

# Curso HTML5, CSS3 y JavaScript

**Formación Sopra Steria**

Formación HTML5, CSS3 y JavaScript

Bloque HTML5, CSS3 y JavaScript

---

Versión 1.1 del martes, 08 de Enero de 2019

# Historial

Versión	Fecha	Origen de la actualización	Realizado por	Validado por
1.0	10/12/2018		José Pedro Checa Gutiérrez, Gabriel Patricio Del Rey Apollonio, Juan Ignacio García Nicolás	
1.1	08/01/2019		José Pedro Checa Gutiérrez, Gabriel Patricio Del Rey Apollonio, Juan Ignacio García Nicolás	

# Contenidos

1.	Gestión del Ciclo de Vida de la Aplicación	6
1.1.	<b>Entendiendo los Conceptos Fundamentales</b>	<b>6</b>
1.1.1.	Breve Historia de HTML5	8
1.1.2.	¿Qué es el nuevo HTML5?	9
1.1.3.	Creando Aplicaciones	9
1.1.4.	Explorando el Empaquetado y el Entorno del Tiempo de Ejecución	11
1.1.5.	Entendiendo las Credenciales y Tipos de Permisos	12
1.2.	<b>Entendiendo y Gestionando el Estado de las Aplicaciones</b>	<b>13</b>
1.2.1.	Almacenamiento de Datos de Estado mediante Almacenamiento Local y de Sesión	13
1.3.	<b>Depurando y Probando las Aplicaciones de HTML5</b>	<b>14</b>
1.3.1.	Validación del Código HTML5	14
1.3.2.	Validación del Empaquetado	15
1.4.	<b>Publicando una Aplicación en el Tienda de Aplicaciones</b>	<b>15</b>
2.	Construyendo la Interfaz de Usuario (UI) Usando HTML5: Textos, Gráficos y Multimedia	16
2.1.	<b>Entendiendo HTML Esencial</b>	<b>16</b>
2.1.1.	Etiquetado Básico y Estructura de la Página	16
2.2.	<b>Elección y configuración de las etiquetas de HTML5 para mostrar textos</b>	<b>21</b>
2.2.1.	Elementos de texto de HTML4 con nuevos significados y nuevas funcionalidades	21
2.2.2.	Nuevos elementos de texto en HTML5	22
2.2.3.	Elementos de texto que no se usan en HTML5	23
2.3.	<b>Elección y configuración de las etiquetas de HTML5 para gráficos</b>	<b>25</b>
2.3.1.	Usando el figure y elementos de tipo <i>figcaption</i>	26
2.3.2.	Creando gráficos con canvas	28
2.3.3.	Creando gráficos con SVG	29
2.3.4.	Cuando se usa canvas en lugar de SVG	30
2.4.	<b>Elección y configuración de etiquetas de HTML5 para Play Media</b>	<b>30</b>
2.4.1.	Entendiendo y usando etiquetas de vídeo	30
2.4.2.	Entendiendo y usando etiquetas de audio	32
3.	Construyendo la Interfaz de Usuario con HTML5: Organización, Elementos de Entrada y Validación	33
3.1.	<b>Eligiendo y Configurando Etiquetas de HTML5 para Organizar Contenido y Formularios</b>	<b>33</b>
3.1.1.	Entendiendo la sintaxis de HTML	33
3.1.2.	Usando Etiquetas para Añadir Estructura a un Documento de HTML	34
3.1.3.	Usando Etiquetas para Crear Tablas y Listas	40



