, ocasionando que los desarrolladores, dependiendo del escenario, se vean en la necesidad de recurrir a imágenes, alternativas propietarias como Flash o dar un rotundo no y explicar las razones.

CSS3 en su nueva especificación trae ayuda a las situaciones donde se requiere algún efecto sobre texto e incluso cuando es necesario alejarse de las fuentes Web seguras, para esto se proponen nuevas alternativas.

7.8.1 text-shadow

Se recurre mucho a las sombras debido a que su efecto agrega una sensación de profundidad, que bien utilizado es ventajoso y atractivo visualmente. Para lograr este efecto se usa la propiedad `text-shadow`, la cual ya existía en CSS2, pero es hasta hoy donde ha encontrado soporte por parte de los navegadores mayores.

Su sintaxis es:

```
text-shadow: sombra-h sombra-v difuminado color;
```

- **Atributo** `sombra-h`. Parámetro obligatorio que establece la posición horizontal de la sombra.
- **Atributo** `sombra-v`. Parámetro obligatorio que establece la posición vertical de la sombra.
- **Atributo** `difuminado`. La distancia del efecto de difuminado de la sombra.
- **Atributo** `color`. El color de la sombra. Utiliza cualquiera de las notaciones de colores permitidas por CSS3.

En la Fig. 7.9 puede ver la imagen de una página que muestra la implementación de `text-shadow`.

7.8.2 text-stroke

Otra propiedad muy práctica, sobre todo en casos en donde el texto se confunde con el color de fondo en una página, ésta es `text-stroke`. La propiedad define una línea de borde alrededor de cada carácter de un texto. Sólo emplea dos parámetros, uno para el grueso de la línea y otro para el color.

Un ejemplo de esto si deseara usar `text-stroke` en un encabezado:

```
h1{  
  color: black;  
  -webkit-text-stroke: 2px orange;  
}
```

Esta propiedad, por el momento, sólo es soportada por Chrome y Safari. En la Fig. 7.9 puede ver la imagen de una página que implementa este código.

7.8.3 @font-face

Anteriormente los desarrolladores sufrían con el tema de las fuentes, ya que siempre que alguna persona no involucrada en el desarrollo Web solicitaba el uso de algún tipo de letra exótico, había que hacer algún esfuerzo de concientización para explicar por qué no era lo más conveniente o buscar alternativas poco ideales, como el uso de imágenes o tecnologías alternativas. El estilo `font-face` ayuda mucho en este escenario, ya que cuando el navegador carga una página, también descarga la fuente indicada, desplegando el tipo de letra correcto.

La particularidad de `font-face` es que no trabaja de la misma manera que la mayoría de los estilos CSS. En lugar de definir un estilo que afecta a una etiqueta HTML, `font-face` define un nuevo estilo que se puede utilizar en otra declaración de hojas de estilo:

```
@font-face {  
    font-family: Berenika;  
    src: url("Berenika-Regular.ttf");  
}
```

En el código anterior `font-face` ha definido el atributo `font-family` con el nombre con el que podrá ser utilizado en el resto del código CSS. Para mayor claridad en su código, no es mala idea usar un nombre igual o parecido al nombre del archivo origen de la fuente.

```
h3{  
    font-family: Berenika;  
}
```

En la actualidad `font-face` es soportada por los grandes navegadores, pero los tipos de formato para las fuentes no son universalmente soportados por todos ellos, por lo que la experimentación es algo requerido. Éstos son los formatos que cuentan con mayor soporte actualmente:

- **TTF.** El estándar TrueType es ampliamente soportado, pero no por todos los navegadores. Muchas fuentes de estándar abierto usan este formato.
- **OTF.** El estándar OpenType es un estándar realmente abierto. Todos los navegadores, excepto Internet Explorer los soportan.
- **WOFF.** Web Open Font Format es soportado por las últimas versiones de todos los navegadores mayores.
- **EOT.** Embedded OpenType Format es un formato propietario de Microsoft que sólo trabaja en Internet Explorer.

Nota

Hay ocasiones en que algunas fuentes tienen problemas en algunos navegadores, haga diversas pruebas con éstos y la fuentes de su interés.

El estilo `@font-face` permite descargar la fuente que está utilizando, pero tenga en cuenta que esto significa implícitamente que usted está distribuyendo una fuente que posiblemente sea propietaria, por lo que como desarrollador debe ser respetuoso de la propiedad intelectual.

La Fig. 7.9 muestra una página con ejemplos para las propiedades `text-shadow`, `text-stroke` y el estilo `@font-face`.



Fig. 7.9 Ejemplo de texto y fuente descargable con CSS3.

Apoyo en la



En los materiales adicionales encontrará los archivos `estiloscss3_transparencia.html` y `Berenika-Regular.ttf`

7.9 Herramientas de selección

A lo largo del uso de CSS es evidente que uno de los aspectos más poderosos y utilizado es la capacidad de selección de grupos de elementos usando el nombre de una clase, el tipo de etiqueta o individualmente con un identificador. Pero CSS3 proporciona nuevos métodos de selección conocidos en inglés como *selectors*.

Hay varios métodos de selección o *selectors*, algunos no en su forma definitiva, ahora verá algunos de los mejor consolidados y de mayor uso.

7.9.1 Selección por atributo

Usted puede aplicar un estilo a cualquier elemento que contenga un atributo con un valor específico. Por ejemplo, imagine que de una etiqueta `<input>` desea dar

estilo solamente a los campos de texto, si recuerda usted bien, en este caso el atributo `type`, sería igual a `text`, por lo tanto se podría emplear el siguiente código:

```
input[type="text"]{
    border-color: #ff0000
}
```



En los materiales adicionales encontrará el archivo `estiloscss3_selector_atributo.html`

Existen variaciones interesantes para seleccionar por atributo en diferentes situaciones:

SELECTOR	EJEMPLO	DESCRIPCIÓN
<code>[atributo^=value]</code>	<code>a[src^="https"]</code>	Selecciona todos los enlaces (<code><a></code>) donde el valor de su atributo <code>src</code> comienza con la cadena de texto <i>https</i> .
<code>[atributo\$=value]</code>	<code>a[src\$=".pdf"]</code>	Selecciona todos los enlaces (<code><a></code>) donde el valor de su atributo <code>src</code> termina con la cadena de texto <i>.pdf</i> .
<code>[attribute*=value]</code>	<code>a[src*="alfaomega"]</code>	Selecciona todos los enlaces (<code><a></code>) donde el valor de su atributo <code>src</code> contiene la cadena de texto <i>alfaomega</i> .

7.9.2 Selección por relación padre-hijo

Al construir una página, probablemente se ha percatado de la relación jerárquica o de árbol familiar que tienen los elementos HTML, es decir, todos los elementos en la estructura visual de una página están agrupados dentro de la etiqueta `<body>...</body>`, por lo tanto, se podría decir que `<body>` es la etiqueta padre de muchos otros elementos, como enlaces, párrafos, listas etc., continuando con esa lógica, no sería erróneo afirmar que si un enlace estuviera acompañando al texto dentro de un párrafo, entonces `<a>` es hijo de `<p>`.

Explotando esa relación de padre-hijo que tienen los elementos HTML, hoy CSS3 permite hacer selecciones muy interesantes, por ejemplo, usted tiene varios párrafos con varios elementos dentro, como etiquetas, imágenes, enlaces, etc.,

usted sabe que siempre tiene etiquetas `<label>` como primer elemento en los párrafos y quiere darles un color rojo, pero no desea modificar directamente a las etiquetas por su tipo, ya que no quiere afectar al resto de las etiquetas `<label>` que no están dentro de un párrafo. Una posible solución a esta situación es sólo seleccionar al primer elemento, dentro de cada párrafo utilizando `nth-child`:

```
p>label:nth-child(1){
    color: red;
}
```



En los materiales adicionales encontrará el archivo `estiloscss3_selector_nth.html`

El código utiliza el *selector* `nth-child` con su único parámetro que describe la posición del elemento que desea seleccionar. Existen aún más herramientas de selección para sacar provecho de la relación padre-hijo:

SELECTOR	EJEMPLO	DESCRIPCIÓN
<code>:nth-last-child(n)</code>	<code>label:nth-last-child(2)</code>	Selecciona todos los encabezados <code><label></code> que sean el último hijo de su respectivo padre, a partir del segundo hijo.
<code>:nth-of-type(n)</code>	<code>label:nth-of-type(2)</code>	Selecciona cada elemento que sea de tipo <code><label></code> que sea el segundo hijo de su respectivo padre.
<code>:nth-last-of-type(n)</code>	<code>label:nth-last-of-type(2)</code>	Selecciona el último elemento de su respectivo padre que sea de tipo <code><label></code> a partir del segundo hijo.

7.9.3 Selección inversa (not)

En ocasiones es muy rápido y útil hacer una selección inversa. Suponga una página donde tiene múltiples párrafos, la mitad de los párrafos pertenece a la clase *importante* y el resto pertenece a otras clases o a ninguna. Si quisiera seleccionar todos aquellos párrafos que no pertenecen a la clase *importante*, podría emplear el siguiente código:

```
p:not(.importante) {  
    border: 2px solid green;  
}
```

Apoyo en la



En los materiales adicionales encontrará el archivo *estiloscss3_selector_not.html*

El código presentado anteriormente coloca un borde color verde con 2 pixeles de ancho en todos aquellos párrafos que no pertenecen a la clase especial.

Nota

Existen varias herramientas de selección adicionales, algunas existentes desde CSS1 y CSS2, que escapan del objetivo de este libro, véase la lista de enlaces al final del capítulo para conocer más.

Actividades para el lector

Construya una página con una lista donde inserte:

- Un elemento HTML que haga un tipo de transición.
- Otro elemento HTML que haga una transformación.

Como complemento de su página redacte e inserte un texto con sombra, un borde de 3 pixeles en color azul

RESUMEN

En este capítulo se ha conocido la nueva especificación CSS3:

- Sus nuevas capacidades.
- Las mejoras a las capacidades ya existentes.
- Las limitantes entre CSS3 y los navegadores mayores.
- El uso de nuevos elementos visuales, efectos en texto y fuentes descargables.
- Introducción a las nuevas herramientas de selección.

Autoevaluación

1. ¿Qué es CSS3?
2. ¿Qué son los motores de renderizado?
3. ¿Qué tipo de bordes se pueden hacer con CSS3?
4. ¿Cuál es la diferencia entre transformación y transición en términos de CSS3?
5. ¿Cuál es el estilo que permite descargar una fuente que no tiene el usuario?

EVIDENCIA



Construyó una página con una lista, insertó en ella al menos un elemento que realiza transiciones.



En la misma lista insertó otro elemento que hace transformaciones.



Complementó su página con un texto con efecto de sombra y un borde de 3 píxeles en color azul.

REFERENCIAS

Bibliografía

Hogan, Brian (2011). HTML5 and CSS3. Develop with Tomorrow's Standards Today, 1a. ed., Pragmatic Bookshelf, EUA.

Gillenwater, Zoe (2011). Stunning CSS3, 1a. ed., New Riders, EUA.



Páginas Web recomendadas

<http://www.w3schools.com/cssref/default.asp>

<http://www.impressivewebs.com/>

<http://css-tricks.com/>

<http://www.w3.org/TR/css3-selectors/>

Respuestas sugeridas a las preguntas de autoevaluación

1. Es la nueva versión del lenguaje de hojas de estilo en cascada.
2. Es el código de programación que forma el motor que permite a los navegadores desplegar los elementos HTML.
3. Con bordes redondeados, con sombra y con imagen.
4. Las transformaciones afectan la forma, color u orientación del elemento, mientras que la transición cambia su posición.
5. El estilo @font-face.

Novedades en JavaScript

8

Reflexione y responda las preguntas siguientes:

- ¿Qué es lo nuevo en JavaScript para HTML5?
- ¿Cuáles son las nuevas herramientas para seleccionar elementos HTML con JavaScript?
- ¿Cuáles son y cómo se utilizan las nuevas herramientas de selección de JavaScript?
- ¿Qué es el API de almacenamiento Web y en qué consiste?
- ¿Qué es el API Web offline de HTML5 y en qué consiste?
- ¿Qué es el API de geolocalización de HTML5 y en qué consiste?
- ¿Qué es el API WebSocket de HTML5 y en qué consiste?
- ¿Qué es el API Web worker de HTML5 y en qué consiste?

Contenido

Novedades en JavaScript

Introducción

8.1 Nuevas herramientas de selección

8.2 Almacenamiento de sesión y local

8.3 Web offline

8.4 Geolocalización

8.5 WebSocket

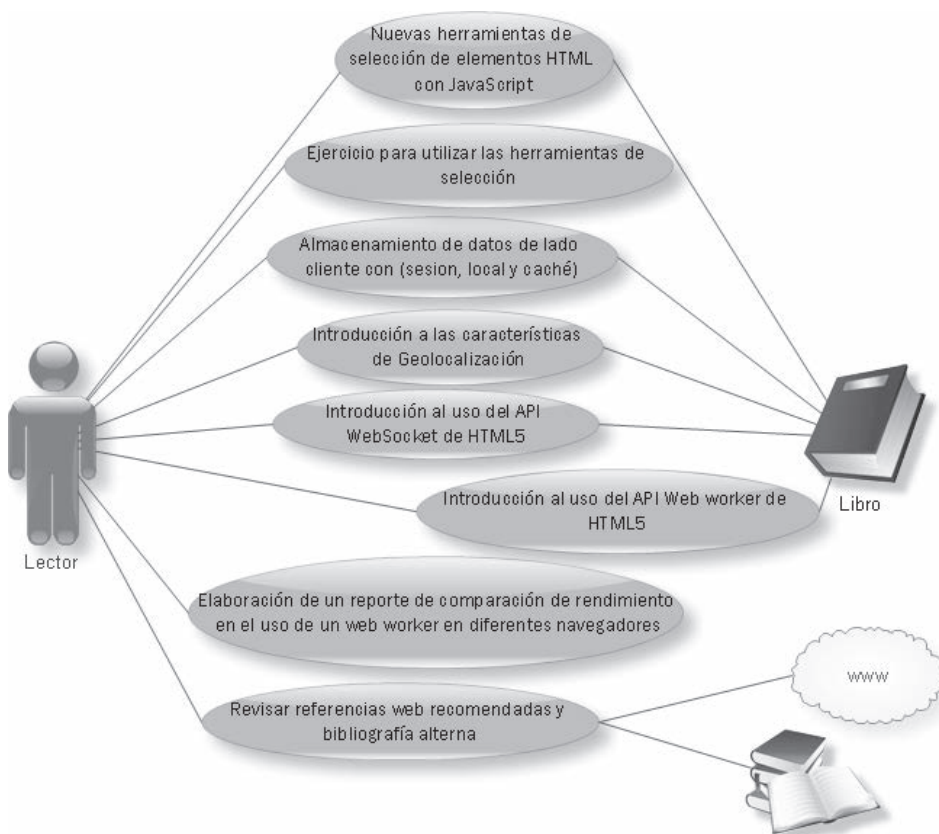
8.6 Web workers

Expectativa

Conocer las nuevas características disponibles en JavaScript para su uso en la creación de documentos HTML5.

Después de estudiar este capítulo, el lector será capaz de:

- Conocer las generalidades de las nuevas características en JavaScript.
- Comprender las nuevas opciones de selección de elementos HTML en JavaScript.
- Conocer las nuevas capacidades del almacenamiento de información del lado cliente de HTML5.
- Comprender la nueva característica de geolocalización de HTML5.
- Comprender las bases de la programación con WebSockets.
- Usar la nueva programación por hilos con Web workers.



INTRODUCCIÓN

JavaScript es actualmente una tecnología vital en lo que al desarrollo de aplicaciones modernas para la Web se refiere. Todos los navegadores se han estado esforzando en dar un mejor soporte y rendimiento para JavaScript, adicionalmente otros grupos han creado extensiones muy interesantes para esta tecnología

Todos los desarrolladores Web hoy en día estarán de acuerdo con las anteriores afirmaciones, pero lo que hace a JavaScript aún más importante es que está acompañando en su evolución a HTML5, lo que hace de la Web ya no sólo un asunto de alojar páginas para después mostrarlas, sino la convierte en una verdadera plataforma de desarrollo para aplicaciones.

Este capítulo habla principalmente de los cambios y novedades que se han hecho a JavaScript. Para poder hablar en el contexto de lo nuevo en JavaScript se asume que usted está familiarizado con el lenguaje y con la programación en general. Si usted recién comienza con JavaScript, por favor consulte una bibliografía especializada antes de abordar este capítulo.

8.1 Nuevas herramientas de selección

Cuando se programa con JavaScript en documento HTML constantemente se utilizan los métodos `getElementById()` y `getElementsByTagName()` para crear variables basadas en un elemento HTML. Siendo esto posible, quizá usted en algún momento (al igual que el autor de este libro) se ha preguntado ¿por qué no existe un `getElementsByClassName()`?, bueno pues ahora no sólo existe, sino que viene con más.

8.1.1 `getElementsByClassName()`

El nuevo método `getElementsByClassName()` permite ahora seleccionar varios elementos que pertenecen a la misma clase y los devuelve en un arreglo, al cual puede usted hacer referencia elemento a elemento. Considere el siguiente ejemplo:

```
function detectar(){
    elemento = document.getElementsByClassName("prueba");
    for (i = 0; i < elemento.length; i++){
        alert(elemento[i].innerHTML);
    }
}
```

Observe que el código anterior utiliza un bucle para ir recuperando cada elemento del arreglo y lanzar un mensaje con el contenido de cada uno de ellos.

Apoyo en la



En los materiales adicionales encontrará el archivo:
javascript_selectorclass.html

8.1.2 getElementByTagName

El método `getElementByTagName()` permite seleccionar de una sola vez a todos los elementos de un solo tipo de etiqueta. Usted podría obtener, por ejemplo, a todos los párrafos de un documento. De la misma forma que hace `getElementByClassName()` el resultado devuelve un arreglo, que se puede recorrer con un bucle y manejado como mejor convenga. Observe el siguiente código:

```
function detectar(){
    elemento = document. getElementByTagName("p");
    for (i = 0; i < elemento.length; i++){
        alert(elemento[i].innerHTML);
    }
}
```

Apoyo en la



En los materiales adicionales encontrará el archivo:
javascript_selectortag.html

8.1.3 querySelector()

Usando la misma sintaxis que utiliza CSS para seleccionar elementos DOM, el método `querySelector()` permite seleccionarlos también. Algunas librerías como JQuery proporcionaban la habilidad de hacer esto desde hace algún tiempo, pero ahora es parte nativa de JavaScript. El siguiente código selecciona el primer párrafo precedido por una etiqueta `<div>` que encuentre.

```
function detectar(){
    elemento = document.querySelector("div + p");
    alert(elemento.innerHTML);
}
```

El código anterior únicamente selecciona un solo elemento, si desea encontrar a todos los que cumplan con el criterio elegido, entonces deberá usar el método `querySelectorAll()`, el cual se aborda a continuación.



En los materiales adicionales encontrará el archivo:
javascript_selectorquery.html

8.1.4 querySelectorAll()

Utilizando exactamente el mismo mecanismo que `querySelector()`, el método `querySelectorAll()` permite seleccionar todos los elementos que coincidan con el criterio proporcionado. El siguiente código selecciona todos los párrafos precedidos por una etiqueta `<div>`.

```
function detectar(){
    elemento = document.querySelectorAll("div + p");
    for (i = 0; i < elemento.length; i++){
        alert(elemento[i].innerHTML);
    }
}
```

En la función en el código anterior `querySelectorAll()` coloca los elementos obtenidos en un arreglo que después se recorren uno a uno con un bucle `for`.



En los materiales adicionales encontrará el archivo:
javascript_selectorqueryall.html

Actividades para el lector

Elabore una página en la que utilice las herramientas de selección `getElementsByClassName`, `getElementsByTagName`, `querySelector`, `querySelectorAll` para dar diversos estilos a los elementos seleccionados.

8.2 Almacenamiento de sesión y local

Actualmente la Web se ha convertido y se consolida como una plataforma de desarrollo, por lo cual el almacenamiento de datos se convertido en un tema de importancia. Las aplicaciones cliente siempre han estado limitadas para almacenar datos del lado cliente con el objetivo de prevenir las malas intenciones de algún posible código malicioso, esto es algo bueno, pero al mismo tiempo complica el desarrollo de aplicaciones en los navegadores.