<b>3.2. Eligiendo y Configurando Etiquetas de HTML5 para Elementos de Entrada y Validación</b>	<b>44</b>
3.2.1. Entendiendo los Elementos de Entrada y Formularios	46
a. Explorando Formas de Creación, Atributos para Elementos de Entrada y Valores	51
3.2.2. Entendiendo las Validaciones	54
<b>4. Entendiendo CSS Básico: Contenido Flotante, Posicionamiento y Estilos</b>	<b>57</b>
<b>4.1. Breve Historia de CSS</b>	<b>57</b>
<b>4.2. Entendiendo CSS</b>	<b>57</b>
<b>4.3. Usando las Herramientas Apropriadas</b>	<b>58</b>
<b>4.4. Explorando la Conexión entre HTML y CSS</b>	<b>59</b>
<b>4.5. Separando Contenido de Estilo</b>	<b>59</b>
<b>4.6. Entendiendo Selectores y Declaradores</b>	<b>59</b>
<b>4.7. Entendiendo Fuentes y Familias de Fuentes</b>	<b>61</b>
<b>4.8. Manejando Contenido Flotante</b>	<b>61</b>
<b>4.9. Posicionando Elementos Individuales</b>	<b>62</b>
4.9.1. Aplicando Posicionamiento Flotante	62
4.9.2. Aplicando Posicionamiento Absoluto	62
<b>4.10. Manejando Contenido Desbordante</b>	<b>63</b>
4.10.1. Entendiendo Scrolling Desbordante	63
4.10.2. Entendiendo el Desbordamiento Visible y Oculto	63
<b>5. Entendiendo CSS Básico: Layouts</b>	<b>64</b>
<b>5.1. Organizando el Contenido de la Interfaz de Usuario (UI) usando CSS</b>	<b>64</b>
5.1.1. Usando Flexbox para Layouts simples y usando Grid para Layouts complejos	65
<b>5.2. Usando un Flexible Box para Establecer Contenido de Alineación, Dirección y Orientación</b>	<b>67</b>
5.2.1. Trabajando con Flexboxes y Flexbox Items	68
<b>5.3. Usando Grid Layouts para Establecer Contenido de Alineación, Dirección y Orientación</b>	<b>75</b>
5.3.1. Creando una cuadrícula usando propiedades CSS para filas y columnas	76
5.3.2. Entendiendo Grid Templates	78
<b>6. Manejando Texto Flotante usando CSS</b>	<b>80</b>
<b>6.1. Manejando el Flotamiento del Contenido de Texto usando CSS</b>	<b>80</b>
<b>6.2. Entendiendo y usando Regions para Contenido de Texto entre Múltiples Secciones</b>	<b>80</b>
6.2.1. Contenido Flotante a través de Contenedores Dinámicos	81
6.2.2. Usando columnas y unión con guión para optimizar la lectura del texto	84
6.2.3. Usando CSS Exclusions para crear texto flotante alrededor de un objeto flotante	90
<b>7. Entendiendo JavaScript y Fundamentos de Programación</b>	<b>93</b>
<b>7.1. Breve Historia de JavaScript</b>	<b>93</b>
<b>7.2. Gestión y Mantenimiento JavaScript</b>	<b>93</b>
7.2.1. Creación y uso de Funciones	94
7.2.2. Usando jQuery y otras Librerías	95
<b>7.3. Actualizando la UI usando JavaScript</b>	<b>95</b>



7.3.1.	Localizando y Accediendo a Elementos	96
7.3.2.	Escuchando y Respondiendo a Eventos	96
7.3.3.	Mostrando elementos ocultos	97
7.3.4.	Actualizando el Contenido de Elementos	97
7.3.5.	Añadiendo Elementos	97
8.	Trabajando con Gráficos y Accediendo a Datos	98
8.1.	<b>Trabajando con Imágenes, Formas y otros Gráficos</b>	<b>98</b>
8.1.1.	Manipulando <i>Canvas</i> con <i>JavaScript</i>	98
8.2.	<b>Enviando y Recibiendo Datos</b>	<b>99</b>
8.2.1.	Transmitiendo Objetos Complejos y Parsing	100
8.3.	<b>Cargando y Guardando Archivos</b>	<b>100</b>
8.3.1.	Usando <i>AppCache</i>	101
8.3.2.	Entendiendo y Usando Tipos de Datos	101
8.4.	<b>Usando <i>JavaScript</i> para Validar Campos de Formularios Introducidos por Usuarios</b>	<b>102</b>
8.5.	<b>Entendiendo y usando <i>Cookies</i></b>	<b>102</b>
8.6.	<b>Entendiendo y usando Almacenamiento Local</b>	<b>102</b>



# 1. Gestión del Ciclo de Vida de la Aplicación

---

## 1.1. Entendiendo los Conceptos Fundamentales

Este libro se divide en 3 grandes bloques, que son *HTML5*, *CSS* y *JavaScript* y en este apartado se va a mostrar una breve introducción a cada uno de estos conceptos y se hablará un poco de la historia de cada uno:

**Hypertext Markup Language (HTML)** es un lenguaje que se usa para describir las páginas Web. Es un lenguaje para etiquetas de marcado o **markup language** y no un lenguaje de programación; **HTML** usa etiquetas como `<body>` y `<h1>` para describir partes de la página Web. Un archivo *HTML* no se ejecuta como un programa. En lugar de eso, un archivo *HTML* se interpreta por el navegador para mostrar una página Web basada en las etiquetas que hayamos usado.