Debido a las cuestiones de seguridad, y a otras causas, los desarrolladores utilizan tecnologías de acceso a datos del lado servidor (como las variables de sesión de ASP o PHP, entre otras), pero todos estos mecanismos se reducen al

almacenamiento de *cookies* en el navegador, las *cookies* son útiles, pero su capacidad es muy reducida debido a que pueden almacenar muy poca información (alrededor de 4k), y son ineficientes, ya que se pueden refrescar, y dado que se envían en cada petición HTTP resulta que la misma información se envía una y otra vez haciendo más lento el rendimiento de una aplicación. HTML5 proporciona algo nuevo para cubrir estos problemas, el API de almacenamiento Web, compuesto por dos objetos básicos para guardar información del lado cliente, el almacenamiento por sesión y el almacenamiento local.

8.2.1 Almacenamiento por sesión

Uno de los problema más comunes en el desarrollo Web cuando se usa el enfoque de las *cookies*, es controlar su consistencia a lo largo de una aplicación, por ejemplo, cuando el mismo usuario, en el mismo navegador pero en diferentes ventanas o pestañas hace operaciones diferentes pero comparte las mismas *cookies*, puede ocasionar errores de consistencia en ellas, es decir, si no se tiene cuidado con esto, podría ocasionar errores como comprar un boleto para un concierto 2 veces sin darse cuenta.

Para resolver situaciones como la descrita anteriormente puede utilizar `sessionStorage`, ya que los objetos instanciados por este objeto sólo persistirán hasta que la ventana o pestaña del navegador donde se inicializaron sea cerrada.

En el punto 8.2.3 “Asignando y recuperando datos”, se estudiará cómo inicializar el objeto y asignar información que se pueda utilizar.

8.2.2 Almacenamiento local

Otra manera que se tiene para almacenar información del lado cliente es utilizando el objeto `localStorage`, el cual, a diferencia de `sessionStorage` (y por supuesto de la *cookies*) tiene la característica de que la información que guardan persiste más allá de la vida de una simple ventana o una pestaña, además de que es accesible desde diferentes instancias del navegador. En estos casos es cuando es mejor utilizar `localStorage`.

8.2.3 Asignando y recuperando datos de `sessionStorage` y `localStorage`

Como se observa en los dos puntos anteriores, la diferencia básica entre `sessionStorage` y `localStorage` es que la duración de los datos del primero no dependen de la vida de la ventana o pestaña del navegador, pero ambos objetos trabajan con el mismo mecanismo, la única diferencia es el nombre del objeto, pero la sintaxis es idéntica y sus métodos también.

Asignar un valor se logra fácilmente para `sessionStorage` con la siguiente sintaxis:

```
sessionStorage.setItem(miIdentificador, miValor);
```

Es Igualmente fácil para `localStorage` con la sintaxis:

```
localStorage.setItem(miIdentificador, miValor);
```

El método al que se llama `setItem` utiliza dos parámetros en formato de cadena de caracteres, esto es importante porque aunque la especificación dice que soporta otro tipo de datos, los navegadores solamente pasan cadenas de caracteres actualmente.

El primer parámetro *miIdentificador* que utiliza `setItem` será lo que sirva más tarde como un nombre clave para recuperar el valor que ha colocado en el parámetro *miValor*. El método que se utiliza para recuperar los valores guardados es `getItem` y funciona de igual manera para `sessionStorage` y `localStorage`.

Sintaxis para `sessionStorage`:

```
sessionStorage.getItem(miIdentificador);
```

Misma sintaxis para `localStorage`:

```
localStorage.getItem(miIdentificador);
```

Ahora que ya conoce cómo asignar un valor con `setItem`, almacene con `sessionStorage` un nombre, para ello utilice la siguiente función:

```
function detectar(){
    elnombre = sessionStorage.getItem("nombre");
    if (elnombre == "null" || elnombre == null){
        alert("Nadie ha estado aquí");
        sessionStorage.setItem("nombre", "Juan");
    }else{
        alert(elnombre + " ya ha estado aquí");
    }
}
```

Lo que hace esta función es tratar de obtener el valor señalado como nombre del objeto `sessionStorage` con el método `getItem`, después verifica si el valor existe o no, si no existe lanza el mensaje “Nadie ha estado aquí” y coloca el valor “Juan” con `setItem`, pero si lo encuentra entonces lanzará el mensaje “Juan ya ha estado aquí”. El único momento en que el mensaje “Nadie ha estado aquí” aparece, es la primera vez que cargue la página en una ventana o pestaña nueva, el resto de las veces el mensaje será “Juan ya ha estado aquí”, para regresar al estado original tiene que cerrar la pestaña o la ventana y entonces volver a cargar la página.

Ahora observe cómo sería este mismo ejemplo pero empleando `localStorage`, el mecanismo es prácticamente el mismo pero se emplea una función adicional:

```
function detectar(){
    elapellido = localStorage.getItem("apellido");
    if (elapellido == "null" || elapellido == null){
        alert("Nadie ha estado aquí");
        localStorage.setItem("apellido", "Pérez");
    }else{
        alert("El Sr. " + elapellido + " ya ha estado aquí");
    }
}

function limpiar(){
    elapellido = localStorage.getItem("apellido");
    if (elapellido != "null" & elapellido != null){
        alert("Limpiando " + elapellido )
        localStorage.removeItem("apellido");
    }else{
        alert("Nada que limpiar")
    }
}
```

Si observa la función `detectar()` notará que las únicas diferencia con el ejemplo anterior es el uso del objeto `localStorage` y el uso de un apellido en lugar del nombre (esto último es irrelevante) el mecanismo es igual, sin embargo, la diferencia real radica en que, pasada la primera vez que cargue la página, aunque usted cierre la ventana o la pestaña donde ejecuta esta página, el mensaje “El Sr. Pérez ya ha estado aquí” aparecerá, incluso si abre una nueva ventana de navegador sin importar si cierra o no la primera, este mensaje seguirá apareciendo. La única manera de regresar al estado original es removiendo explícitamente el valor guardado.

Cuando desea borrar un valor específico debe utilizar el método `removeItem()`, remover el valor es justamente el objetivo del código de la función `limpiar()`, la cual es llamada en este caso con un elemento `<button>`. Su funcionamiento es determinar si el valor “apellido” existe, si es así entonces lanza el mensaje “Limpiando Pérez” y elimina el valor, si no hay valor llamado “apellido” entonces el mensaje será “Nada que limpiar”.



En los materiales adicionales encontrará los archivos:
javascript_session_storage.html y *javascript_local_storage.html*

8.2.3 Métodos y propiedades de `sessionStorage` y `localStorage`

El API de almacenamiento de HTML5 potenciado con el uso de JavaScript es una de las cosas más simples y potentes de HTML5.

A continuación se muestra el juego completo de métodos y propiedades que tienen `sessionStorage` y `localStorage`:

- **`setItem(identificador, value)`**. Coloca un valor en almacenamiento identificado con un nombre clave, o reemplaza a un valor si es que éste usa el mismo nombre clave. Tenga en cuenta que es posible recibir un error si el usuario decide apagar la característica de almacenamiento, o si el espacio de almacenamiento ha sido llenado al máximo, si se da alguno de estos casos, un error `QUOTA_EXCEED_ERR` será lanzado. Asegúrese de manejar apropiadamente el caso de este error.
- **`getItem(identificador)`**. Permite recuperar el valor señalado por nombre clave. En caso de que el identificador no esté almacenado (no exista), el método regresará un valor `null`.
- **`removeItem(identificador)`**. Si un valor existe bajo el nombre clave indicado el método lo removerá, si no existe, ninguna acción será efectuada.
- **`clear()`**. Este método remueve todos los valores de una lista de almacenamiento. Considere que una vez borrados todos los elementos en almacenamiento, no existe forma de recuperarlos.
- **`key(posición)`**. Dado un número en *posición* el método traerá al nombre clave que esté en dicha posición. No hay manera de garantizar el orden de los nombres clave, por lo que este método normalmente se usa en un bucle para traer todos los nombres clave.
- **`length`**. Este atributo especifica cuántos pares identificador-valor existen actualmente en el espacio de almacenamiento. Tenga en cuenta que este dato está en el contexto de un solo origen, es decir, no estarán en el mismo conteo elementos que se originan en un sitio y los que se originan en otro.

8.3 Web offline (caché)

Aunque las conexiones a internet están cada vez más a la mano y las personas que viven en los países más desarrollados comienzan a tener la sensación de que pueden estar en línea todo el tiempo y en todo lugar, tristemente la realidad es que esto no es así. Existen muchas regiones del mundo y situaciones donde no es posible estar en línea de forma permanente, o simplemente existen fallas de cualquier tipo que causan intermitencia en su conexión, sobre todo si se piensa en dispositivos móviles

Éstas son las causas que hacen que una aplicación Web pueda mantener el funcionamiento offline algo atractivo y deseable. Para cubrir estas situaciones,

HTML5 ofrece un mecanismo para que el navegador pueda salvar todos los recursos que utiliza una página para desplegarse correctamente.

Hasta este momento, todos los navegadores mayores, con excepción de Internet Explorer, soportan el API para aplicaciones Web *offline*. Para poder verificarlo, puede utilizar el detector de características HTML5 que se puso a su disposición desde el Cap. 2, la Fig. 8.1 muestra el estado existente del API:

applicationcache	←
draganddrop	
geolocation	
hashchange	
history	
indexeddb	
localStorage	
postmessage	
sessionstorage	
touch	
websockets	
websqldatabase	
webworkers	

Fig. 8.1 Detector HTML5 valida la existencia del API para aplicaciones Web *offline*.

Imagine que quiere hacer una pequeña aplicación de una página, consta del documento HTML, un archivo para hojas de estilo y una imagen. Para incluir soporte *offline* requiere añadir el atributo `manifest` a la etiqueta `<html>`, como se muestra en el siguiente código:

```
<!DOCTYPE html>
<html manifest="aplicacion.manifest">
.
.
.
</html>
```

El archivo manifiesto es un archivo de texto simple, el cual para este caso se debe llamar `aplicacion.manifest`, y lo que contiene es la lista de archivos que la página necesita:

```
CACHE MANIFEST  
javascript_offline.html  
estilo_offline.css  
ClockworkOrange.jpg
```

El archivo *.manifest* debe de comenzar con CACHE MANIFEST (tal cual), los archivos pueden usar rutas relativas.

Otro aspecto importante para que todo esto funcione es que debe de configurar a un tipo MIME específico para el archivo *.manifest* en su servidor Web. El tipo MIME que debe agregar es:

```
text/cache-manifest
```

Para el archivo:

```
.manifest
```

La manera en que agrega el tipo MIME varía dependiendo de cuál servidor Web utilice. Puede experimentar con servidor Web local, o bien, puede pedir la ayuda de su administrador de servidor.

El resto del proceso es exactamente igual al que usaría cuando crea otra página de internet. La prueba final para verificar si su cache offline está funcionando es cargar la página, dar permisos de almacenamiento local si el navegador lo solicita, desconectarse de Internet e intentar cargar la página nuevamente, si la página se despliega, entonces la misma está usando el API para aplicaciones web *offline* correctamente.

No hay que confundir con el caché tradicional de los navegadores, la forma de caché que está utilizando es diferente. Si realiza cambios en el archivo *.manifest* los cambios no se verán hasta después de un par de horas.



En los materiales adicionales encontrará los archivos:
javascript_offline.html, *estilo_offline.css*,
aplicacion.manifest y *ClockworkOrange.jpg*

8.4 Geolocalización

Hoy en día hay situaciones donde la localización es particularmente útil porque puede proporcionar interesantes servicios, sobre todo para aplicaciones móviles. Como por ejemplo, ubicar e informar al usuario de lugares y servicios de su interés que se ubiquen cerca de su posición actual.

Por supuesto, hay que estar consciente de que aunque HTML5 proporciona el API para geolocalización, hay muchos otros factores que influyen y no hay garantía de que el resultado del posicionamiento sea veraz.

Es importante tener en mente esta última situación, porque mientras algunos dispositivos móviles cuentan con un GPS que permite una localización más precisa por una triangulación de antenas y satélites, otros sólo cuentan con conexión vía el ISP que da el servicio de conexión, y el navegador típicamente tomará esta información por IP o por triangulación por puntos de acceso Wi-Fi.

Para implementar la geolocalización primeramente debe asegurarse de que el navegador soporta la API de geolocalización de HTML5, para ello puede utilizar el detector de características HTML5 que se ha puesto a su disposición desde el Cap. 2. La Fig. 8.2 muestra la existencia del API:



Fig. 8.2 Detector HTML5 validando la existencia del API para geolocalización.

Usted puede recuperar la geolocalización del usuario una sola vez, o por petición, esto sería útil en algunos casos, por ejemplo, imagine que tiene hambre a la hora del almuerzo y quiere saber qué restaurante está cerca de su posición actual. Para lograr esto, el objeto que debe utilizar es

`navigator.geolocation`, y la función básica para este caso es `getCurrentPosition()`, su sintaxis es:

```
getCurrentPosition(FuncionExito[, FuncionError], Opciones);
```

- **Parámetro *FuncionExito*.** Este parámetro le dice al navegador a cuál función debería llamar si los datos de localización han sido encontrados como disponibles. La función indicada toma los datos de la localización obtenida y actúa en consecuencia.
- **Parámetro *FuncionError*.** En los escenarios de programación siempre es buena idea tener un plan en caso de alguna falla, es ésta la razón para que este parámetro llame a una función en caso de que la petición de localización falle por alguna razón que escapa de su control. Este parámetro es opcional, pero es buen hábito usarlo siempre.
- **Parámetro *Opciones*.** El parámetro opciones maneja internamente a su vez tres parámetros opcionales que se encierran entre los símbolos "{}":
 - **Opción *enableHighAccuracy*.** Este atributo le indica al navegador que utilice una fuente de información más exacta si está disponible. Los valores posibles son `true` o `false`, usted no puede controlar la fuente de información. Tenga cuidado si coloca el valor a `true`, porque puede ocasionar que la computadora o dispositivo utilice más tiempo y energía para completar la localización.
 - **Opción *Timeout*.** Le indica al navegador el monto máximo de tiempo del que dispone para calcular la posición actual. Si el cálculo no se completa en el tiempo indicado, la función de error se dispara. El valor se da en milisegundos
 - **Opción *maximumAge*.** Señala qué tan vieja puede ser la información de localización antes de volver a recalcularla. El valor también se indica en milisegundos.

Implemente las siguientes funciones JavaScript, puede llamar a `cargaLocalizacion()` desde el evento `onload` del elemento `<body>`:

```
function cargaLocalizacion(){
    navigator.geolocation.getCurrentPosition(muestraMapa,
    MuestraError, {timeout:10000});
}

function muestraMapa(posicion){
    var longitud = posicion.coords.longitude;
    var latitud = posicion.coords.latitude;
```

```

    var enlace = "http://maps.google.com/?ll=" + longitud + "," +
    longitud + "&z=14";
    document.getElementById("long").innerHTML = "Longitud: " +
    longitud ;
    document.getElementById("lat").innerHTML = "Latitud: " + longitud;
    document.getElementById("enlace").href= enlace;
}

function MuestraError(error){
    alert(error.code);
}

</script>

```

El código HTML es:

```

<body onload="cargaLocalizacion();">
    <h1>Geolocalización</h1>
    <p id="info">
        <label id="long"></label><br/>
        <label id="lat"></label><br/>
        <label id="prec"></label><br/>
        <a id="enlace" target="_blank">Enlace de mapa</a><br/>
    </p>
</body>

```

Lo que está sucediendo en este código es sencillo, observe el siguiente análisis paso a paso:

1. Cuando la página carga el evento onload en el elemento <body> es disparado y llama a la función JavaScript cargaLocalizacion(). El código HTML es muy simple, ya que sólo utiliza tres elementos <label> y un enlace <a>.
2. Una vez que comienza el llamado a cargaLocalizacion(), la función llama al método getCurrentPosition, que a su vez llama a la función muestraMapa en caso de obtener los datos de localización correctamente, a la función MuestraError en caso de error, e indica que el proceso de cálculo para la localización debe completarse en 10 000 milisegundos antes de considerar el proceso de localización como erróneo con la opción timeout.
3. Si los datos se obtuvieron correctamente, se llama a la función muestraMapa(posicion), esta función extrae la longitud y la latitud de

la posición actual. Esto se logra con el objeto `position`, el cual tiene una propiedad llamada `coords`. Al mismo tiempo, la propiedad `coords` es también un objeto que posee dos propiedades más, que son `latitude` y `longitude`, estos valores quedan almacenados en las variables locales `longitud` y `latitud`.

4. Una vez que se obtienen los datos de longitud y latitud hay que hacer algo con ellos. En este caso lo que hace el resto de la función con ellos es relacionarlos con Google Maps a través de la variable `enlace`. La variable `enlace` únicamente es una cadena donde se concatenan las variables `latitud` y `longitud`.
5. Una vez que se han asignado los valores a las tres variables (`longitud`, `latitud` y `enlace`) se colocan a las etiquetas HTML `<label>` con la propiedad `innerHTML` y para el enlace a la propiedad `href`.
6. Finalmente, si los datos no se obtuvieron correctamente y en el tiempo límite indicado, la función que se ejecutará es `MuestraError(error)` que únicamente mostrará el código del error.

Se pueden hacer muchas cosas más, pero el proceso base se mantiene igual, por ejemplo, se puede interactuar directamente con el API de Google Maps para hacer más cosas interesantes. La Fig. 8.3 muestra cómo se ve la implementación del código:

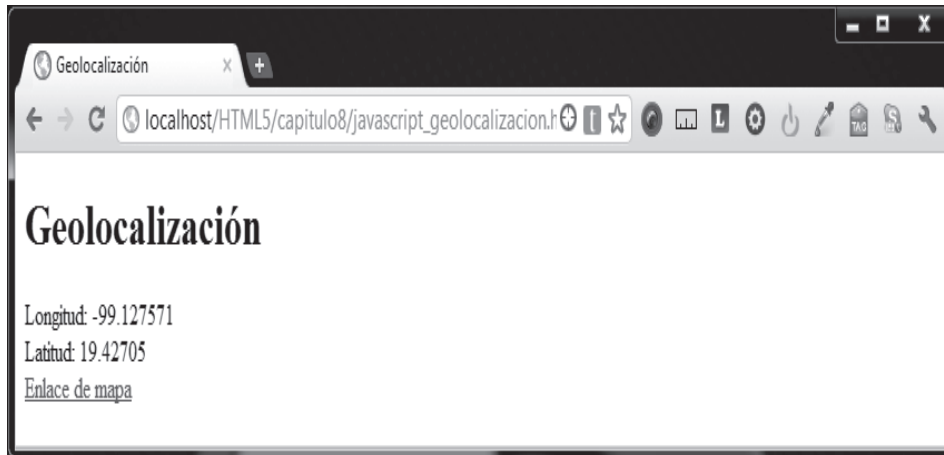


Fig. 8.3 Ejemplo de API de geolocalización HTML5.



En los materiales adicionales encontrará el archivo:
javascript_geolocalizacion.html

8.5 WebSocket

El uso de *sockets* de manera nativa es una característica que sin duda implica uno de los avances técnicos más dramáticos en lo que se refiere al desarrollo Web por parte de HTML5, incluso tiene el potencial de cambiar en gran medida el uso que se le da hoy a Internet.

Para comprender el uso del API WebSocket de HTML 5 (o web sockets) hay que entender a un nivel un poco más profundo el funcionamiento de la Web hoy en día. Todas las plataformas de comunicación en Internet utilizan reglas para enviar información entre sí. La información enviada y recibida a través de la Web no es la excepción, la mayoría del trabajo en el tráfico de información por la Web es usando el protocolo HTTP (*HyperText Transfer Protocol*).

Cuando usted solicita una página remota en su navegador, un servidor Web responde enviando de regreso la página en cuestión, cuando termina el envío, la conexión entre su navegador y el servidor Web remoto se termina y se cierra. Debido a este comportamiento, el protocolo HTTP es un protocolo *sin estado*, en otras palabras, cuando una transacción se finaliza, la conexión entre cliente y servidor se termina, una nueva transacción crea una conexión totalmente nueva. La razón para este comportamiento es que si un usuario solicitara una página y la conexión no se cerrara, la máquina del usuario y el servidor Web estarían gastando recursos de procesamiento y conexión todo el tiempo.