Desde 1999, *HTML 4.01* había sido el standard para las páginas Web, pero el mundo ha cambiado mucho desde entonces. Los usuarios de la Web, buscan aplicaciones multimedia que incorporen audio, vídeo y muchas otras interacciones con los sitios Web que ellos visiten. Y con el aumento de la popularidad de los dispositivos móviles como pizarras táctiles, tablets y smartphones, los usuarios quieren experimentar e interactuar en sus aplicaciones Web, de acuerdo con los dispositivos que poseen y han elegido para hacerlo. Todo ésto ha llevado a la necesidad de un nuevo estándar, que será *HTML5*. El World Wide Web Consortium (W3C) es el principal organismo de normalización que desarrolla especificaciones para *HTML5*. El logo de *HTML5* es el siguiente:



Un punto importante que se debe recordar sobre *HTML5* es que es un estándar formado por una combinación o familia de nuevas etiquetas de marcado *HTML*, *CSS*, *JavaScript* y otras tecnologías.

**Cascading Style Sheets (CSS)** definen estilos para *HTML* en un archivo separado, de modo que se pueden cambiar fácilmente las fuentes, el tamaño de fuente, y otros atributos en un archivo *CSS*, y los cambios se reflejan a través de todos los archivos *HTML* que hacen referencia al archivo *CSS*. La última versión de *CSS* es *CSS3*.

**JavaScript** es un lenguaje de scripting (una programación que utiliza scripts y no requiere ningún compilador) que añade interactividad a las páginas Web. JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.



Técnicamente, *JavaScript* es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con *JavaScript* se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, *JavaScript* no guarda ninguna relación directa con el lenguaje de programación *Java*. Legalmente, *JavaScript* es una marca registrada de la empresa *Sun Microsystems*, como se puede ver en <http://www.sun.com/suntrademarks/>.

Aunque se puede usar **HTML5**, **CSS3** y **JavaScript** para crear páginas Web, también se puede usar ésta combinación para desarrollar aplicaciones para clientes que se ejecuten en los dispositivos táctiles como ordenadores, pizarras táctiles, tablets y smartphones. Esencialmente, la misma tecnología que los desarrolladores utilizan para construir las páginas web, está empezando a ser utilizada para construir aplicaciones que se ejecuten en diferentes tipos de dispositivos.

*HTML5* es también una plataforma independiente. Esto quiere decir, que se pueden crear apps (aplicaciones) usando el conjunto de tecnologías de *HTML5* y ejecutarlas en los diferentes sistemas operativos de escritorio y dispositivos móviles, como *Microsoft Windows*, *Internet Explorer* y *Windows Phone*. Incluso se pueden ejecutar en *Mac OS X*, *Android*, *IOs* y *Blackberry OS*. Como *HTML5* está construido sobre una base abierta, los usuarios de aplicaciones *HTML5* no tienen que descargar un *plug-in* ni utilizar dispositivos que tengan soporte para *plug-ins*. En su lugar, se puede utilizar cualquier navegador Web, ya sea en su ordenador o dispositivo móvil, y obtener la misma experiencia web.



Finalmente, una parte importante del desarrollo de una aplicación en un entorno de Windows es la **Metro Style User Interface (UI)** el cual es el interfaz de usuario usado por *Windows 8*. La *Metro style UI* incluye características como apariencia y tacto de pantalla fluido, uso de pantalla completa o un foco para el desplazamiento lateral.

### 1.1.1. Breve Historia de HTML5

El origen de *HTML* se remonta a 1980, cuando el físico **Tim Berners-Lee**, trabajador del *CERN*(*Organización Europea para la Investigación Nuclear*) propuso un nuevo sistema de "*hipertexto*" para compartir documentos. Los sistemas de "*hipertexto*" habían sido desarrollados años antes. En el ámbito de la informática, el "*hipertexto*" permitía que los usuarios accedieran a la información relacionada con los documentos electrónicos que estaban visualizando. De cierta manera, los primitivos sistemas de "*hipertexto*" podrían asimilarse a los enlaces de las páginas web actuales.

Tras finalizar el desarrollo de su sistema de "*hipertexto*", Tim Berners-Lee lo presentó a una convocatoria organizada para desarrollar un sistema de "*hipertexto*" para Internet. Después de unir sus fuerzas con el ingeniero de sistemas **Robert Cailliau**, presentaron la propuesta ganadora llamada *WorldWideWeb* (*W3*).

El primer documento formal con la descripción de *HTML* se publicó en 1991 bajo el nombre *HTML Tags* (*Etiquetas HTML*) y todavía hoy puede ser consultado online a modo de *reliquia informática*.

La primera propuesta oficial para convertir *HTML* en un estándar se realizó en 1993 por parte del organismo *IETF* (*Internet Engineering Task Force*). Aunque se consiguieron avances significativos (en esta época se definieron las etiquetas para imágenes, tablas y formularios) ninguna de las dos propuestas presentadas como estándar, llamadas *HTML* y *HTML+* consiguieron convertirse en estándar oficial.

En 1995, el organismo *IETF* organiza un grupo de trabajo de *HTML* y consigue publicar, el 22 de septiembre de ese mismo año, el estándar *HTML 2.0*. A pesar de su nombre, *HTML 2.0* es el primer estándar oficial de *HTML*.

A partir de 1996, los estándares de *HTML* los publica otro organismo de estandarización llamado *W3C*(*World Wide Web Consortium*). La versión *HTML 3.2* se publicó el 14 de Enero de 1997 y es la primera recomendación de *HTML* publicada por el *W3C*. Esta revisión incorpora los últimos avances de las páginas web desarrolladas hasta 1996, como *applets* de Java y texto que fluye alrededor de las imágenes.

*HTML 4.0* se publicó el 24 de Abril de 1998 (siendo una versión corregida de la publicación original del 18 de Diciembre de 1997) y supone un gran salto desde las versiones anteriores. Entre sus novedades más destacadas se encuentran las hojas de estilos *CSS*, la posibilidad de incluir pequeños programas o *scripts* en las páginas web, mejora de la accesibilidad de las páginas diseñadas, tablas complejas y mejoras en los formularios.

La última especificación oficial de *HTML* se publicó el 24 de diciembre de 1999 y se denomina *HTML 4.01*. Se trata de una revisión y actualización de la versión *HTML 4.0*, por lo que no incluye novedades significativas.

Desde la publicación de *HTML 4.01*, la actividad de estandarización de *HTML* se detuvo y el *W3C* se centró en el desarrollo del estándar *XHTML*. Por este motivo, en el año 2004, las empresas *Apple*, *Mozilla* y *Opera* mostraron su preocupación por la falta de interés del *W3C* en *HTML* y decidieron organizarse en una nueva asociación llamada *WHATWG* (*Web Hypertext Application Technology Working Group*).

La actividad actual del *WHATWG* se centra en el futuro estándar *HTML 5*, cuyo primer borrador oficial se publicó el 22 de enero de 2008. Debido a la fuerza de las empresas que forman el grupo *WHATWG* y a la publicación de los borradores de *HTML 5.0*, en marzo de 2007 el *W3C* decidió retomar la actividad estandarizadora de *HTML*.

De forma paralela a su actividad con *HTML*, *W3C* ha continuado con la estandarización de *XHTML*, una versión *avanzada* de *HTML* y basada en *XML*. La primera versión de *XHTML* se denomina *XHTML 1.0* y se publicó el 26 de Enero de 2000 (y posteriormente se revisó el 1 de Agosto de 2002).





*XHTML 1.0* es una adaptación de *HTML 4.01* al lenguaje *XML*, por lo que mantiene casi todas sus etiquetas y características, pero añade algunas restricciones y elementos propios de *XML*. La versión *XHTML 1.1* ya ha sido publicada en forma de borrador y pretende modularizar *XHTML*. También ha sido publicado el borrador de *XHTML 2.0*, que supondrá un cambio muy importante respecto de las anteriores versiones de *XHTML*.

### 1.1.2. ¿Qué es el nuevo HTML5?

A continuación, se va a ver una breve lista de estas nuevas características y breves descripciones que definen lo que es *HTML5*:

- **Etiquetas de audio y vídeo:** Integra audio y vídeo usando etiquetas de *HTML5*, como `<audio>` y `<video>`.
- **Canvas:** Es un elemento *HTML5* que crea un espacio para hacer gráficos y usa *JavaScript* para dibujar las gráficas que se necesiten.
- **Media Queries (Consultas Multimedia):** Una característica de *CSS3* es que detecta el tipo de pantalla y tamaño que usa el dispositivo el usuario y adapta los outputs en función de ello.
- **Nueva aplicación para programar interfaces (APIs):** Proporciona a las aplicaciones una innumerable cantidad de accesos a muchos recursos, como archivos, webcams y animaciones aceleradas por el ordenador.
- **Geolocalización:** Utiliza *JavaScript* para detectar la localización (posicionamiento geográfico) del dispositivo de un cliente si es un Windows Phone, *Android* phone, o un *PC*.
- **Modernizr:** Es una librería de *JavaScript* que ayuda trasladar las nuevas capacidades de *HTML5* y *CSS3* en los antiguos navegadores.