Como ya se mencionó, la Web tiende a convertirse en una plataforma de desarrollo para aplicaciones, en lugar de permanecer únicamente como un gran depósito de textos e imágenes intercambiables. Con esta idea en mente, la naturaleza *sin estado* del protocolo HTTP se vuelve una limitante importante, para enfrentar esta situación, han surgido algunas técnicas y recursos para simular el *estado constante* de diversas partes en una página web, como por ejemplo, los controles Web basados en AJAX, que descomponen una solicitud de datos a un servidor Web en varias solicitudes pequeñas, a veces logrando que el usuario no lo note. Esto permite una relación más fluida entre página y usuario, pero se puede ocasionar mayor ineficiencia en el tráfico de datos y el uso de recursos.

La idea de los *sockets* no es nueva, se utilizaba antes de la Web como se conoce actualmente. Si usted ha utilizado un cliente FTP, SSH, un cliente local de correo o Telnet, entonces ha usado programas que usan *sockets*. Normalmente un programador crea un *socket* en el lado servidor (en estado de espera), y un *socket* en el lado cliente (en estado de lanzador), estos programas deben de usar el mismo tipo de reglas de comunicación para poderse comprender, esto es llamado *protocolo de comunicación*. Un servidor Web es de hecho un programa especializado en gestionar información “hablando” principalmente en protocolo HTTP utilizando *sockets*.

El programa Telnet es una herramienta cliente multipropósito que se utiliza principalmente para probar diferentes tipos de *socket*. Para probar esto, hará uso

de un servidor conocido, utilice en su consola el programa Telnet, escriba la siguiente línea de comando y oprima *enter*:

```
telnet yahoo.com 80
```

Después de oprimir *enter*, verá un mensaje que dirá algo como esto (si el servidor contesta muy rápido quizá no lo alcance a ver bien):

```
Conectándose a yahoo.com...
```

Si marcha bien, es posible que vea un mensaje indicando que se ha conectado al servidor remoto o podría ver simplemente una ventana vacía (esto varía un poco de servidor a servidor). En ambos casos usted se ha logrado conectar y el servidor Web cree que su consola Telnet es un navegador, por lo tanto debe enviarle algo que tenga sentido en protocolo HTTP para que el servidor regrese alguna respuesta útil. Escriba la siguiente petición (use *enter* entre la primera y segunda línea y al terminar oprima *enter* dos veces):

```
GET / HTTP/1.1  
  
host: www.yahoo.com
```

Si escribió correctamente la petición, después del segundo *enter*, comenzará a ver el código de la página que obtuvo como respuesta, si su terminal efectivamente fuera un navegador, entonces el navegador desplegaría una página normal, pero la terminal Telnet muestra sólo el resultado de manera literal. Al final verá una línea similar a ésta:

```
Se ha perdido la conexión con el host.
```

La conexión siempre se cerrará al final de la petición debido a que HTTP es un protocolo *sin estado*.

Cuando se utiliza WebSocket en una página, se provee de un protocolo adicional, mientras que la página en sí misma sigue trabajando en el mismo HTTP *sin estado*. Lo que sucede de esta manera es que mientras se ve una página común y corriente, WebSocket mantiene la conexión paralelamente, usando su propio protocolo, y así continúa mientras la página está viva, permitiendo la comunicación sin tener que restablecer la conexión

Con estas nuevas posibilidades usted puede crear, por ejemplo, un juego multijugador, un reporteador de información en tiempo real o un chat, entre muchas otras cosas, basadas completamente en HTML5, CSS3 y JavaScript.

Para ver cómo funciona el API de WebSockets, se necesita forzosamente contar con un socket servidor; crear uno está fuera del alcance de este libro, ya que para esto requiere de conocer la programación de sockets en algún lenguaje como C, Python, Java, Visual Basic etc., pero por fortuna existen algunas alternativas públicas para hacer pruebas, el sitio <http://websocket.org/> es una de ellas y es la que se utilizará.

El sitio <http://websocket.org/> ofrece un bonito servidor de repetición (*echo server*), lo que hace es simplemente proporcionar un socket que recibe el mensaje del cliente y lo regresa de la misma manera (<ws://echo.websocket.org/>). Este mismo sitio plantea un ejemplo (<http://websocket.org/echo.html>), el cual se ha utilizado como base para crear otro ligeramente mejorado en este libro, la Fig. 8.4 muestra cómo se ve en el navegador:

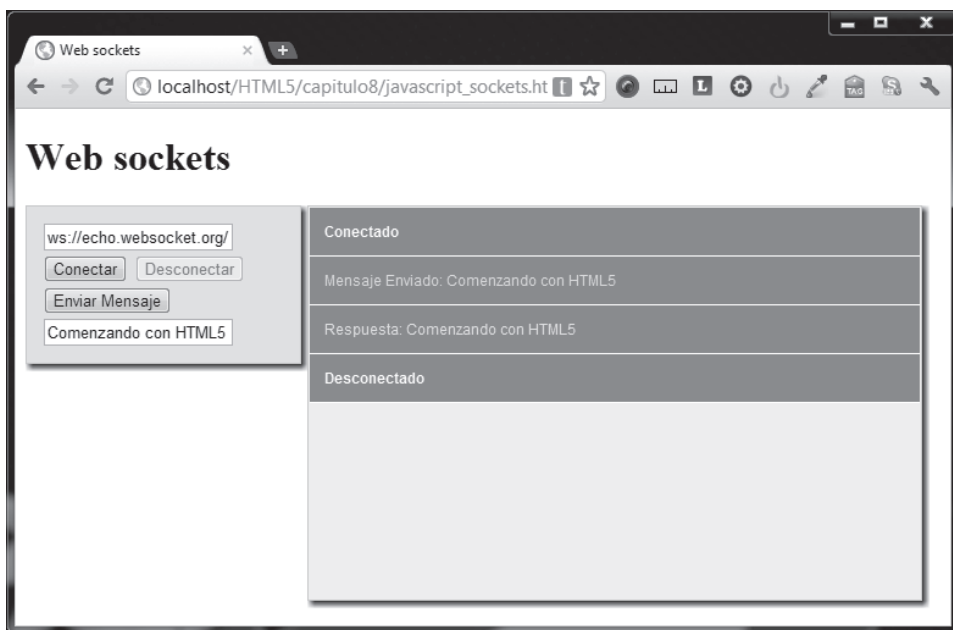


Fig. 8.4 Ejemplo de cliente WebSocket.

El código de este ejemplo se muestra a continuación, luce relativamente largo, pero en realidad es sencillo, dé un vistazo previo y después verá el análisis con detenimiento.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web sockets</title>
```

```
<meta charset="iso-8859-1" />
<style type = "text/css">
    .conectado {
        color: #ffffff;
        font-weight: bold;
    }

    .mensaje {
        color: #e7e4dd;
    }

    .desconectado {
        color: #ffffff;
        font-weight: bold;
    }

    .error {
        color: red;
    }

    .response {
        color: #eeeeee;
    }

    #contenedor{
        width: 2000px;
    }

    p {
        background-color: rgba(0,0,0,0.5);
        padding: 1em;
        margin: 0px;
        border-bottom: 1px solid #ffffff;
    }

    #formulario{
        font-family: arial;
        font-size: 12px;
        width: 200px;
        float: left;
```

```
        background-color: #eeeeee;
        padding: 1em;
        border: 1px solid #cccccc;
        box-shadow: 3px 3px 5px #000000;
    }

    #resultado{
        font-family: arial;
        font-size: 12px;
        width: 500px;
        height: 320px;
        float: left;
        margin-left: 5px;
        overflow: auto;
        background-color: #f6f6f6;
        padding: 1px;
        border: 1px solid #cccccc;
        box-shadow: 3px 3px 5px #000000;

    }
</style>

<script type = "text/javascript">

    var resultado;
    var websocket;

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

    function conectar(){
        if ("WebSocket" in window){
            txtServidor= document.getElementById("txtServidor");
            servidor = txtServidor.value;

            websocket = new WebSocket(servidor);
            resultado.innerHTML = "Conectando..." ;

            //se crean los eventos
```

```
        websocket.onopen = onOpen;
        websocket.onclose = onClose;
        websocket.onmessage = onMessage;
        websocket.onerror = onError;
    }else{
        alert("El API Web sockets de HTML5 (Objeto WebSocket) no
es soportado en este navegador.");
    }
}

//función que es llamada en cuanto la conexión se abre
function onOpen(evt){
    resultado.innerHTML = "<p class='conectado'>Conectado</p>";
    document.getElementById("btnConectar").disabled=true;
}

//función que es llamada en cuanto la conexión se cierra
function onClose(evt){
    resultado.innerHTML += "<p
class='desconectado'>Desconectado</p>";
    document.getElementById("btnDesonectar").disabled=true;
    document.getElementById("btnConectar").disabled=false;
}

//función que es llamada en cuanto se recibe respuesta por
parte del servidor remoto
function onMessage(evt){
    resultado.innerHTML += "<p class='response'>Respuesta: " +
    evt.data + "</p>";
}

//función que es llamada en caso de error en la conexión
function onError(evt){
    resultado.innerHTML += "<p class = 'error'>Error: " +
    evt.data + "</p>";
}

function enviarMensaje(){
    //se obtiene el texto del campo de texto txtMensaje
    txtMensaje= document.getElementById("txtMensaje");
    mensaje = txtMensaje.value;
```

```

        //se envía el mensaje al servidor
        websocket.send(mensaje );
        resultado.innerHTML += "<p class='mensaje'>Mensaje Enviado:
" + mensaje + "</p>";
    }

</script>
</head>
<body onload="iniciar();">
    <h1>Web sockets</h1>
    <div id="contenedor">
        <div id="formulario">
            <label for = "txtServidor">
                <input type="text" id="txtServidor"
value="ws://echo.websocket.org/" />
            </label>
            <button id="btnConectar" onclick="conectar();">
                Conectar
            </button>
            <button id="btnDesconectar" onclick="websocket.close();">
                Desconectar
            </button>
            <button onclick="enviarMensaje();">
                Enviar Mensaje
            </button>
            <input type="text" id="txtMensaje" value="Comenzando con
HTML5" />
        </div>
        <div id="resultado">Clic en botón "Conectar" para iniciar la
conexión con el servidor remoto</div>
    </div>
</body>
</html>

```



En los materiales adicionales encontrará el archivo:
javascript_sockets.html

8.5.1 Creación interfaz

Se requiere crear una interfaz entre el usuario y esta pequeña aplicación Web, de esta forma el usuario puede indicar los datos requeridos e iniciar y terminar la conexión cuando él mismo lo crea conveniente. Para el caso del ejemplo se utilizó un pequeño formulario con tres botones y dos campos de texto siguiendo los pasos siguientes:

1. Después de crear la estructura básica del documento, se agregó en la etiqueta `<body>` el evento `onload` que llamará a la función JavaScript `iniciar()` al cargar la página.
2. Se creó una etiqueta `<input>` de tipo `text` para permitir al usuario ingresar el servidor remoto que desee, sin embargo, previamente capturado aparecerá la URL del servidor de repetición propuesto `ws://echo.websocket.org/`.
3. Se colocó un botón para conectar con el servidor remoto haciendo uso del elemento `<button>`, se utiliza el evento `onclick` para llamar a la función `enviarMensaje()`.
4. Se hizo un botón para desconectar del servidor remoto utilizando el elemento `<button>` y se añadió el evento `onclick` para llamar al método `websocket.close()`, note que no fue necesario crear una función JavaScript adicional.
5. Se creó otro campo `<input>` de tipo `text` para colocar el mensaje que se desea enviar al servidor de repetición.
6. Se agregó un elemento `<button>` para enviar el mensaje escrito dentro del campo de texto hecho para este fin, el botón llama a la función `enviarMensaje()` con su evento `onclick`.
7. Se colocó un área para ver el estado de la comunicación con el servidor remoto utilizando la etiqueta `<div>`.
8. Finalmente se han incluido estilos para mejorar la interfaz y hacerla más agradable a los ojos del usuario.

8.5.2 Conexión de sockets

Si ya tiene listo un socket servidor (como es el caso del ejemplo) entonces es fácil comunicarse con él usando el objeto `WebSocket` en JavaScript, para esto se utilizaron los pasos siguientes:

1. Primero se crearon las variables globales `resultado` y `websocket`. De inmediato se escribió la función `iniciar()`, la cual asigna al elemento HTML `resultado` a la variable del mismo nombre, creada en este mismo paso.
2. Se implementó la función `conectar()`, la cual es importante, ya que cuando el usuario haga clic en el botón “Conectar” la función será llamada,

en ella se verificará el soporte para el WebSocket por parte del navegador, si el navegador soporta el API WebSocket, entonces creará el socket cliente y se asignará a la variable local `websocket`, es en este momento donde se utiliza la URL del socket servidor, para este caso se utilizará <ws://echo.websocket.org/> como opción por default.

3. Enseguida en la misma función `conectar()`, se inicializan los eventos que puede manejar el objeto WebSocket instanciado en la variable `websocket` y se asigna una función para cada uno, los eventos posibles son:
 - **onopen.** Este evento se dispara cuando la conexión se abre. La función utilizada con este evento es `Abrir(evt)`.
 - **onclose.** El evento se dispara cuando la conexión es cerrada. La función utilizada con este evento es `Cerrar(evt)`.
 - **onError.** El evento se dispara si ocurre algún tipo de error durante la conexión. El valor de la propiedad `evt.data` contiene la descripción del error. La función utilizada con este evento es `Error(evt)`.
 - **onmessage.** Si se recibe una respuesta por parte del servidor remoto (que no sea un error) el evento se dispara. Para recuperar el mensaje se utiliza el valor de `evt.data`. La función utilizada con este evento es `MensajeServ(evt)`.

8.5.3 Envío de mensaje

Cuando el usuario decide que desea enviar un mensaje al socket servidor hace clic en el botón “Enviar Mensaje”, esto llama a la función `enviarMensaje()` que básicamente hace 3 simples pasos:

1. Obtiene el texto deseado para el mensaje del campo de texto `txtMensaje`. En el ejemplo simplemente se usa un método `getElementById()` y se asigna a una variable el valor de `txtMensaje`.
2. Asigna el texto del mensaje al método `send()` del objeto WebSocket.
3. Se muestra al usuario el mensaje que se ha enviado en el panel `resultado`. Cuando el servidor responde, el evento `onmessage` se dispara automáticamente y la respuesta del servidor al usuario también es mostrada en el panel de resultados.

8.5.4 Algunas puntualizaciones

El API WebSocket de HTML5 es soportado por Chrome y Safari, FireFox soporta sockets pero utiliza un API desarrollado por la Mozilla Foundation, la cual se llama MozWebSocket, por lo tanto, el ejemplo no funcionará en FireFox. Opera 11 trabaja sin problema con WebSocket, pero los permisos necesarios están desactivados por default, por lo que debe activarlos previamente. Internet Explorer no soporta WebSocket nativamente, ya que según lo que ha declarado Microsoft esperará a que el API WebSocket sea más estable, sin embargo, ofrecen un plug-in que permite a Internet Explorer el uso de sockets: <http://html5labs.interoperabilitybridges.com/> y tiene que usar su propio API experimental, por todo esto el ejemplo tampoco funcionará en IE.

8.6 Web workers

Cuando trabaja en una computadora de escritorio o en una portátil y utiliza programas que se instalan sobre un sistema operativo moderno, implícitamente significa que está trabajando con varios programas al mismo tiempo, incluso aunque no lo note (observe los procesos activos en su administrador de tareas para comprobarlo), esta capacidad se conoce como *multitarea*.

Todos los sistemas operativos modernos en el mercado tienen la capacidad de la multitarea (los móviles, aunque limitada, esencialmente la tienen). Los sistemas de cómputo tienen uno o más procesadores, memoria, etc., todos estos recursos son usados por los programas que cualquier persona utiliza, pero lo interesante es comprender cómo hace un equipo de cómputo y su sistema operativo para prestarle atención a todos los programas que se ejecutan al mismo tiempo. Los sistemas operativos usan algoritmos que actúan como controladores de asignación de tareas (*task-switching*), y proporcionan a una tarea suficientes recursos y tiempo para actuar, después saltar a la siguiente y eventualmente regresar a la anterior si es necesario. Los usuarios, y en muchos casos los mismos desarrolladores, dependiendo del tipo de programas que se hagan, no tienen que pensar en todo esto.

Cuando ejecuta un navegador, éste normalmente funciona como un solo proceso, ya que clásicamente ha trabajado como un medio para consumir un documento. Ahora un navegador puede actuar de manera similar a un sistema operativo, ya que puede manejar aplicaciones internamente, con todo lo que esto implica. Si piensa en el desarrollo Web tradicional, cuando utiliza una instrucción recursiva como un `for` en una función JavaScript, y simplemente hace un conteo de 0 a 90 000, el navegador únicamente podrá dedicarse a hacer ese conteo y no continuará con el resto de sus instrucciones hasta terminar. Si usa un navegador en forma constante (seguramente alguna vez le ha sucedido que éste deja de responder totalmente), la interfaz e incluso el puntero del mouse dejan de responder sobre el navegador. Esto sucede porque cuando el navegador hace todo en un solo

proceso asignado a la CPU, tiene que esperar su turno para hacer todo lo que tiene que ver con él.

Este tipo de situaciones en aplicaciones tradicionales de escritorio se resuelven aplicando lo que se llama *programación por hilos*. Esencialmente un hilo es un programa pequeño que se ejecuta como un proceso independiente, de esta manera tiene su propio turno para consumir recursos. Utilizando esta técnica, la aplicación principal no se congelará si un código muy exigente se encuentra corriendo paralelamente por detrás, ya que éste hace uso de recursos en su propio hilo.

Hasta ahora los navegadores no podían enfrentar estos casos de manera realmente efectiva. Incluso entre desarrolladores no son demasiados aquellos que usan la programación de hilos, ya que esto se considera programación avanzada, por esto el uso de hilos es considerada por muchos una especie de magia oculta que sólo algunos cuantos iniciados utilizan.

Después de todo lo expuesto, HTML5 llega con una nueva característica sumamente interesante llamada Web worker. Esto es una manera muy fácil y poderosa de implementar hilos en el navegador.

Para comprenderlo bien, la mejor manera es un ejemplo que ponga en acción el API Web worker de HTML5. En la Fig. 8.5 puede ver el aspecto del ejemplo que se analizará para ilustrar mejor este tema:

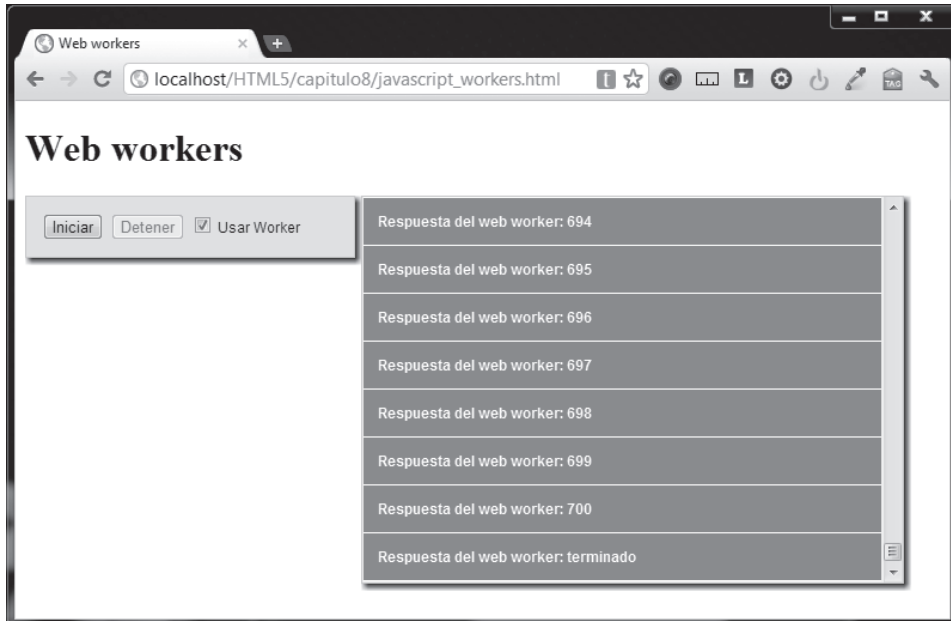


Fig. 8.5 Ejemplo de página que utiliza un Web worker.

El código que utiliza este ejemplo se presenta a continuación, véalo completo y después analícelo más detenidamente en los siguientes puntos.