Esta pequeña muestra de las características disponibles y tecnologías. Se aprenderá a cómo usarlas en los siguientes tutoriales a lo largo del curso.

### 1.1.3. Creando Aplicaciones

A pesar de que se puede crear una gran cantidad de código usando un simple editor *HTML*, si se quiere implementar la aplicación que se haya creado, se necesitará una herramienta de desarrollo de aplicaciones, como *Microsoft Visual Studio*.

En esta sección, se van a mostrar los pasos generales involucrados en la creación de una aplicación:

- **Planificación del proyecto:** Primero se debe pensar el tipo de aplicación que se debe crear en función del tipo de proyecto. Después de decidir sobre cuál será la acción principal de la aplicación, se debe crear un esquema del flujo general de la aplicación de principio a fin. Además, se tendrá que determinar el tipo de interacción con el usuario, el hecho de si se va a tener que guardar datos fuera de la aplicación y si la aplicación debe conectar con otras aplicaciones y/o servidores.
- **Diseño de UI:** UI son las siglas en inglés de interfaz de usuario. En este paso se determinará la forma en la que se quiera que el usuario vea la aplicación, por ejemplo, listar los elementos de comandos que se necesitan para que cuando la aplicación se ejecute como se espera se carguen las imágenes y archivos multimedia necesarios.
- **Actualizar el archivo de manifiesto de la aplicación:** Toda aplicación requiere un manifiesto en el que se encuentra la información de lo que la aplicación necesita para ejecutarse y las propiedades de la aplicación.



```
<?xml version="1.0" encoding="utf-8"?>
<Package xmlns="http://schemas.microsoft.com/appx/2010/manifest">
  <Identity Name="CompanyX.Samples.App1"
    Version="1.0.0.0"Publisher="CN=Company X, O=Company X, L=Coolsville, S=TX, C=USA" />
  <Properties>
    <DisplayName>Samples</DisplayName>
    <PublisherDisplayName>Company X</PublisherDisplayName>
    <Logo>images\CompanyX-logo.png</Logo>
  </Properties>
  <Prerequisites>
    <OSMinVersion>6.2</OSMinVersion>
    <OSMaxVersionTested>6.2</OSMaxVersionTested>
  </Prerequisites>
  <Resources>
    <Resource Languages="en-us" />
  </Resources>
  <Applications>
    <Application Id="App1" StartPage="default.html">
      <VisualElements DisplayName="App1" Description="A handy little app."
        Logo="images\icon.png"SmallLogo="images\icon-sml.png"
        ForegroundText="dark" BackgroundColor="#FFFFFF"
        <SplashScreen Image="images\splash.png" />
      </VisualElements>
    </Application>
  </Applications>
</Package>
```

- **Escribir el código**: Es el momento de escribir el código de la aplicación, el cual podrá ser una mezcla de código tipo *HTML*, *CSS* y *JavaScript*.
- **Construir la aplicación**: Para ello, se necesitará una herramienta de desarrollo de aplicaciones como *Visual Studio*, donde se convertirá el código y otros recursos que se hayan usado en lo que es la aplicación.
- **Depurar y testear**: Se deberá comprobar que la aplicación hace lo que se desea y será importante probar la aplicación desde diversos emuladores de los dispositivos para los que se haya pensado la aplicación.
- **Empaquetar la aplicación**: Para este paso, se creará un contenedor, en el que se guardarán los archivos requeridos por la aplicación como los archivos de *JavaScript*, imágenes, etc...
- **Validar la aplicación**: Se ejecutará la aplicación y se asegura que está todo lo que se pretendía en la aplicación.



- **Poner en uso la aplicación:** Se pone la aplicación a la venta en alguna tienda online, como *Windows Store*. Esto implica que la aplicación deberá ser accesible desde una gran variedad de dispositivos y de manera segura.