```
<!DOCTYPE html>

<html>

  <head>

    <title>Web workers</title>
    <meta charset="iso-8859-1" />
    <style type = "text/css">


      .sinworker{
        color: #e7e4dd;
      }


      .conworker{
        color: #ffffff;
        font-weight: bold;
      }


      #contenedor{
        width: 2000px;
      }


      p{
        background-color: rgba(0,0,0,0.5);
        padding: 1em;
        margin: 0px;
        border-bottom: 1px solid #ffffff;
      }


      #formulario{
        font-family: arial;
        font-size: 12px;
        width: 250px;
        float: left;
        background-color: #eeeeee;
        padding: 1em;
        border: 1px solid #cccccc;
        box-shadow: 3px 3px 5px #000000;
      }

    </style>

  </head>

  <body>

    <div id="contenedor">

      <div id="formulario">

        <p>Formulario de contacto</p>

        <div id="datos">

          <div id="nombre">Nombre:</div>
          <input type="text"/>

          <div id="telefono">Telefono:</div>
          <input type="text"/>

          <div id="email">Email:</div>
          <input type="text"/>

          <div id="password">Password:</div>
          <input type="password"/>

          <div id="confirmar_password">Confirmar Password:</div>
          <input type="password"/>

          <div id="boton"><input type="button" value="Enviar"/></div>

        </div>

      </div>

    </div>

  </body>

</html>
```

```
#resultado{
  font-family: arial;
  font-size: 12px;
  width: 450px;
  height: 320px;
  float: left;
  margin-left: 5px;
  overflow: auto;
  background-color: #f6f6f6;
  padding: 1px;
  border: 1px solid #cccccc;
  box-shadow: 3px 3px 5px #000000;
}
</style>

<script type = "text/javascript">

  var resultado;
  var worker;

  function iniciar(){
    var i=0;

    if (typeof(Worker)){
      resultado = document.getElementById("resultado");
      chkUsarWorker = document.getElementById("chkUsarWorker");
      use_worker = chkUsarWorker.checked;
      resultado.innerHTML = "";
      document.getElementById("btnIniciar").disabled=true;
      document.getElementById("btnDetener").disabled=false;
      chkUsarWorker.disabled=true;

      if(use_worker){
        worker = new Worker("worker.js");

        //se asigna función al evento disparado cuando se
        recibe mensaje desde el web worker
        worker.onmessage=mensajeWorker;
      }
    }
  }
</script>
```

```

        //se le indica al web worker que arranque
        worker.postMessage("arranca");
        document.getElementById("btnDetener").disabled=false;
    }else{
        for (i=0; i<=700; i++){
            resultado.innerHTML += "<p class='sinworker'>" + i +
"</p>";

            if (i==700){
                reiniciaBotones();
            }
        }
    }else{
        alert("El API Web Workers de HTML5 no es soportado en
este navegador.");
    }
}

//función que es llamada cuando se recibe respuesta por parte
del Web worker
function mensajeWorker(evt){
    resultado.innerHTML += "<p class='conworker'>Respuesta del
web worker: " + evt.data + "</p>";
    if (evt.data=="terminado"){
        reiniciaBotones()
    }
}

function detenerWorker(){
    worker.terminate();
    reiniciaBotones();
}

function reiniciaBotones(){
    document.getElementById("btnIniciar").disabled=false;
    document.getElementById("btnDetener").disabled=true;
    document.getElementById("chkUsarWorker").disabled=false;
}

</script>
</head>
<body>

```

```

<h1>Web workers</h1>
<div id="contenedor">
  <form id="formulario">
    <button id="btnIniciar" onclick="iniciar();">
      Iniciar
    </button>
    <button id="btnDetener" disabled="disabled"
onclick="detenerWorker();">
      Detener
    </button>
    <input type="checkbox" id="chkUsarWorker">
      Usar Worker
    </input>
  </form>
  <div id="resultado">Clic en botón "Iniciar" para comenzar
conteo</div>
</div>
</body>
</html>

```



En los materiales adicionales encontrará el archivo:
javascript_workers.html y *worker.js*

8.6.1 Creación de interfaz

Para permitir que el usuario interactúe con esta pequeña aplicación Web, se creó una sencilla interfaz siguiendo los pasos siguientes:

1. Se colocó un botón para iniciar un conteo de 1 a 700 utilizando un bucle `for`. El bucle puede ser llamado desde el hilo de un web worker o desde la página principal (véase el paso 3).
2. Se creó un botón para detener el bucle utilizado en el web worker. Este botón sólo tendrá efecto cuando haga el proceso del bucle utilizando el hilo web worker, si el proceso se hace desde la página principal, este botón no funcionará.
3. Se ha colocado una caja de selección o *checkbox* para que pueda experimentar la diferencia en el conteo de 1 a 700 cuando utiliza el hilo de un web Worker, de cuando lo hace todo desde la página principal. El web worker sólo será utilizado si se marca la caja de selección.

4. Se utilizó `<div>` para visualizar el conteo y el comportamiento del bucle y del web worker de manera más clara.
5. Finalmente se incluyeron estilos para mejorar la interfaz del usuario.

Nota

Tenga en cuenta que las interfaces que disparan un web worker, en la realidad serán mucho más sutiles, ya que el objetivo es lograr una mejor experiencia para el usuario y no necesariamente hacer notar cómo se logra.

8.6.2 Llamar al web worker

Lo que se ha hecho podría catalogarse como programación avanzada, aunque de hecho el código es bastante simple. En este caso lo que se ha planteado es solamente la posibilidad de elegir entre dos maneras de ejecutar una instrucción `for`, una con un web worker y otra sin él. Analice los siguientes pasos:

1. Se crearon las variables globales `resultado` y `worker`. A la primera se le asigna un elemento `<div>` para mostrar cómodamente los resultados en esa misma etiqueta. La segunda variable es para asignarle a ésta el objeto `Worker` que se crea.
2. Se usaron cuatro funciones para este ejemplo, aunque sólo 3 son realmente relevantes para el funcionamiento de la aplicación. La función `Iniciar()` es la que desencadena el conteo del ciclo `for` utilizando el web worker o sólo la página principal. La función `MensajeWorker(evt)` será convocada cuando el web worker envíe mensajes a la página principal. La otra función importante es `detenerWorker()`, que por supuesto terminará el hilo abierto por el `worker`. La función `reiniciaBotones()` únicamente activa y desactiva los botones y la caja de selección para regresar al estado apropiado para relanzar el conteo.
3. En la función `Iniciar()` primeramente se verifica la existencia del API Web worker con la instrucción `if (typeof(Worker))`, si el API existe el proceso continuará, y dependiendo de si el usuario marcó la caja de selección `chkUserWorker` el conteo se efectuará con el web worker o en la página principal. Cuando el usuario decide usar el web worker se inicializa el objeto `Worker` y se asigna el archivo `worker.js`. Si el usuario decide no usar el web worker, la página simplemente ejecutará el ciclo `for` y el navegador dejará de responder hasta terminarlo, será hasta entonces cuando se mostrarán los resultados y el usuario recuperará el control del navegador.
4. Una vez inicializado el objeto `Worker` se debe designar una función al evento `onmessage` que en este caso es `mensajeWorker`. Este evento y la función designada se dispararán cuando el `worker` envíe un mensaje de regreso a la página principal (más detalles en el paso 5).

5. Para iniciar el *worker* se debe enviar un mensaje que el código dentro de él pueda reconocer para comenzar a funcionar. En este caso se asignó la cadena “arranca” como mensaje para este fin.
6. En el paso 4 se asignó la función `mensajeWorker` para recibir los mensajes que envía el web worker. La función, por supuesto, debe tener el mismo nombre y debe de recibir un parámetro que contendrá el mensaje enviado, la función finalmente es `mensajeWorker(evt)`. Esta misma función se encargará de comunicar los mensajes obtenidos escribiendo en `resultado` (variable que contiene al elemento `<div>` con ese identificador) y su propiedad `innerHTML` el valor del elemento `evt.data`.
7. Si el usuario desea detener el funcionamiento del web worker mientras aún se está ejecutando, entonces presionará el botón “Detener” y la función `detenerWorker()` hará el trabajo llamando al método `terminate()`, el cual destruye al web worker.

8.6.3 Construir el código del web worker

Hasta este punto lo único que falta para completar el proceso de construcción de ejemplo es el web worker como tal, por fortuna es realmente fácil crearlo. El web worker es un archivo JavaScript con extensión `.js` común y corriente. Este archivo será ejecutado como un programa independiente, pero tenga presente que al ser manejado en un hilo de procesamiento diferente, este archivo no tiene acceso a los elementos u objetos de la página principal (como `window` o `document`), la única manera de comunicar al worker con la página y viceversa, es con el envío de mensajes con el método `postMessage`. Éste es el código utilizado para el del worker, el archivo nombrado `worker.js`:

```
onmessage = conteoWorker;

function conteoWorker(evt){
    if (evt.data==="arranca"){
        for (i=0; i<=700; i++){
            postMessage(i);
        }
        postMessage("terminado");
    }
}
```

Como puede ver, el código del *worker* es muy breve y simple, los pasos utilizados para crearlo fueron los siguientes:

1. Se asignó función al evento `onmessage` para ejecutar de inmediato, ésta es `conteoWorker`. A la función automáticamente se le pasa el argumento `evt`, la función finalmente es `conteoWorker(evt)`.

2. Dentro la función `conteoWorker(evt)` se colocó una condición para verificar que el mensaje fue recibido desde la página principal, de manera que el código pesado comenzará sólo si el mensaje es la cadena “arranca”.
3. Se colocó el código que exige demasiados recursos, que en este caso es un conteo de 0 a 700. Por cada ciclo se envía un mensaje a la página principal usando el método `postMessage` y se disparará la función `mensajeWorker()` (véase el paso 6 del punto 8.6.2).
4. Finalmente se añadió otro mensaje que indica que el proceso ha terminado. En el ejemplo este mensaje se utiliza para indicar a la página principal en qué momento regresar los botones de control al estado en que se puedan usar nuevamente para hacer un conteo nuevo.

8.6.4 Resumen: pasos para crear un Web worker

En lo que se ha visto hasta el momento, se ha explicado en qué consiste y cómo implementar el ejemplo propuesto, pero no es mala idea plantear algunos criterios generales para la implementación general de cualquier Web worker:

1. Normalmente los web workers se utilizan como alternativa al código crítico que está consumiendo demasiados recursos, de manera que comience sus páginas como lo haría normalmente. Después de probar el rendimiento de su código, es entonces cuando puede identificar cuál parte moverá a un Web worker.
2. Prepare un archivo `.js` (JavaScript) por separado para contener el código pesado. En este archivo podrán existir una o varias funciones que normalmente contendrán bucles pesados o inicialización de datos numerosos.
3. En la página principal se debe identificar y relacionar al archivo JavaScript que ha creado, esto lo hará utilizando el objeto `Worker`.
4. Asigne al evento `onmessage` una función para llamar cuando el web worker envíe mensajes a la página principal. La página principal y el web worker serán considerados dos procesos separados para la CPU de una computadora, por lo que la única manera que tienen de comunicarse entre sí es utilizando el método `postMessage`, es por esto que es importante que tenga vinculados el evento `onmessage` y una función.
5. Planee e indique los mensajes que va a enviar al web worker. Los mensajes que envía serán la única manera en que el web worker sabrá qué hacer.
6. En el código del *worker* asigne al evento `onmessage` una función para llamar cuando la página principal envíe mensajes. Justo como hizo en la página principal.

7. Debido a que el web worker no está al tanto de los elementos existentes en la página principal, debe procurar siempre mandar los mensajes apropiados en el momento apropiado de regreso a ella con `postMessage`.
8. Cuando el proceso del web worker termine, recuerde finalizarlo con el método `terminate()`, de lo contrario el proceso se quedará en memoria y continuará consumiendo recursos.

8.6.5 Algunas puntualizaciones entre Web workers y navegadores

Todos los navegadores mayores, con excepción de Internet Explorer 9 y menores, soportan Web workers, sin embargo, hasta el momento Chrome en su versión 14 y Safari 5, no tienen muy buen rendimiento con ellos. Microsoft ha anunciado y hecho demostraciones indicando que soportará Web workers en su versión 10. Firefox y Opera tienen un notable rendimiento con Web workers.

Actividades para el lector

Pruebe el ejemplo tratado a lo largo del punto 8.6, en todos los navegadores mayores (y los que guste incluir), escriba un reporte con sus impresiones acerca del comportamiento del web worker de cada uno de ellos.

Actividades para el lector

Construya su propia versión del ejemplo utilizado para explicar los web workers e implemente un control de formulario adicional para permitir al usuario introducir el límite superior para el conteo.

RESUMEN

En este capítulo se expusieron las características más novedosas en cuanto al lenguaje JavaScript para su interacción con HTML5:

- Las nuevas alternativas para seleccionar elementos HTML5 con JavaScript.
- Las nuevas alternativas para el almacenamiento local de información.
- La manera de crear una página offline.
- La nueva posibilidad para manejar la geolocalización.
- La capacidad de hacer programación con sockets en el navegador.
- La nueva característica de crear procesos con programación por hilos utilizando web workers.

Autoevaluación

1. ¿Para qué sirven las nuevas herramientas de selección de JavaScript?
2. ¿Cuáles son las nuevas maneras para almacenar información del lado cliente?
3. ¿Qué es una página offline?
4. ¿Qué es y para qué puede servir el concepto de Geolocalización?
5. ¿Qué tipo de aplicación podría utilizar websockets?
6. ¿Para qué sirve utilizar un web worker?

EVIDENCIA

☐

Construyó una página con las herramientas de selección `getElementsByClassName`, `getElementsByTagName`, `querySelector`, `querySelectorAll`, para dar diversos estilos a sus elementos.

☐

Construyó su propia versión del ejemplo utilizado para explicar los web workers.

☐

Construyó su propia versión del ejemplo utilizado para explicar los web workers e implementó un control de formulario adicional para permitir al usuario introducir el límite superior para el conteo.

REFERENCIAS

Bibliografía

Lubbers, Peter. Brian Albers y Frank Salim (2010). Pro HTML5. Programming, 1a. ed., Apress, EUA.

Harris, Andy (2011). HTML5 for Dummies, 1a. ed., Wiley Publishing, EUA.

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.



Páginas Web recomendadas

https://developer.mozilla.org/en/Gecko_DOM_Reference

<http://diveintohtml5.org/storage.html>

<http://www.w3.org/TR/offline-webapps/>

<http://dev.w3.org/geo/api/spec-source.html>

<http://www.w3.org/TR/websockets/>

<http://websocket.org/>

<http://dev.w3.org/html5/workers/>

Respuestas sugeridas a las preguntas de autoevaluación

1. Permiten seleccionar los elementos de un documento HTML5 de maneras más flexibles más allá de sólo seleccionar por identificador o por clase.
2. El API de almacenamiento local de HTML5 permite almacenamiento por sesión o almacenamiento local.
3. Es el almacenamiento de los recursos de una página, como imágenes y archivos de manera local, para poder acceder a ellos sin conexión y desplegar una página correctamente.
4. Para crear aplicaciones que utilizan la posición geográfica del usuario y en función a esta proporcionar información de utilidad.
5. Aplicaciones de comunicación, colaboración o información remota en tiempo real
6. Cuando una operación exige demasiados recursos en una página, se puede aislar a ésta en un hilo de proceso separado con un web Worker, para que la página principal pueda continuar con su propio proceso.

Utilizar Canvas

9

Reflexione y responda las preguntas siguientes:

- ¿Qué es y para qué sirve el elemento Canvas de HTML5?
- ¿Cómo preparar el elemento Canvas para su uso?
- ¿Cómo dibujar textos dentro de Canvas?
- ¿Cómo dibujar figuras con diversas formas dentro de Canvas?
- ¿Cómo utilizar imágenes dentro de Canvas?
- ¿Cómo aplicar transformaciones a los dibujos dentro de Canvas?
- ¿Cómo realizar animaciones básicas dentro de Canvas?

Contenido

Utilizando Canvas

Introducción

9.1 Primeros pasos con Canvas

9.2 Dibujando textos

9.2 Dibujando figuras rectangulares

9.3 Dibujando caminos (paths)

9.4 Utilizando imágenes

9.5 Transformaciones

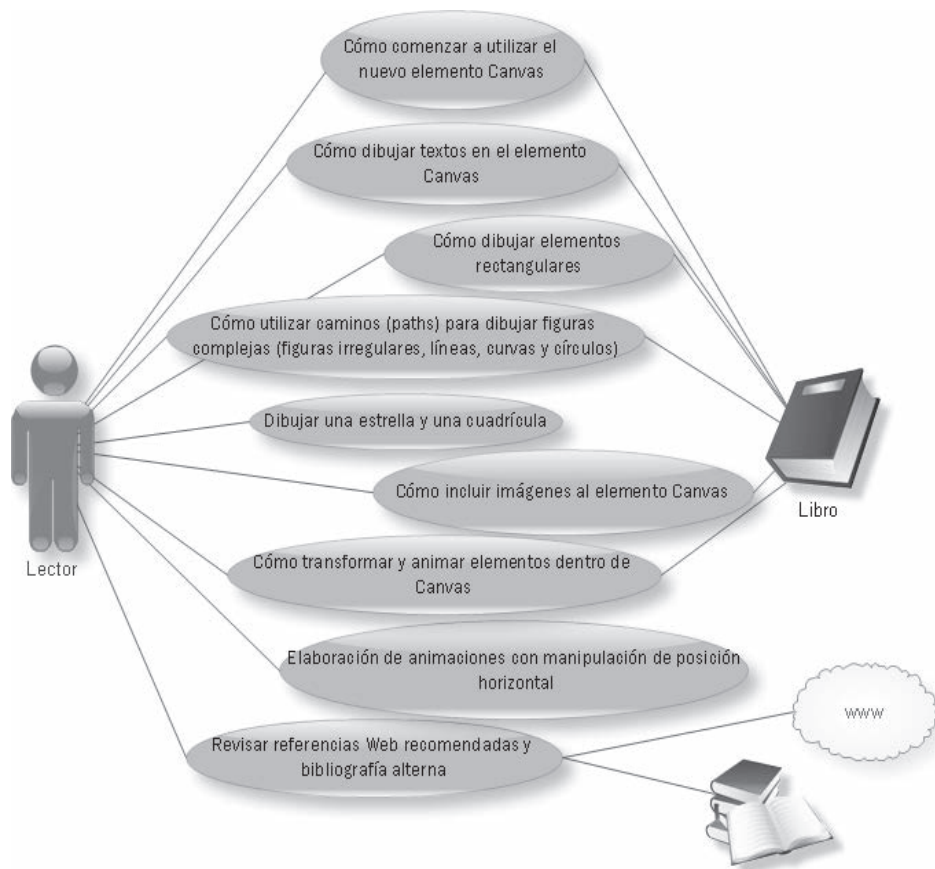
9.6 Animación

Expectativa

Conocer la nueva experiencia en el manejo de gráficos que proporciona HTML5 con el uso del API Canvas.

Después de estudiar este capítulo, el lector será capaz de:

- Conocer las bases del trabajo con el elemento Canvas.
- Dibujar textos como parte del contenido del elemento Canvas.
- Dibujar rectángulos, arcos, círculos y figuras de mayor complejidad utilizando caminos en el elemento Canvas.
- Explotar el uso de imágenes dentro de Canvas.
- Manipular el color, posición e incluso brindar animación a los elementos dentro de Canvas.



INTRODUCCIÓN

El elemento Canvas sin duda es una de las características más interesantes en lo que respecta a HTML5 y entusiasma principalmente a los desarrolladores multimedia. El API Canvas proporciona un área que adquiere un contexto gráfico dentro de un documento HTML.

Cuando se utiliza un elemento Canvas se cuenta también con una serie de métodos y propiedades para dibujar. Las posibilidades para crear gráficos bajo este esquema llevan al desarrollo Web hacia un nuevo rumbo lleno de posibilidades. Hasta hace poco tiempo las únicas maneras de crear juegos, RIA (Rich Internet Application) o elementos visuales realmente interesantes en la Web era haciendo uso de Flash o Java. Con Canvas se tiene una nueva y flexible manera de crear en varios casos ese mismo tipo de contenido, lo cual lleva a serias implicaciones en la relación entre aplicaciones Web, la experiencia del usuario y el uso de recursos.

Canvas lo desarrolló inicialmente Apple para su navegador Safari y luego lo utilizó y estandarizó la organización WHATWG (comunidad muy activa en el desarrollo de HTML) para incorporarlo a HTML5. La propiedad intelectual de Canvas fue sujeto de polémica hasta que Apple decidió finalmente liberar esta propiedad intelectual y la W3C pudo incluir la idea a la especificación HTML5.

Canvas es un elemento que actualmente está soportado por todos los navegadores mayores en sus últimas versiones, esto es algo muy bueno, ya que usted no tendrá problemas de compatibilidad.

Para poder manejar este elemento debe contar con conocimientos básicos de JavaScript, por favor tenga esto en cuenta al leer este capítulo.

9.1 Primeros pasos con Canvas

Hay que comenzar por los aspectos más sencillos de `<canvas>` para comenzar a armar su arsenal de recursos y después logre con ello elaborar gráficos y comportamientos mucho más interesantes y divertidos.

9.1.1 Preparación del elemento Canvas

Cuando se habla del elemento Canvas y al ser tan prometedor pudiera pensar que es complejo colocarlo dentro de un documento HTML, pero nada más lejos de la verdad, en términos de HTML no es más que otra etiqueta con muy pocos parámetros. Lo único que debe hacer es colocar la etiqueta con cierre `<canvas>` y proporcionar valores para sus atributos:

- **Atributo** `width`. Indica el ancho del Canvas en el documento HTML.
- **Atributo** `height`. Indica el alto del Canvas en el documento HTML.

También deberá dar un valor al identificador de la etiqueta usando el atributo `id`, sólo con esto tendrá lista su área para dibujar. La implementación HTML de `<canvas>` se observa de la manera siguiente:

```
<canvas id="miCanvas" width="300" height="300">
  <p>El elemento canvas requiere soporte HTML5<p>
</canvas>
```

Dentro del inicio de la etiqueta `<canvas>` y el cierre `</canvas>` puede colocar cualquier otro elemento HTML, pero éstos sólo se considerarán si el navegador que utiliza no cuenta con soporte para este elemento.

La traducción de la palabra *Canvas* es lienzo, lo que significa que así como sucede con el lienzo de una pintura, para que valga la pena su existencia debería de “pintar” algo interesante en él. En el caso de la etiqueta `<canvas>` nada interesante sucederá hasta que usted decida interactuar con el API Canvas de HTML5 utilizando JavaScript para dibujar dentro. Podría utilizar por ejemplo el siguiente código para dibujar un par de rectángulos dentro de un Canvas:

```
function dibuja(){
    var micanvas = document.getElementById('miCanvas');
    var contexto = micanvas.getContext('2d');
    //color de relleno a negro
    contexto.fillStyle = '#000000';
    //dibuja un rectángulo
    contexto.fillRect(0, 0, 600, 600);
    //cambio de color de estilo de relleno a rojo
    contexto.fillStyle = '#cc0000';
    //dibujo otro rectángulo
    contexto.fillRect(10, 10, 280, 280);
}
```

Este pequeño ejemplo muestra en general cuál es el mecanismo que utiliza el API Canvas. Para resumirlo de manera sencilla podría establecer los siguientes pasos:

1. Crear el área donde se dibujará utilizando la etiqueta `<canvas>`.

2. Asignar a una variable la referencia al Canvas mediante su atributo `id`. Para hacer esto se utilizará por supuesto `getElementById()`.
3. Hay que obtener el contexto gráfico del Canvas utilizando el método `getContext()` del API Canvas. Para el caso actual el contexto gráfico sólo puede ser 2d, pero es muy posible que pronto sea también 3d.
4. Establecer el color del dibujo. Usted puede asignar colores usando cualquier notación de color CSS3 al método `fillStyle` (dibujos con el color indicado como relleno) o `strokeStyle` (dibujos con el color indicado sólo en el perímetro y sin relleno). Para este caso, dado que se desean rectángulos con relleno, se usó `fillStyle`.
5. Se establece qué tipo de dibujo se quiere realizar, pueden ser rectángulos, cuadros, texto, etc. Para el caso en el ejemplo se dibujaron cuadros con colores de relleno, por lo que se utilizó el método `fillRect()`.



En los materiales adicionales encontrará el archivo:
canvas_ini.html.

9.1.2 La cuadrícula y su eje de coordenadas

Primero debe pensar en Canvas como una cuadrícula con un eje de coordenadas, ya que de hecho lo es.

Para posicionar elementos en el Canvas debe tener en cuenta su eje de coordenadas en dos dimensiones, que comienzan en la esquina superior izquierda del elemento. El lienzo o Canvas creado tendrá las dimensiones indicadas con los atributos `width` y `height` en la etiqueta, por lo tanto, la esquina superior izquierda será el punto (0,0) y la esquina inferior derecha el punto definido por `width` y `height`, es decir, el punto máximo de coordenadas marcado por el ancho y el alto indicados.

Normalmente una unidad en la cuadrícula corresponderá a un pixel en el lienzo o Canvas. La Fig. 9.1 ilustra este concepto:

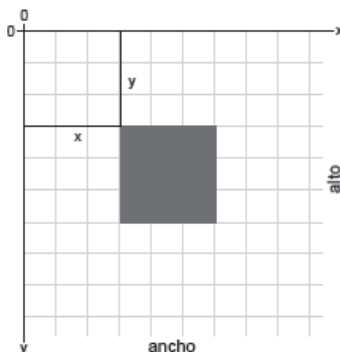


Fig. 9.1 Cuadrícula y ejes imaginarios dentro de un elemento Canvas.

9.2 Dibujar textos

La etiqueta `<canvas>` y el API que la soporta tienen métodos y propiedades para desplegar texto en cualquier punto del lienzo. Los métodos para dibujar texto son dos y tienen la misma sintaxis:

```
strokeText(texto, x, y)
fillText(texto, x, y)
```

- **Parámetro** `texto`. Indica la cadena de caracteres que formarán el texto deseado.
- **Parámetro** `x`. Establece la posición de izquierda a derecha relativa al contenedor para la esquina superior izquierda del texto.
- **Parámetro** `y`. Establece la posición de arriba a abajo relativa al contenedor para la esquina superior izquierda del texto.

El primer método `strokeText()` dibujará sólo la línea perimetral de cada carácter, mientras que el método `fillText()` dibujará un texto lleno con algún color o imagen como patrón (*pattern*).

Las propiedades disponibles para el texto son:

- **Propiedad** `font`. Especifica la fuente del texto del mismo modo en que se hace en CSS.
- **Propiedad** `textAlign`. Especifica la alineación horizontal del texto. Los valores posibles son: `start`, `end`, `left`, `right`, `center`. El valor por default es `start`.
- **Propiedad** `textBaseline`. Especifica la alineación vertical del texto. Los valores posibles son: `top`, `hanging`, `middle`, `alphabetic`, `ideographic`, `bottom`. El valor por default es `alphabetic`.

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');

contexto.fillStyle = '#FFCC00';
contexto.font = 'italic 30px sans-serif';
contexto.textBaseline = 'top';
contexto.fillText('Hola mundo!', 0, 0);
```

```
contexto.strokeStyle = '#FFCC00';  
contexto.font = 'bold 30px sans-serif';  
contexto.strokeText('Hola mundo!', 0, 50);  
}
```

Los pasos para colocar los dos textos que se utilizaron en este ejemplo son:

1. En la etiqueta <body> se llama a la función `dibuja()` usando el evento `onload`. Se prepara el código HTML donde se coloca el elemento <canvas>. **Este paso se dará por hecho en todos los ejemplos posteriores.**
2. Se establece el estilo que se desea para los textos con las propiedades `fillStyle` y `strokeStyle`.
3. Se selecciona la fuente que se desea usar.
4. Se indica la línea de base que se desea utilizar con la propiedad `textBaseline`. Note que sólo se utilizó esta propiedad explícitamente para el primer texto.
5. Finalmente se coloca el texto deseado.

La Fig. 9.2 muestra el resultado del ejemplo anterior en el navegador:

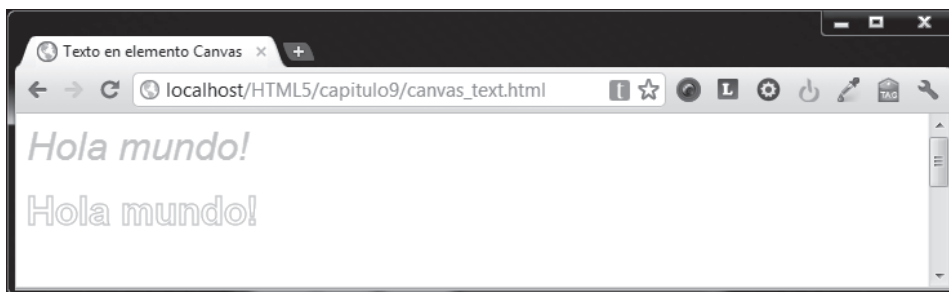


Fig. 9.2 Página que dibuja texto en el elemento Canvas.



En los materiales adicionales encontrará el archivo:
canvas_text.html.

Actividades para el lector

1. Cree el esqueleto básico de un documento HTML5 preparado para usar un elemento <canvas> (HTML y JavaScript).
2. En el documento creado dibuje dentro de su Canvas el texto con la leyenda "Hola mundo".

9.3 Dibujar figuras rectangulares

Las únicas figuras básicas que se puede dibujar directamente utilizando la etiqueta `<canvas>` son rectángulos. No se alarme, el API Canvas proporciona funciones de dibujo por caminos que permiten hacer figuras mucho más complejas, pero eso se verá hasta el siguiente punto.

Por ahora eche un vistazo a los rectángulos. Existen tres métodos para dibujar rectángulos:

MÉTODOS	DESCRIPCIÓN
<i>fillRect(x,y,ancho,alto)</i>	Dibuja un rectángulo relleno.
<i>strokeRect(x,y,ancho,alto)</i>	Dibuja el contorno de un rectángulo.
<i>clearRect(x,y, ancho,alto)</i>	Limpia un área rectangular y la hace totalmente transparente.

Cada una de estas funciones usa los mismos parámetros:

- **Parámetro x.** Establece la posición de izquierda a derecha relativa al contenedor, donde se colocará el ángulo superior izquierdo del rectángulo.
- **Parámetro y.** Establece la posición de arriba a abajo relativa al contenedor, donde se colocará el ángulo superior izquierdo del rectángulo.
- **Parámetro ancho.** La longitud horizontal del rectángulo.
- **Parámetro alto.** La longitud vertical del rectángulo.

Modificando la función `dibuja()` y utilizando todas las funciones para dibujar rectángulos se ha elaborado el siguiente código:

```
function dibuja(){  
  var micanvas = document.getElementById('miCanvas');  
  var contexto = micanvas.getContext('2d');  
  contexto.fillRect(10,10,300,300);  
  contexto.clearRect(40,40,230,230);  
  contexto.strokeRect(50,50,210,210);  
}
```

El método `fillRect()` dibuja un rectángulo relleno (el color por default es negro), enseguida con la función `clearRect()` se crea un área rectangular

transparente sobre el rectángulo negro y finalmente se dibuja el perímetro de un rectángulo con el método `strokeRect()` centrado sobre los dos anteriores. La Fig. 9.3 muestra este código en ejecución.

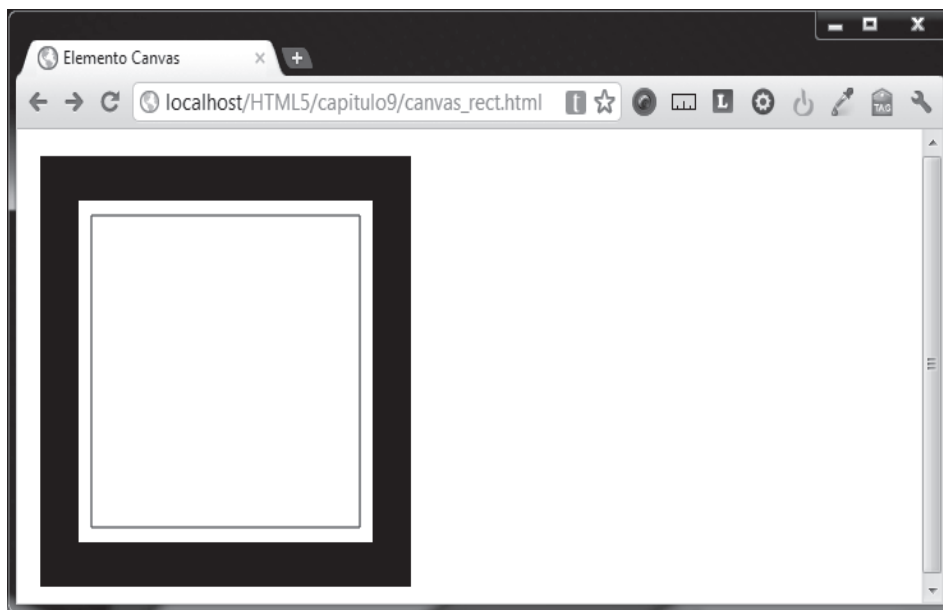


Fig. 9.3 Página donde se dibujan rectángulos.



En los materiales adicionales encontrará el archivo:
canvas_rect.html.

9.4 Dibujar caminos (paths)

Cuando trabaja con Canvas existen diversas funciones que pueden servir para dibujar siluetas como a usted convenga, éstas se utilizan para realizar figuras más complejas. El mecanismo consiste en situarse en un punto inicial dentro del Canvas, luego definir un punto nuevo, como resultado una línea será trazada entre el primero y el segundo, después puede elegir un tercer punto y una nueva línea será trazada entre el segundo y el tercero, de esa manera se continúa cuantas veces sea necesario. Puede pensar en esta lógica como hacer trazos con un lápiz de un punto a otro sin despegar la punta del papel hasta terminar su figura.

Los métodos comunes para dibujar cualquier tipo de camino o *path* utilizan los siguientes métodos:

MÉTODO	DESCRIPCIÓN
<i>beginPath()</i>	Método que indica al contexto del Canvas que se va a comenzar a dibujar un <i>path</i> .
<i>closePath()</i>	Método que cierra un <i>path</i> trazando una línea desde el punto actual al punto inicial.
<i>stroke()</i>	Traza sólo la línea que se ha formado tras colocar los puntos en el camino o <i>path</i> .
<i>fill()</i>	Dibuja la figura rellena que resulta después de trazar el camino o <i>path</i> con los puntos dados.

Si mantiene presentes estos nuevos métodos, el proceso para crear un camino o *path* nuevo no es muy complejo, puede seguir los siguientes pasos generales:

1. Generar todo el contexto gráfico. Hay que crear el Canvas, la referencia y el contexto 2d.
2. Hay que indicar el estilo de la línea con la propiedad `strokeStyle` o si se desea el estilo de relleno con la propiedad `fillStyle` para el *path*. Esto puede ser un color o una imagen que trabaja como patrón (`pattern`).
3. Comenzar el camino. El paso con que comienza realmente a crear una ruta es llamar al método `beginPath()`. Internamente, las rutas se almacenan como una lista de subcaminos (líneas, arcos, etc.), que en conjunto forman una figura. Cada vez que se llama a este método, esa lista se vacía y puede empezar a dibujar nuevas figuras.
4. Continuar con el dibujo del camino. El siguiente paso es llamar a los métodos que realmente indican la ruta que seguirá el *path* o camino. Estos métodos se abordarán en breve.
5. Éste es un paso opcional, puede llamar al método `closepath()`. Este método trata de cerrar la figura trazando una línea recta desde el punto actual hasta el punto de inicio. Si la figura ya se cerró o sólo hay un punto en el *path*, esta función no hará nada.
6. El último paso será llamar a los métodos `stroke()` y/o `fill()`. Cuando se ha llamado a uno de éstos es cuando en realidad se dibuja la figura en el lienzo o Canvas. Se utiliza `stroke()` para dibujar la línea perimetral de la figura, mientras que `fill()` se utiliza para pintar el relleno de una figura "sólida".

Para darle más sentido a todo lo anterior, pondrá en práctica todo esto mientras aprende métodos para crear diferentes caminos o *paths*, como por ejemplo el mostrado en la Fig. 9.4.

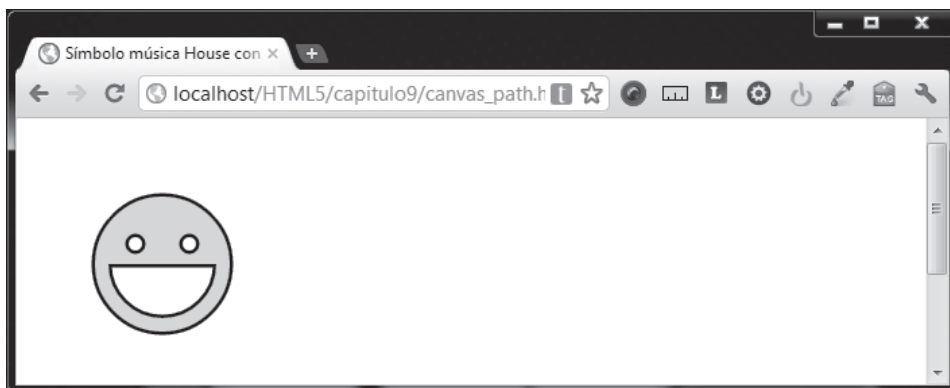


Fig. 9.4 Página con ejemplo del uso de caminos (*paths*).



En los materiales adicionales encontrará el archivo:
canvas_path.html.

9.4.1 moveTo ()

Para lograr la imagen de la Fig. 9.4 se utilizó el método `moveTo()` el cual es muy útil, aunque realmente no dibuja nada, sin embargo, indica dónde ubicar al primer punto imaginario donde se comienza a hacer el *path* y es el primer miembro de la lista de rutas que se mencionó anteriormente. Como ya se señaló, puede pensar en esto como usar una pluma o un lápiz sobre un pedazo de papel que comienza a trazar líneas de un punto al siguiente.

Recibe como parámetro los puntos *x* y *y*, que es el lugar donde ha de moverse el puntero imaginario para comenzar un trazo. Considere el siguiente código como ejemplo:

```
function dibuja(){  
var micanvas = document.getElementById('miCanvas');  
var contexto = micanvas.getContext('2d');  
  
contexto.strokeStyle = "black";  
contexto.beginPath();  
contexto.moveTo(50,5);  
contexto.lineTo(100,85);  
contexto.lineTo(50,165);
```



```
contexto.lineTo(0,85);  
contexto.fill();  
}
```

El resultado de este código se muestra en la Fig. 9.5.

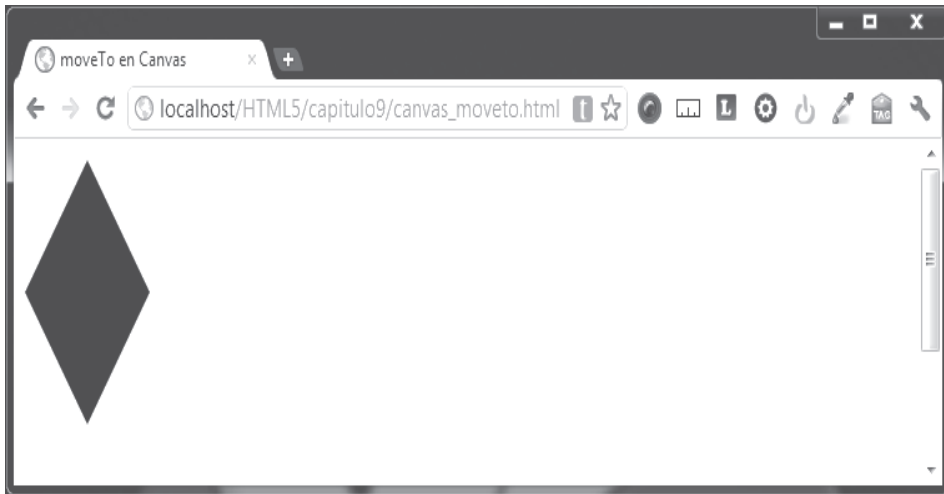


Fig. 9.5 Página con ejemplo del uso del método `moveTo()`.



En los materiales adicionales encontrará el archivo:
canvas_moveto.html

9.4.2 lines()

Para trazar líneas rectas se utiliza el método `lineTo()`. Este método toma dos argumentos `x` y `y`, que son las coordenadas del punto final de la línea. El punto de partida depende de los trazos anteriores, es decir, el punto final de la trayectoria anterior es el punto de partida para el siguiente, etc. El punto de partida también se puede cambiar mediante el uso del método `moveTo()`.

Si dibujara dos triángulos, uno lleno y otro sólo con el esbozo del perímetro, en primer lugar debe llamar al método `beginPath()` para iniciar un camino para una nueva figura. A continuación se utiliza el método `moveTo()` para mover el punto de partida hasta la posición deseada. Continúe con el proceso utilizando `lineTo()`, y para terminar el triángulo lleno simplemente llame al método `fill()`. Para el caso del triángulo sin relleno tendrá que usar una vez más `moveTo()` o `closePath()`. Observe una posible implementación de este ejemplo:

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');

contexto.strokeStyle = "black";

//triángulo con relleno
contexto.beginPath();
contexto.moveTo(10,10);
contexto.lineTo(110,10);
contexto.lineTo(10,110);
contexto.fill();

//perímetro del triángulo
contexto.beginPath();
contexto.moveTo(120,10);
contexto.lineTo(220,10);
contexto.lineTo(120,120);
contexto.closePath();
contexto.stroke();
}
```

Notará la diferencia entre el triángulo relleno y el vacío. Esto es, como se mencionó anteriormente, ya que las figuras se cierran automáticamente cuando el *path* o camino está lleno, pero no cuando se utiliza *stroke()* sólo para trazar el perímetro. Si no se hubiera utilizado *closepath()* para el triángulo vacío, sólo dos líneas hubieran sido dibujadas, no un triángulo completo. El resultado del código utilizado lo puede ver en la Fig. 9.6.



En los materiales adicionales encontrará el archivo:
canvas_lineto.html.

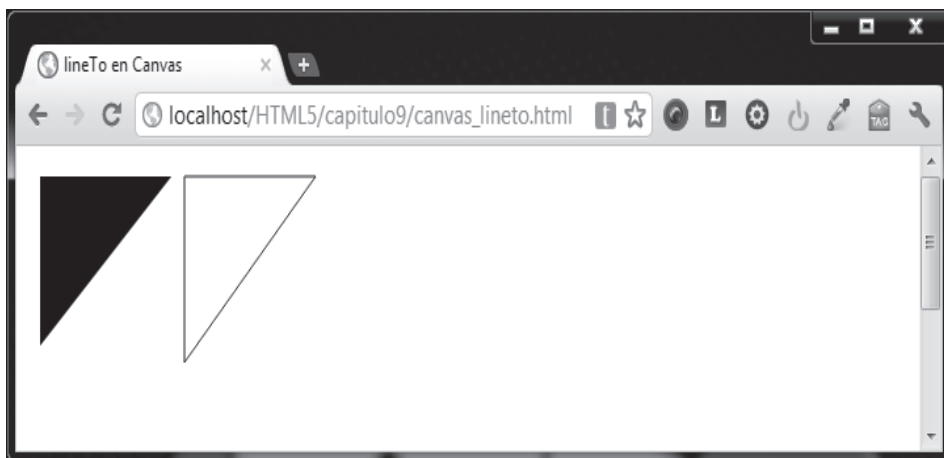


Fig. 9.6 Página con ejemplo del uso del método `lineTo()`.

9.4.3 `arc()`

Para crear arcos y círculos se utiliza el método `arc()`. La sintaxis para este método es la siguiente:

```
arc(x, y, radio, anguloInicio, anguloFin, sentido);
```

- **Parámetro `x`.** Establece la posición de izquierda a derecha relativa al contenedor, donde se colocará el centro del arco o círculo.
- **Parámetro `y`.** Establece la posición de arriba hacia abajo relativa al contenedor, donde colocará el centro del arco o círculo.
- **Parámetro `anguloInicio`.** Define el punto inicial del arco indicado en radianes.
- **Parámetro `anguloFin`.** Define el punto final del arco indicado en radianes.
- **Parámetro `sentido`.** Parámetro opcional, es un valor booleano (`true` o `false`) para indicar si el arco se trazará en contra del sentido de las manecillas del reloj (`true`) o en sentido de ellas (`false`), el valor por default es `false`.

Nota

Los ángulos en el método `arc()` se miden en radianes, no en grados. Para convertir grados a radianes se puede utilizar la siguiente expresión JavaScript:
`variable = radianes (Math.PI/180) * grados`

Para comprender mejor los radianes, así como la referencia sobre los ejes x y y observe la Fig. 9.7.

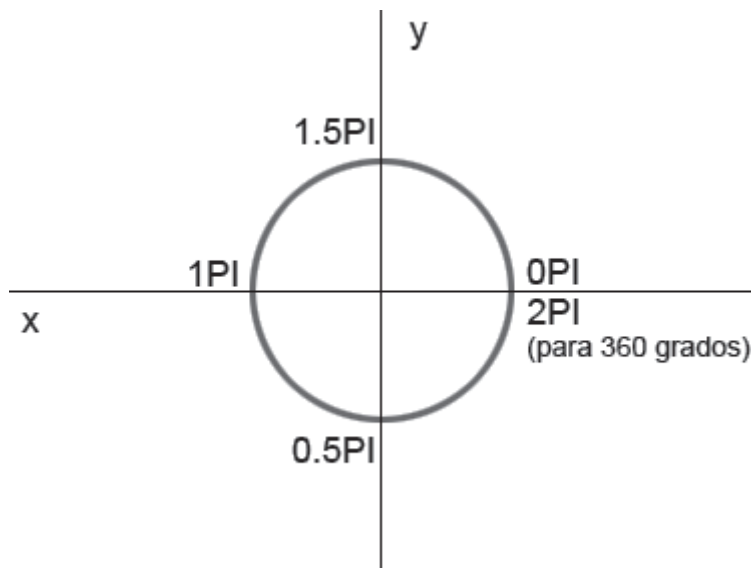


Fig. 9.7 Visualización de ejes y radianes para un arco.

En la Fig. 9.7 tenemos varios valores de radianes:

- 0 radianes son cero grados y es el punto marcado por 0PI, en el eje de las x y a la derecha del centro de la circunferencia.
- 0.5PI radianes son 90 grados y es el punto del eje de las y abajo del centro.
- 1PI radianes es media circunferencia o 180 grados.
- 1.5PI radianes es el equivalente a 270 grados.
- 2PI radianes son 360 grados, es decir, la circunferencia completa y corresponde con el mismo punto que los cero grados.

Analice el siguiente código que utiliza ángulos finales diferentes, para poner en práctica todos estos parámetros, y dibujar diferentes arcos en su versión sólo de línea de circunferencia y también con rellenos.

```
function dibuja(){  
var micanvas = document.getElementById('miCanvas');  
var contexto = micanvas.getContext('2d');
```

```
//primer arco
contexto.strokeStyle = "black";
contexto.beginPath();
contexto.arc(150,150,100,0,Math.PI*2);
contexto.stroke();

//segundo arco
contexto.strokeStyle = '#ff8800';
contexto.beginPath();
contexto.arc(150,150,80,Math.PI*0,Math.PI*1.5,false);
contexto.stroke();

//tercer arco
contexto.strokeStyle = '#ff0000';
contexto.beginPath();
contexto.arc(150,150,60,Math.PI*0,Math.PI,false);
contexto.stroke();

//cuarto arco
contexto.strokeStyle = '#00ff00';
contexto.beginPath();
contexto.arc(150,150,40,0,Math.PI*0.5,false);
contexto.stroke();

//arcos rellenos
//primer arco lleno
contexto.fillStyle = "black";
```

```
contexto.beginPath();
contexto.arc(400,150,100,0,Math.PI*2);
contexto.moveTo(400,150);
contexto.fill();

//segundo arco lleno
contexto.fillStyle = '#ff8800';
contexto.beginPath();
contexto.arc(400,150,80,Math.PI*0,Math.PI*1.5,false);
contexto.fill();

//tercer arco lleno
contexto.fillStyle = '#ff0000';
contexto.beginPath();
contexto.arc(400,150,60,Math.PI*0,Math.PI,false);
contexto.fill();

//cuarto arco lleno
contexto.fillStyle = '#00ff00';
contexto.beginPath();
contexto.arc(400,150,40,0,Math.PI*0.5,false);
contexto.fill();
}
```

El elemento <canvas> se hizo un poco más ancho para este ejemplo modificando la propiedad width:

```
<canvas id="miCanvas" width="700" height="300">
  <p>El elemento canvas requiere soporte HTML5<p>
</canvas>
```

Puede ver el resultado del código en la Fig. 9.8.

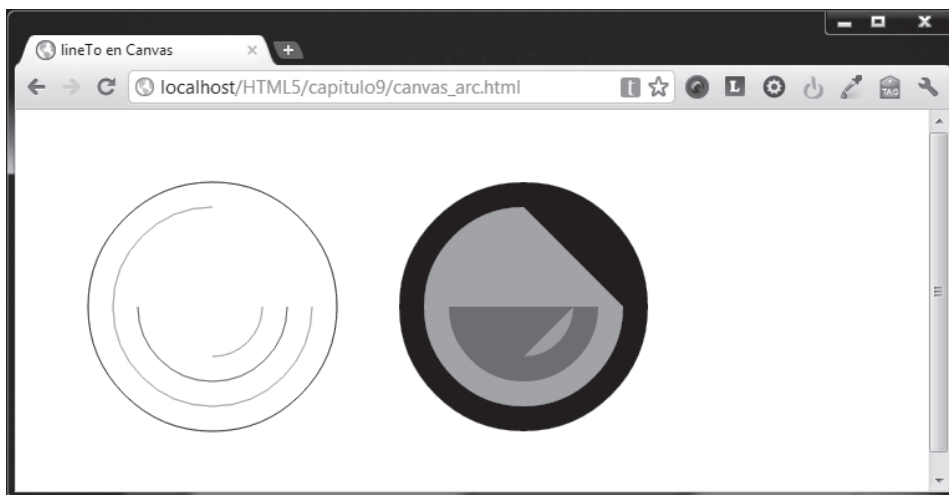


Fig. 9.8 Página con ejemplo del uso del método `arc()`.



En los materiales adicionales encontrará el archivo:
`canvas_arc.html`.

9.4.4 Curvas cuadráticas

Existen dos métodos para crear curvas en un Canvas, uno de ellos es para dibujar curvas cuadráticas, que es un tipo de curva que cuenta con un punto de inicio y un punto final, sin embargo, la línea entre estos dos puntos es afectado por un punto de control, el método es `quadraticCurveTo()`. La sintaxis para este método es:

```
quadraticCurveTo(PuntoControl1x, PuntoControl1y, x, y)
```

- **Parámetro *PuntoControl1x*.** Establece la posición de izquierda a derecha relativa al contenedor, del punto de control.
- **Parámetro *PuntoControl1y*.** Establece la posición de arriba hacia abajo relativa al contenedor, del punto de control.

- **Parámetro x.** Establece la posición de izquierda a derecha relativa al contenedor del punto de final de la curva.
- **Parámetro y.** Establece la posición de arriba hacia abajo relativa al contenedor del punto de final de la curva.

Cabe mencionar que el punto de control usualmente no está en la trayectoria de la curva, es decir, este punto determina la “tendencia” de la curva. Dé un vistazo al siguiente código, en él se incluye, además de una curva cuadrática, una función para mostrar en dónde se encuentra el punto de control:

```
var micanvas;
var contexto;

function dibuja(){
    micanvas = document.getElementById('miCanvas');
    contexto = micanvas.getContext('2d');

    contexto.strokeStyle = "black";
    contexto.lineWidth = 5;
    contexto.beginPath();
    contexto.moveTo(50,150);
    contexto.quadraticCurveTo(150,10,250,150);
    contexto.stroke();

    //para mostrar el punto de control
    muestraPunto(150, 10, "orange");
}

function muestraPunto(x, y, color){

    contexto.fillStyle = color;
    contexto.beginPath();
    contexto.arc(x, y, 5, 0, Math.PI*2, true);
    contexto.fill();
    contexto.closePath();

}
```


Una vez armado el esqueleto del archivo creando el elemento `<canvas>` y el contexto gráfico en la función, el proceso de implementación de este código es simple:

1. Se utiliza el método `moveTo()` para establecer la posición del puntero de dibujo en el primer punto de la curva cuadrática.
2. Se coloca el punto de control con los primeros dos parámetros del método `quadraticCurveTo()`. Ese punto no se ve explícitamente en la curva, como se ha mencionado, simplemente sirve para definir la tendencia de la curvatura.
3. Con los dos parámetros finales de `quadraticCurveTo()` se coloca el tercer punto. Éste es el final de la curva.

La Fig. 9.9 muestra el resultado del ejemplo en el navegador:



Fig. 9.9 Página con curva cuadrática.



En los materiales adicionales encontrará el archivo:
canvas_curva_cuadratica.html.

9.4.5 Curvas bezier

El segundo método para crear curvas en un Canvas es `bezierCurveTo()` los parámetros son muy similares a los del método, pero este tipo de curva toma un segundo punto de control. La sintaxis para este método es:

```
bezierCurveTo(PuntoControl1x, PuntoControl1y, PuntoControl2x,  
PuntoControl2y, x, y)
```

- **Parámetro** *PuntoControl1x*. Establece la posición de izquierda a derecha relativa al contenedor para el primer punto de control.
- **Parámetro** *PuntoControl1y*. Establece la posición de arriba hacia abajo relativa al contenedor para el primer punto de control.
- **Parámetro** *PuntoControl2x*. Establece la posición de izquierda a derecha relativa al contenedor para el segundo punto de control.
- **Parámetro** *PuntoControl2y*. Establece la posición de arriba hacia abajo relativa al contenedor para el segundo punto de control.
- **Parámetro** *x*. Establece la posición de izquierda a derecha relativa al contenedor del punto de final de la curva.
- **Parámetro** *y*. Establece la posición de arriba hacia abajo relativa al contenedor del punto de final de la curva.

A continuación se propone un código para dibujar una curva bezier, y al igual que en el ejemplo de curva cuadrática, se utiliza una función para mostrar los dos puntos de control:

```
var micanvas;  
var contexto;  
  
function dibuja(){  
    micanvas = document.getElementById('miCanvas');  
    contexto = micanvas.getContext('2d');  
  
    contexto.strokeStyle = "black";  
    contexto.lineWidth = 5;  
    contexto.beginPath();  
    contexto.moveTo(10,100);  
    contexto.bezierCurveTo(110, 10, 180, 190, 290, 100);  
    contexto.stroke();  
  
    //para mostrar los puntos de control  
    muestraPunto(110, 10, "orange");  
    muestraPunto(180, 190, "orange");  
}
```

```
function muestraPunto(x, y, color){  
  
    contexto.fillStyle = color;  
    contexto.beginPath();  
    contexto.arc(x, y, 5, 0, Math.PI*2, true);  
    contexto.fill();  
    contexto.closePath();  
}
```

El mecanismo de la curva bezier es el mismo que para la curva cuadrática, a excepción de requerir las coordenadas del segundo punto de control en el método `bezierCurveTo()`.

La Fig. 9.10 muestra el resultado del código del ejemplo en el navegador.

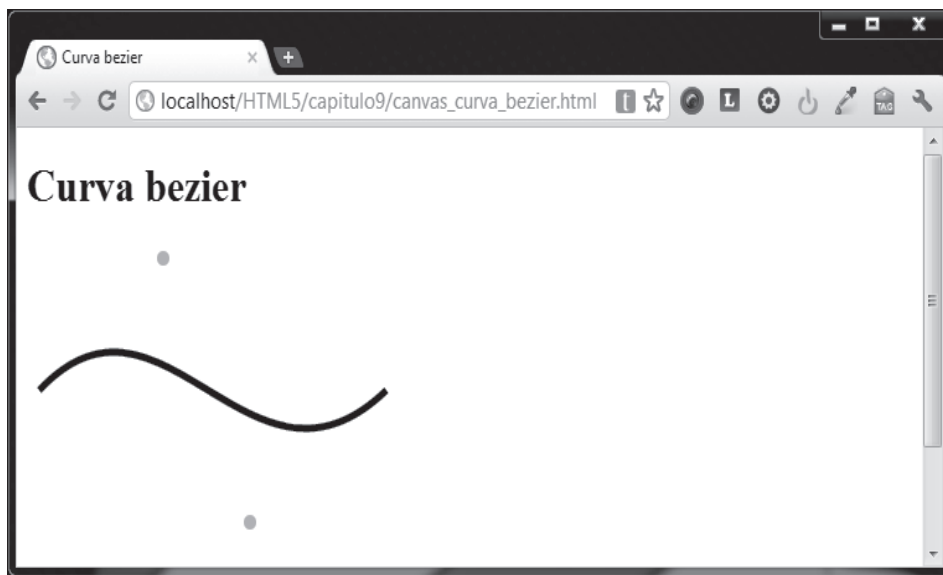


Fig. 9.10 Página con curva bezier.



En los materiales adicionales encontrará el archivo:
canvas_curva_bezier.html.

Actividades para el lector

1. Cree un documento y dibuje una estrella de cinco picos llena con color verde utilice caminos.
2. Cree otro documento y dibuje un cuadro perfecto y un medio círculo sin relleno.

9.5 Utilizar imágenes

Sin duda las imágenes han sido importantes de los documentos HTML desde hace bastante tiempo, pero ahora con la etiqueta `<canvas>` las imágenes adquieren un nuevo dinamismo integrándose a las capacidades de dibujo vectorial, características de transformación y animación, entre otras novedades que proporciona el elemento.

Las imágenes provenientes de archivos gráficos pueden incluirse en el área de un elemento Canvas, de esta manera puede conseguir que sus documentos sean más ricos que si sólo dibuja con las funciones JavaScript del API Canvas de HTML5.

Con un poco de creatividad y código JavaScript, puede lograr resultados basados en imágenes realmente interesantes. La única restricción es que el navegador soporte el formato de las imágenes que utiliza.

9.5.1 `drawImage()`

El método utilizado para dibujar una imagen dentro del lienzo formado por la etiqueta `<canvas>` es `drawImage()`, su sintaxis es:

```
drawImage(objetoImagen, x, y)
```

- **Parámetro** *objetoImagen*. Indica la imagen que se desea incluir dentro del Canvas.
- **Parámetro** *x*. Establece la posición de izquierda a derecha relativa al contenedor para la esquina superior izquierda de la imagen.
- **Parámetro** *y*. Establece la posición de arriba hacia abajo relativa al contenedor para la esquina superior izquierda de la imagen.

En el siguiente ejemplo se muestra el código que utiliza `drawImage()` para colocar una imagen; se emplea el objeto `Image()` como primera técnica para dibujar la imagen dinámicamente sólo con JavaScript y después se usa el método `getElementById()` para tomar un elemento `` y colocarlo directamente en el lienzo:

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');
var img = new Image();
var img2 = document.getElementById('clockwork');

//llamando a la imagen dinámicamente
img.src = 'ClockworkOrange_small.jpg';
contexto.drawImage(img, 10, 10);

//llamando a un elemento <img>
contexto.drawImage(img2 , 100, 100);
}
```

El código HTML que complementa este ejemplo es:

```
<canvas id="miCanvas" width="300" height="300">
    <p>El elemento canvas requiere soporte HTML5<p>
</canvas>


```

A continuación se proporcionan más detalladamente los pasos que se describieron con anterioridad para implementar este ejemplo:

1. Se prepara el código HTML donde además del elemento `<canvas>` se incluye una etiqueta ``, que por supuesto llama a una imagen.
2. En la función JavaScript se crean dos variables. La primera variable es `img` y se le asigna el objeto `Image` de JavaScript. La segunda `img2` se le asigna directamente el elemento `` con `id` con valor `clockwork` utilizando el método `getElementById()`.
3. Se dibuja la primera imagen de manera dinámica utilizando JavaScript. A la propiedad `src` se le asigna la ruta de la imagen que se desea usar. Se utiliza el método `drawImage()` usando el objeto guardado en la variable `img`, y las coordenadas deseadas para la posición de la imagen.

- Se dibuja la misma imagen, pero ahora se utiliza la variable *img2* que contiene directamente el elemento ``. Se asigna al método `drawImage()` la variable *img2* y las coordenadas deseadas para la posición de la imagen.

La Fig. 9.11 muestra el resultado del ejemplo anterior en el navegador.

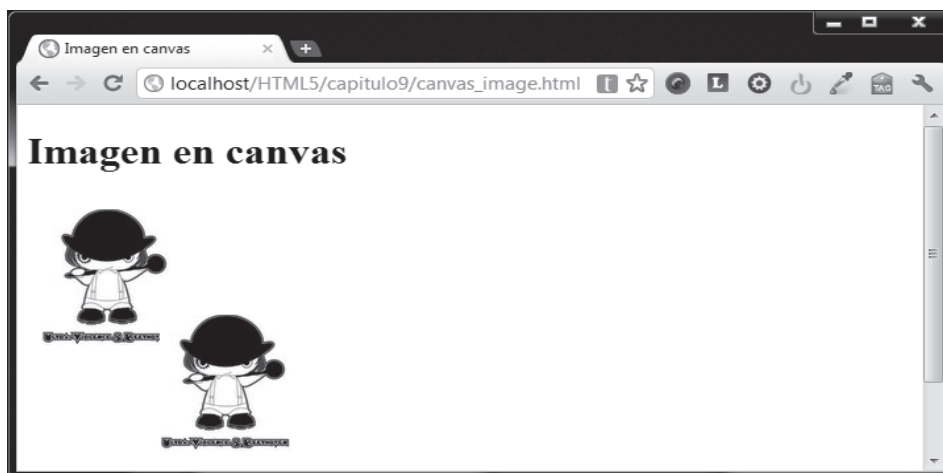


Fig. 9.11 Página que dibuja imágenes en el elemento Canvas.



En los materiales adicionales encontrará los archivos:
canvas_image.html
ClockworkOrange_small.jpg

9.5.2 Escalar imagen con `drawImage()`

El método `drawImage()` tiene algunas variantes, una de éstas permite escalar una imagen, es decir, permite redimensionar su tamaño. El mecanismo para usar `drawImage()` se mantiene, pero se agregan dos parámetros adicionales:

```
drawImage(objetoImagen, x, y, ancho, alto)
```

Observe la definición de los nuevos parámetros, para ver la definición de los primeros véase el punto 9.4.1.

- **Parámetro *ancho*.** Es el ancho deseado para la imagen que se desea incluir dentro del Canvas.
- **Parámetro *alto*.** Es la altura deseada para la imagen que se desea incluir dentro del Canvas.

Observe el código de la función de ejemplo que se muestra a continuación:

```
function dibuja(){  
var micanvas = document.getElementById('miCanvas');  
var contexto = micanvas.getContext('2d');  
var img = new Image();  
  
//llamando a la imagen dinámicamente  
img.src = 'ClockworkOrange_small.jpg';  
contexto.drawImage(img, 10, 10, 170, 250);  
  
}
```

El mecanismo no tiene ninguna diferencia con el método original, a excepción claro de los parámetros adicionales de ancho y alto.

La Fig. 9.12 muestra el resultado esperado en el navegador.



Fig. 9.12 Página que dibuja y escala una imagen en el elemento Canvas.



En los materiales adicionales encontrará los archivos:
canvas_image_resize.html y *ClockworkOrange_small.jpg*

9.5.3 Recortar y dibujar parte de una imagen con `drawImage()`

El último modo de invocar al método `drawImage()` es un poco más complejo, ya que debe de indicar nueve datos para poder recortar y colocar la imagen antes de dibujarla realmente en el Canvas. La sintaxis es:

```
drawImage(objetoImagen, imgX, imgY, imgAncho, imgAlto, x, y, ancho, alto)
```

- **Parámetro *objetoImagen*.** Indica la imagen que se desea incluir dentro del Canvas.
- **Parámetro *imgX*.** Establece la posición de izquierda a derecha relativa a la imagen origen.
- **Parámetro *imgY*.** Establece la posición de arriba hacia abajo relativa a la imagen origen.
- **Parámetro *imgAncho*.** Indica el ancho de la selección realizada relativa a la imagen de origen.
- **Parámetro *imgAlto*.** Indica el alto de la selección realizada relativa a la imagen de origen.
- **Parámetro *x*.** Establece la posición de izquierda a derecha relativa al contenedor para colocar la imagen cortada.
- **Parámetro *y*.** Establece la posición de arriba hacia abajo relativa al contenedor para colocar la imagen cortada.
- **Parámetro *ancho*.** Es el ancho deseado para la imagen cortada que se desea incluir dentro del Canvas.
- **Parámetro *alto*.** Es la altura deseada para la imagen cortada que se desea incluir dentro del Canvas.

Observe con detenimiento la siguiente función que implementa esta modalidad del método `drawImage()`:

```
function dibuja(){  
    var micanvas = document.getElementById('miCanvas');  
    var contexto = micanvas.getContext('2d');  
    var img = new Image();  
    var img2 = document.getElementById('clockwork');
```



```
//llamando a la imagen dinámicamente  
img.src = 'ClockworkOrange.jpg';  
contexto.drawImage(img, 100, 135, 200, 70, 10, 10, 300, 95);  
}
```

En lugar de describir los pasos realizados para crear este ejemplo, será más claro utilizar la Fig. 9.13 como ilustración para mostrarlo de manera gráfica. En la parte izquierda está la imagen que se utiliza, y en la parte derecha el resultado del corte dentro del área descrita para el Canvas. Ponga especial atención en los nombres de los parámetros:

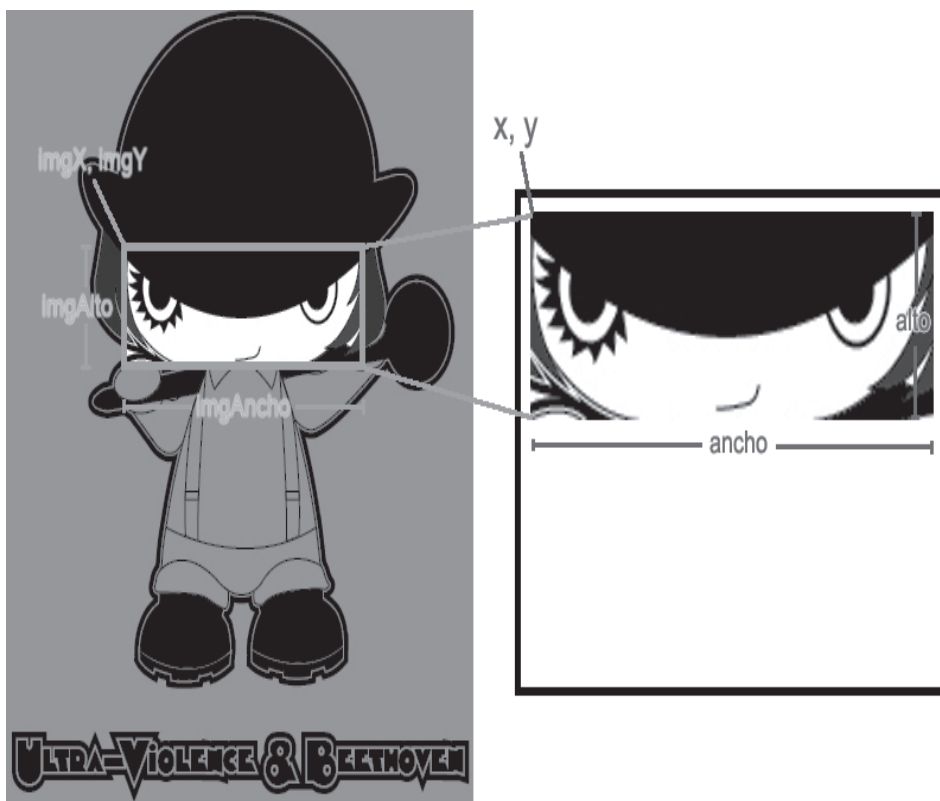


Fig. 9.13 Se muestra el uso de los parámetros para cortar y colocar imágenes.

En la Fig. 9.14 se muestra el aspecto que debe tener la imagen cortada cuando finalmente la coloca dentro del lienzo indicado por <canvas> en el navegador.



Fig. 9.14 Página que dibuja imágenes en el elemento Canvas.



En los materiales adicionales encontrará los archivos:
canvas_image_cut.html
ClockworkOrange.jpg

9.6 Transformaciones

Las transformaciones son operaciones matemáticas que se aplican a cualquier dibujo o imagen que se encuentra en el área del Canvas. Se cuenta con tres tipos de transformaciones básicas:

- **Traducción.** Mueve un elemento de un punto a otro.
- **Rotación.** Rota sobre un punto específico.
- **Escala.** Cambia el tamaño del objeto X y el objeto Y.

Quizá ya haya notado que la manipulación precisa de los gráficos dentro del elemento `<canvas>` se acerca más a las matemáticas que al diseño gráfico. Cuando transforma un objeto realmente no lo mueve, rota o escala como lo haría un paquete gráfico, en realidad, lo que hace es modificar coordenadas y grados, de manera que obtiene el resultado deseado, en otras palabras, está manipulando elementos gráficos a un nivel más profundo.

Para llevar a cabo estas operaciones de manera un poco más fácil en el contexto de Canvas, la mejor manera es manipular elementos en un sistema de coordenadas independiente, ya sea para cada elemento o por grupos, esto con el fin de aplicar determinadas transformaciones sólo al elemento o elementos que le interesan sin afectar a otros. Antes de entrar de lleno a las transformaciones se expondrá más acerca de este mecanismo.

9.6.1 Guardar y restaurar el estado

Antes de examinar los métodos de transformación, conozca dos métodos que serán indispensables una vez que comience a generar dibujos de mayor complejidad:

```
save()  
  
restore()
```

Los métodos `save()` y `restore()` se utilizan para guardar y recuperar un sistema de coordenadas. Puede ver un sistema de coordenadas dentro de un Canvas como una fotografía instantánea de todos los estilos y transformaciones que se han aplicado en él. Ambos métodos no requieren parámetros.

Cuando se llama al método `save()` todos los dibujos en el sistema de coordenadas actual se guarda en una pila o matriz de datos, esto incluye las transformaciones, estilos y valores en general que se hayan utilizado.

Puede llamar al método `save()` cuantas veces sea necesario. El método `restore()` recuperará el último estado guardado y todos los datos almacenados, en esa posición de la pila se restauran. Un ejemplo sencillo para mostrar esta lógica:

```
function dibuja(){  
    var micanvas = document.getElementById('miCanvas');  
    var contexto = micanvas.getContext('2d');  
  
    //rectángulo con estado inicial  
    contexto.fillStyle = '#FFAC42';  
    contexto.fillRect(0,0,200,200);  
    //guarda el estado inicial del sistema de coordenadas  
    contexto.save();  
  
    //se hacen los cambios deseados  
    contexto.fillStyle = '#6600FF';  
    contexto.fillRect(10,10,150,150);  
    //guarda el estado actual (con los cambios indicados)  
    contexto.save();
```

```
//una vez más se hacen los cambios deseados
contexto.fillStyle = '#eeee33';
contexto.fillRect(20,20,112,112);

//restaura el último estado guardado
contexto.restore();

//dibuja rectángulo con los cambios hechos en el estado
restaurado
contexto.fillRect(30,30,84,84);

//restaura el estado inicial
contexto.restore();

//dibuja rectángulo con los valores de estado inicial (que es el
estado restaurado)
contexto.fillRect(40,40,63,63);

}
```

Lo que sucede en este código se puede resumir en los pasos siguientes:

1. Se dibuja un rectángulo lleno convencional con un color específico, en este caso es un color de tono naranja.
2. Se guarda el estado con el método `save()`. Es aquí cuando comienza a hacer cambios en un nuevo sistema de coordenadas, ya que el original ha quedado guardado en la memoria.
3. Se dibuja un nuevo rectángulo con un color morado en el nuevo sistema de coordenadas que se obtuvo desde el paso anterior.
4. Una vez más se guarda el sistema de coordenadas actual con el método `save()`.
5. Se dibuja otro nuevo rectángulo con color amarillo en el nuevo sistema de coordenadas que se obtuvo desde el paso anterior.

6. Ahora se utiliza por primera vez el método `restore()`. Con esto se recupera el último sistema de coordenadas guardado. Una vez restaurado un estado se elimina de la pila de almacenamiento.
7. Se dibuja un nuevo rectángulo de color morado, pero note que este rectángulo ya no requiere de especificar de nuevo el color de relleno. Esto sucede porque el sistema de coordenadas restaurado contiene todos los cambios realizados en él.
8. Por segunda vez se llama al método `restore()` para restaurar al primer estado guardado que encuentre, que en este caso será el estado del sistema de coordenadas original.
9. Una vez más se dibuja un rectángulo que no requiere especificar el color de relleno, ya que toma el que se indicó en el estado del sistema de coordenadas restaurado, es decir, el rectángulo vuelve a ser naranja.

La Fig. 9.15 muestra el resultado del código descrito.

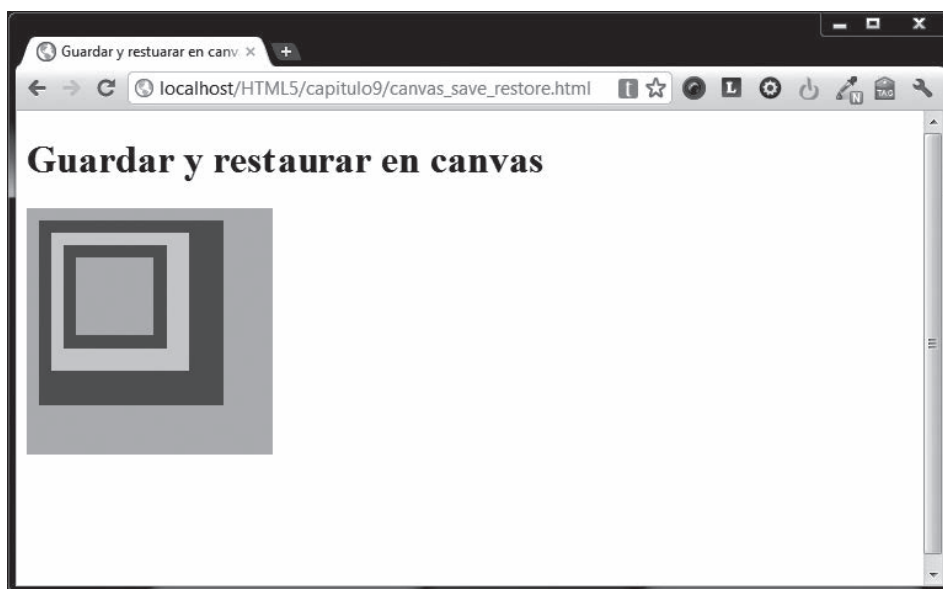


Fig. 9.15 Página que guarda y restaura estados en el elemento Canvas.



En los materiales adicionales encontrará el archivo:
canvas_save_restore.html

9.6.2 translate ()

El primero de los métodos de transformación que se analizará será `translate()`. Este método se utiliza para mover el sistema de coordenadas actual y su contenido a un punto diferente. Su sintaxis es simple:

```
translate(x, y)
```

Sólo usa dos argumentos: `x` sirve para elegir el nuevo destino del eje de coordenadas con base en la línea horizontal de izquierda a derecha, y el parámetro `y` para determinar la nueva posición vertical de arriba hacia abajo.

Lo mejor que puede hacer cuando desea usar una transformación es guardar el estado del sistema de coordenadas de manera previa, ya que en la mayoría de los casos, es más fácil llamar al método `restore()` que hacer un `translate()` con valores para volver al estado original. También si usted está usando `translate()` en un bucle y no guardar y restaura el estado actual del sistema de coordenadas en el Canvas, se puede perder parte de su dibujo, ya que podría quedar fuera de la cuadrícula.

Observe el siguiente código:

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');
var img = new Image();

//se guarda el sistema de coordenadas original
contexto.strokeStyle = "orange";
contexto.save();

//mover sistema de coordenadas
contexto.translate(50, 50);

//llamando a la imagen dinámicamente
img.src = 'ClockworkOrange_small.jpg';
contexto.drawImage(img, 0, 0);
contexto.restore();
contexto.strokeRect(0, 0, 300, 300);
}
```

Lo que sucede en este ejemplo es que después de crear todo lo necesario para dibujar es:

1. Se establece un estilo de línea perimetral de color naranja.
2. Se guarda el estado con el método `save()`.
3. Se llama al método `translate()` y se ubica al sistema de coordenadas en 50,50
4. Se dibuja una imagen, note que se utilizan los valores 0,0 para los parámetros `x`, `y`. Esto, sin embargo, equivale a dibujar una imagen en 50,50 en el sistema de coordenadas original, pero dado que hemos trasladado la actual cuadrícula a 50,50, el valor apropiado en este nuevo contexto es 0,0.
5. Se dibuja el perímetro de un rectángulo abarcando toda el área del elemento `<canvas>`, pero note que previamente se restauró el sistema de coordenadas original con el método `restore()`. De no haber restaurado previamente se hubiera perdido parte de este rectángulo, ya que se hubiera ubicado en las coordenadas de la cuadrícula con transformación y no en la original.

La Fig. 9.16 muestra el resultado del código descrito.



Fig. 9.16 Página con transformación *Translate*.



En los materiales adicionales encontrará el archivo:
canvas_translate.html

9.6.3 rotate ()

El segundo método de transformación es `rotate()`. Se utiliza para girar el lienzo alrededor de la posición 0,0 del sistema de coordenadas actual. Su sintaxis es bastante sencilla:

```
rotate(ángulo)
```

Este método sólo toma un parámetro que representa el ángulo del sistema de coordenadas que se gira. Éste es un giro en el sentido de las manecillas del reloj y se mide en radianes.

El punto central de la rotación es siempre el origen 0,0 del sistema de coordenadas. Para cambiar el punto central, tendrá que mover el lienzo mediante el uso del método `translate()`.

Vea y analice el siguiente código:

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');
var img = new Image();

//se guarda el sistema de coordenadas original
contexto.strokeStyle = "orange";
contexto.save();

//mover sistema de coordenadas
contexto.translate(50, 50);

//rotar 315 grados
contexto.rotate(Math.PI * 1.75);

//llamando a la imagen dinámicamente
img.src = 'ClockworkOrange_small.jpg';
contexto.drawImage(img, 0, 0);
```



```
contexto.restore();  
  
contexto.strokeRect(0, 0, 300, 300);  
  
}
```

El ejemplo es igual al que se utiliza para el método `translate()` pero inmediatamente después de mover el sistema de coordenadas se ha añadido el método `rotate()`, que en este caso girará la imagen 315 grados a la derecha.

La Fig. 9.17 muestra cómo se ve este ejemplo en el navegador.



Fig. 9.17 Página con transformación *Rotate*.



En los materiales adicionales encontrará el archivo:
canvas_rotate.html

9.6.4 scale ()

El siguiente método de transformación es `Scale()`. Sirve para aumentar o disminuir las dimensiones del sistema de coordenadas actual. El método puede ser utilizado para dibujar versiones reducidas o ampliadas de figuras vectoriales y mapas de bits. Su sintaxis es:

```
scale(escalaX, escalaY)
```

Este método tiene dos parámetros. *escalaX*, que es el factor de escala en la dirección horizontal y *escalaY* es el factor de escala en la dirección vertical. Ambos parámetros deben ser números reales, y no necesariamente positivos. Valores inferiores a 1.0 reducen el tamaño del sistema de coordenadas y los valores mayores a 1.0 lo aumentan de tamaño. Establecer el factor de escala con valor 1.0 no afecta el tamaño. El uso de números negativos puede hacer un efecto de espejo (por ejemplo, utilizando `translate(0, canvas.height)` y `scale(1, -1)`, usted tendrá al sistema de coordenadas actual, con origen en la esquina inferior izquierda).

Por defecto una unidad en el Canvas es exactamente un pixel. Si se aplica, por ejemplo, un factor de escala de 0.5, la unidad resultante se convertiría en 0.5 pixeles y por lo tanto las figuras se establecerían en la mitad del tamaño. De manera similar el ajuste del factor de escala a 2.0 incrementaría el tamaño de la unidad y una unidad actual se convierte en dos pixeles. Esto se traduce en figuras que se dibujan dos veces más grandes.

Considere como ejemplo el siguiente código:

```
function dibuja(){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');
var img = new Image();

//se guarda el sistema de coordenadas original
contexto.strokeStyle = "orange";
contexto.save();

//mover sistema de coordenadas
contexto.translate(50, 50);

//escala el sistema de coordenadas 50% más
```

```
contexto.scale(1.5,1);

//llamando a la imagen dinámicamente
img.src = 'ClockworkOrange_small.jpg';
contexto.drawImage(img, 0, 0);

contexto.restore();

contexto.strokeRect(0, 0, 300, 300);

}
```

El ejemplo también se utilizó para el método `translate()`, pero en este caso después de mover el sistema de coordenadas se agrega el método `scale()`, que aumentará un 50% más de longitud a lo ancho al sistema de coordenadas actual.

La Fig. 9.18 muestra cómo se ve este ejemplo en el navegador.



Fig. 9.18 Página con transformación Scale.

Actividades para el lector

1. En un nuevo documento HTML5 coloque de manera centrada una imagen dentro de un elemento `<canvas>`.
2. Aplique transformaciones de movimiento, rotación y escala a la imagen insertada.



En los materiales adicionales encontrará el archivo:
`canvas_scale.html`

9.7 Animación

Una de las grandes preguntas cuando se trata este tema en HTML5 es si esto es lo suficientemente bueno para sustituir a Flash como recurso para crear animaciones y juegos. Podrá escuchar o leer muchas opiniones al respecto, pero la realidad es que la respuesta está aún por contestarse.

A diferencia de HTML5, Flash fue concebido desde el principio como una herramienta para crear animaciones, que poco a poco ha ido adquiriendo mayor poder vía programación. Canvas, por otro lado, se diseñó para controlar su contenido vía la programación con un script desde el principio, por lo que hay limitaciones en cuanto a la animación pura se refiere.

Quizá la mayor limitación es que una vez que se ha dibujado alguna figura, se mantiene en su forma original. Si tiene que mover la figura, entonces tiene que volver a dibujarla junto con todo lo que había dibujado antes. Por tanto, se necesita mucho tiempo para volver a dibujar los fotogramas complejos, además de que los resultados dependen en gran medida del rendimiento del equipo de cómputo que los ejecute en ese momento.

Los pasos básicos para realizar una animación son:

1. **Inicialización.** Es la creación de todos los recursos necesarios, como imágenes de fondo o cualquier otro objeto que se vaya a utilizar. Los objetos a los cuales se les hace algún tipo de manipulación en tiempo real comúnmente se les llama *sprites*. Normalmente el proceso de inicialización tiene lugar la primera vez que el código se ejecuta para animar y se aprovecha para definir valores que no cambiarán a lo largo de la animación.
2. **Velocidad de los fotogramas (frame rate).** Las animaciones en general trabajan llamando a un cuadro o fotograma (*frame* en inglés) cada cierta cantidad de tiempo. En HTML5 por lo general usted llamará a una función repetidamente para formar un fotograma. La función que usará por lo general es `setInterval ()` con la cual llamará a otra función cada cierto intervalo de tiempo. Entre más corto sea este intervalo más alto será la

velocidad en que llama al fotograma, es decir, su *frame rate* será más alto. Debe tener en cuenta que a mayor *frame rate* más recursos de cómputo consumirá.

3. **Evaluación y creación del contenido del fotograma.** Debe analizar qué objetos va a incluir en el fotograma actual y por qué. Una vez tomadas las decisiones generales, proceda a colocar los objetos pertinentes.
4. **Guardar el estado.** Si desea que algo de lo que ha hecho en el fotograma actual persista en los siguientes debe salvar el estado.
5. **Limpiar el nuevo fotograma.** Si ha cambiado de fotograma para iniciar uno nuevo, normalmente comenzará limpiándolo para deshacerse de elementos del fotograma anterior.
6. **Dibujar nuevamente *sprites* y/o restaurar el estado guardado.** Puede restaurar un estado anterior para recuperar algo que le interesa mantener en el fotograma nuevo (como una imagen de fondo). Enseguida vuelva a dibujar los *sprites* que deben sufrir alguna modificación de localización, orientación, etcétera.

9.7.1 Control de una animación

Las figuras se dibujan en el área `<canvas>` mediante el uso directo de los métodos del objeto Canvas o llamando a funciones personalizadas. En circunstancias normales, sólo verá que los resultados aparecen en el Canvas cuando el script finaliza la ejecución. Por ejemplo, no es posible hacer una animación desde un bucle.

Por lo anterior, necesita una manera de ejecutar sus funciones de dibujo en un periodo. Existen dos formas de controlar una animación como ésta. Tenemos las funciones `setInterval()` y `setTimeout()` que se pueden utilizar para llamar a una función específica durante un periodo determinado. La sintaxis de estas funciones es:

```
setInterval(funciónDibuja, milisegundos);  
setTimeout(funciónDibuja, milisegundos);
```

Ambas funciones utilizan los mismos parámetros:

- **Parámetro *funciónDibuja*.** Indica a qué función debe llamarse para dibujar elementos.
- **Parámetro *milisegundos*.** Para `setInterval()` establece cada cuántos milisegundos se llama a la función indicada. Para `setTimeout()` indica después de cuántos milisegundos se llamará una sola vez a la función indicada.

Si no desea ninguna interacción con el usuario, lo más adecuado es utilizar la función `setInterval()` que ejecuta repetidamente el código de la función a la que llama.

El segundo método se puede utilizar para controlar una animación respondiendo a un evento disparado por el usuario. Si deseara hacer un juego, podría utilizar los eventos del teclado o el ratón para controlar la animación. Preparando configuración de eventos para la captura de cualquier interacción del usuario puede ejecutar sus funciones de animación.

En los siguientes ejemplos se utiliza el primer método para controlar la animación.

Ejemplo 1

Observe el siguiente código ejemplo donde se creará un reloj con una manecilla rotando al marcar los segundos:

```
var reloj = new Image();
var manecilla = new Image();
function inicia(){
    reloj.src = 'reloj_base.png';
    manecilla.src = 'reloj_manecilla1.png';
    setInterval(dibuja,100);
}
function dibuja(){
    var micanvas = document.getElementById('miCanvas');
    var contexto= micanvas.getContext('2d');
    var img = new Image();

    //para que las nuevas figuras aparezcan detrás de las más nuevas
    contexto.globalCompositeOperation = 'destination-over';
    //para limpiar el canvas
    contexto.clearRect(0,0,300,300);

    contexto.fillStyle = 'rgba(0,0,0,0.4)';
    contexto.strokeStyle = 'rgba(0,153,255,0.4)';
    contexto.save();
    contexto.translate(150,148);

    // Manecilla
    var tiempo = new Date();
    /*
```

```
Math.PI*2      es igual a una vuelta completa
Math.PI*2/60   para obtener la fraccion de grados por un segundo
multiplica los grados por segundo por los segundos actuales
según el la hora enviada por el navegador (tiempo.getSeconds())
*/
contexto.rotate( (Math.PI*2/60) * tiempo.getSeconds());
contexto.drawImage(manecilla,0,0);

contexto.restore();
contexto.drawImage(reloj,39,39);

}
```

Observe que en el código anterior se usan dos funciones: `inicia()` y `dibuja()`. A diferencia de los ejemplos anteriores, el evento `onload` del elemento `<body>` llama a `inicia()` en lugar de `dibuja()`, para así cumplir con los dos primeros pasos descritos anteriormente.

El código en la función `dibuja()` no hace nada nuevo en realidad, la única novedad es que se utiliza una fórmula un poco más elaborada para determinar el ángulo en radianes para la transformación `rotate()` con base en la hora proporcionada por el navegador, con el uso del objeto `Date()` de JavaScript y su método `getSeconds()` para extraer los segundos de la hora obtenida.

La Fig. 9.19 muestra el código del ejemplo en ejecución.

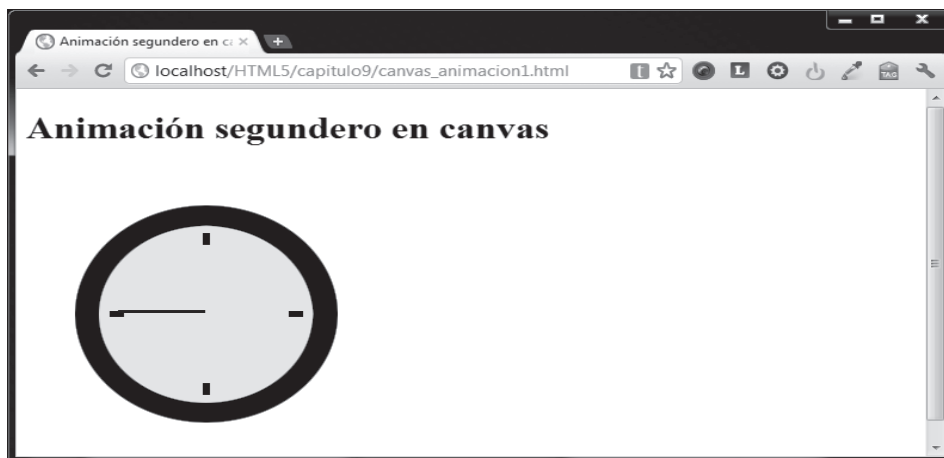


Fig. 9.19 Página que muestra una animación en Canvas.



En los materiales adicionales encontrará los archivos:
canvas_animacion1.html
reloj_base.png
reloj_manecilla1.png

Ejemplo 2

En este ejemplo verá cómo se dibujan veinte círculos para crear el efecto de burbujas flotando:

```
var width = 300;
var height = 300;
var items = new Array();

function burbuja(x,y,w,h,paso){
    this.x = x;
    this.y = y;
    this.w = w;
    this.h = h;
    this.paso = paso;
}

function inicia(){
    for (i = 0; i < 20; i++) {
        x = Math.floor(Math.random()*(width + 1)); // 0-width;
        y = Math.floor(Math.random()*(height + 1)); // 0-height;
        w = Math.floor(Math.random()*30) + 1; // 1-30;
        h = Math.floor(Math.random()*51) + 50; // 50-100
        paso = Math.floor(Math.random()*9) - 4; // -4 - 4
        var esta_burbuja = new burbuja(x, y, w, h, paso);
        items.push(esta_burbuja);
    }
    setInterval(dibuja,100);
}

function dibuja(){
    var micanvas = document.getElementById('miCanvas');
    var contexto= micanvas.getContext('2d');

    //para limpiar el canvas
```



```
contexto.clearRect(0,0,300,300);

contexto.fillStyle = "orange";
for (i in items) {

    contexto.beginPath();
    contexto.arc(items[i].x, items[i].y, items[i].w, 0, Math.PI*2,
true);
    contexto.fill();

    items[i].y = items[i].y + items[i].paso;
    if (items[i].y > height + items[i].w)
        items[i].y = -items[i].w;
    else if (items[i].y < -items[i].w)
        items[i].y = height + items[i].w;
}
}
```

Este ejemplo muestra lo que es posible hacer sólo redibujando los elementos que desee sin guardar ni restaurar ningún elemento de fotogramas anteriores. En este caso se dibujan una serie de círculos de color naranja con algunas propiedades al azar, que varían su posición vertical en el eje Y.

La Fig. 9.20 muestra el ejemplo en el navegador.



Fig. 9.20 Página que muestra una animación en Canvas.



En los materiales adicionales encontrará el archivo:
canvas_animacion2.html

Ejemplo 3

Finalmente, en el último ejemplo se utiliza la función `setTimeout ()` para controlar el tiempo que debe de pasar antes de dibujar un nuevo fotograma. En este caso se tiene la imagen de una mira de tiro que reacciona cuando el usuario mueve las teclas direccionales:

```
window.addEventListener('keydown', aplicaKeyDown, true)

var width = 300;
var height = 300;
var mira = new Image();
var fondo = new Image();
var miraObj = {x:100, y:100};

function inicia(){
    mira.src = 'target-100.png';
    fondo.src = 'duck_hunt_wallpaper_1600x1200.png';
    dibujaMira();
}

function dibujaMira(){
    var micanvas = document.getElementById('miCanvas');
    var contexto = micanvas.getContext('2d');
    contexto.drawImage(mira , miraObj.x, miraObj.y);
```

```
}

function mueveMira(dir){
var micanvas = document.getElementById('miCanvas');
var contexto = micanvas.getContext('2d');
    contexto.clearRect(miraObj.x,miraObj.y,100,100);

    if(dir == 'arriba'){
        if(miraObj.y > 0){
            miraObj.y -= 10;
        }
    }

    if(dir == 'abajo'){
        if(miraObj.y < (height - mira.height) ){
            miraObj.y += 10;
        }
    }

    if(dir == 'izquierda'){
        if(miraObj.x > 0){
            miraObj.x -= 10;
        }
    }

    if(dir == 'derecha'){
        if(miraObj.x < (width - mira.width) ){
            miraObj.x += 10;
        }
    }
}
```

```
    }  
  }  
  setTimeout(dibujaMira,100)  
}  
  
function aplicaKeyDown(evt){  
  //alert(evt.keyCode);  
  
  if(evt.keyCode == 38)  
    mueveMira('arriba')  
  
  if(evt.keyCode == 40)  
    mueveMira('abajo')  
  
  if(evt.keyCode == 37)  
    mueveMira('izquierda')  
  
  if(evt.keyCode == 39)  
    mueveMira('derecha')  
}
```

En el código anterior en la función `mueveMira()` se llama a `setTimeout()` para que después de un periodo de 100 milisegundos llame a la función que dibuja el siguiente fotograma.

La Fig. 9.21 muestra el código desplegado en el navegador.

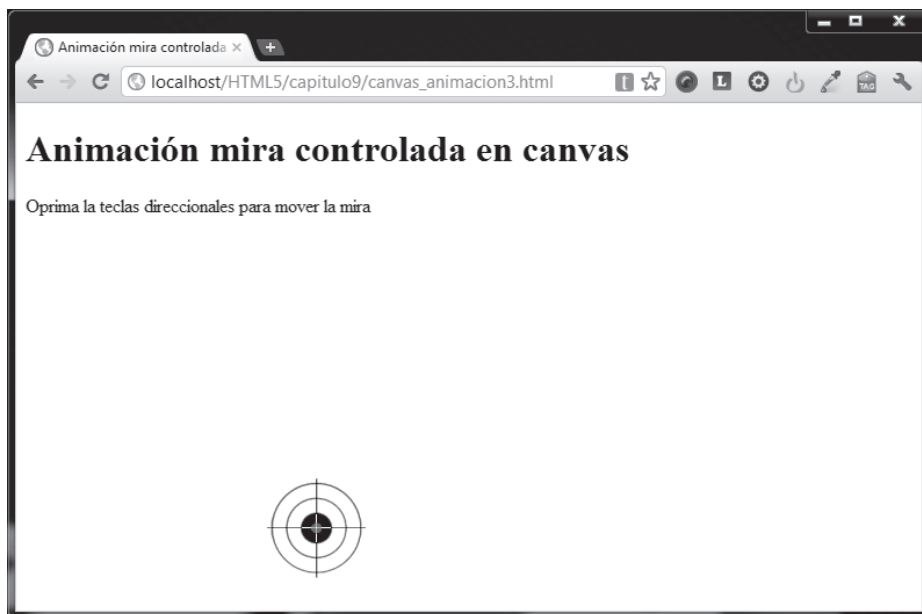


Fig. 9.21 Animación que utiliza el método `setTimeout()` en Canvas.



En los materiales adicionales encontrará los archivos:
canvas_animacion3.html
target-100.png

Actividades para el lector

1. Elabore un documento con una animación donde desplace un círculo a lo ancho del área de un elemento `<canvas>` con 700 pixeles de longitud de izquierda a derecha.
2. Usando el documento anterior como base construya otro donde el círculo en movimiento al llegar al lado derecho desaparezca y surja de nuevo por el lado izquierdo para reiniciar el recorrido.

RESUMEN

En este capítulo se han mostrado las bases indispensables para elaborar gráficos y animaciones utilizando las prestaciones del API Canvas de HTML5, cubriendo los siguientes puntos:

- La manera de crear figuras rectangulares sólo con líneas perimetrales o llenas con color.
- La manera de crear diversos tipos de caminos para crear figuras de mayor complejidad.
- El uso de imágenes dentro del elemento Canvas.
- Las transformaciones fundamentales que se pueden aplicar sobre los gráficos en un Canvas.
- Los fundamentos de las técnicas de animación utilizadas en el elemento Canvas.

Autoevaluación

1. ¿Qué es el elemento Canvas?
2. ¿Cuál es la diferencia entre las propiedades `strokeStyle` y `fillStyle`?
3. ¿Por qué no existe un método para crear círculos, como sucede con los rectángulos?
4. ¿Qué son los caminos o *paths*?
5. ¿Qué es una transformación?
6. ¿Cuál es la diferencia entre los métodos `setInterval()` y `setTimeout()`?

EVIDENCIA



Construyó una página con los elementos de curvas bezier dibujo una estrella de cinco picos rena con color verde.



Creó un documento donde dibujó una cuadrícula de 2 por 2 cuadros.



Elaboró una página con animación donde se muestra un círculo desplazándose de manera horizontal a lo largo de un Canvas de 700 pixeles de ancho.



Elaboró una versión de la actividad anterior, donde el círculo desaparece al llegar al extremo derecho del mismo Canvas y aparece de nuevo en el lado izquierdo.

REFERENCIAS

Bibliografía

Fulton, Steve, Jeff Fulton (2011), HTML5 Canvas, 1a. ed., O'Reilly, EUA.

Harris, Andy (2011). HTML5 for Dummies, 1a. ed., Willey Publishing, EUA.

Pilgrim, Mark (2010). HTML5: Up and Running, 1a. ed., O'Reilly, EUA.



Páginas Web recomendadas

<http://sixrevisions.com/html/canvas-element/>

<http://www.html5canvastutorials.com/>

http://www.w3schools.com/html5/html5_canvas.asp

<http://dev.opera.com/articles/view/html-5-canvas-the-basics/>

Respuestas sugeridas a las preguntas de autoevaluación

1. Es un elemento o etiqueta HTML5 (`<canvas>`) que utiliza JavaScript para dibujar gráficos en una página web.
2. Cuando utiliza la propiedad `strokeStyle` se define el estilo sólo para la línea perimetral de un figura, si utiliza `fillStyle` entonces se definirá el estilo de relleno de la figura.
3. En el contexto del API Canvas de HTML5 es redundante, ya que la función `arc()` establece a un círculo como un arco con ángulo de 360 grados.
4. Un *path* o camino es básicamente una secuencia de líneas que se conectan sucesivamente para formar diferentes figuras.
5. Una transformación es cuando algún elemento en un Canvas es afectado en su forma, color o posición.
6. El método `setInterval()` llamará a una función repetidas veces en el intervalo de tiempo indicado, mientras que `setTimeout()` llamará a la función señalada sólo una vez y después de que transcurra el tiempo indicado.

Arrancar con HTML5

Curso de programación

Éste no es un manual o guía para migrar aplicaciones de las versiones anteriores de HTML a HTML5, es un libro que introduce poco a poco al lector, en forma amable pero consistente con las alternativas que ofrece HTML5. Cuenta con recursos en línea y contenido adicional para respaldar mejor el aprendizaje obtenido con el texto. Es un libro que ordena ideas, conceptos e información en nuestro idioma, es una forma muy bella y eficaz de aprender HTML5.

Ventajas competitivas:

En la Web se encuentran todos los programas fuentes del libro ordenados por capítulo.

Conozca:

Las características fundamentales de HTML5.

Aprenda:

A trabajar con los nuevos elementos de HTML5.

A desarrollar páginas Web con HTML5.

Emmanuel Herrera Ríos Es Ingeniero en Sistemas Computacionales, colaborador en el desarrollo contenidos multimedia destinados a la educación, elaborados en México y en América Latina. Ha trabajado como desarrollador de diversos sistemas de información y control para diferentes empresas e instituciones. También se ha desempeñado como diseñador, desarrollador y administrador de sistemas de información financiera, ha desarrollado y coordinado proyectos multimedia para instituciones públicas a través del Instituto Latinoamericano de la Comunicación Educativa.

www.alfaomega.com.mx

ÁREA	SUBÁREA
Computación	Programación Web


Apoyo en la



ISBN 978-607-707-331-4



"Te acerca al conocimiento"

 **Alfaomega Grupo Editor**