#### 1.1.4. Explorando el Empaquetado y el Entorno del Tiempo de Ejecución

Cuando una aplicación es lanzada, es decir, se ejecuta, se considera que está en el entorno de ejecución o runtime environment (RTE). En este entorno es donde se prueba la aplicación en el momento de testeo y donde los usuarios ejecutan la aplicación.

Windows tiene su propio RTE y se llama **Windows Runtime (WinRT)**, el cual trabaja con *C#*, *C++*, *Visual Basic* y *JavaScript*. Se podrán crear aplicaciones con el estilo Metro, con la librería *WinRT* y la librería de Windows para APIs *JavaScript*. Una **application programming interface (API)**, es una lista de instrucciones que permite al programa, comunicarse con otros programas. En una aplicación Web, una API permite que un navegador Web o un servidor Web se comunique con otros programas. Hay cientos de APIs disponibles para muchos usos diferentes.

**Nota:** El *Document Object Model (DOM)*, es un tipo de API importante, la cual está diseñada por *HTML* y *Extensible Markup Language (XML)*, y permite a los programas y scripts actualizar el contenido, la estructura y los estilos de la aplicación.

La biblioteca de Windows para *JavaScript* incluye archivos *JavaScript* y *CSS* que los desarrolladores pueden usar para crear aplicaciones de estilo Metro de forma más fácil y rápida. La biblioteca se utiliza junto con *HTML*, *CSS* y *WinRT* para crear aplicaciones.

El runtime environment es responsable del acceso a los dispositivos, los medios, las redes, las redes locales y las redes de datos, almacenamiento remoto y otros artículos. Un desarrollador puede utilizar las API y en el runtime environment para solicitar el acceso a los dispositivos de usuario dentro de una aplicación. Un dispositivo puede ser un teclado, un ratón, un touchpad, una impresora, una cámara web o un micrófono.



### a. Entendiendo el Proceso Host

Tanto si una aplicación es una aplicación web como si se ha creado para *Windows*, necesita un proceso host en tiempo de ejecución para iniciarla. Por ejemplo, al iniciar *Internet Explorer*, un proceso host en el sistema operativo controla la ejecución global del navegador. En este caso, cada pestaña del navegador tiene su propio proceso, así que, si tiene tres pestañas abiertas, el sistema tiene tres procesos ejecutándose para cada una de ellas. (Un "proceso" es simplemente un programa que se está ejecutando)

Cuando se ejecuta una aplicación con estilo Metro que se crea con *JavaScript*, en el caso de *Internet Explorer*, se muestra el *HTML* de forma muy parecida a cuando se navega a una página Web, pero el navegador está alojado en un archivo proceso diferente, llamado *WWAHost.exe*. *WWAHost* pasa el *HTML*, *CSS* y *JavaScript* en la página default.

Para mantener las cosas en orden, el código que se ejecuta en un contenedor de aplicaciones de estilo Metro está restringido a ciertas indicaciones o acciones que se harán por defecto. Si se quiere que la aplicación acceda a un dispositivo, a otra aplicación, a Internet, o a cualquier otro sitio web de la empresa, hay que declararlo en el manifiesto de la aplicación, más concretamente en la sección de *Capabilities*. Cuando el usuario finalmente instale la aplicación, ésta deberá pedirle los permisos necesarios.

Los estilos Metro de las aplicaciones usan contratos y algo llamado extensiones cuando se crean interacciones entre aplicaciones. Las *APIs* de *WinRT* se encargan de la comunicación entre las aplicaciones.

### b. Entendiendo el Empaquetado de una Aplicación y el Contenedor de Aplicaciones

El empaquetado de una aplicación es el resultado de un conjunto de procesos. Empaquetar es parecido a poner varios archivos en una carpeta que los contenga.

Algunas cosas que se debe saber sobre los paquetes son las siguientes:

- Un paquete puede contener páginas Web, código, tablas de base de datos y procedimientos. Cuando un empaquetado tiene una interfaz de usuario, se le conoce como una aplicación.
- Un paquete puede contener otros paquetes.
- Se puede mover uno o más elementos dentro o fuera de un paquete. Ya que un paquete se encuentra en su propio contenedor, si se mueve un paquete, todo en el paquete se mueve como una unidad.
- Un usuario puede instalar, actualizar o eliminar un paquete.
- Un solo paquete puede tener muchas funcionalidades. Para mantener todos los componentes separados de manera que no entren en conflicto, un paquete define un **namespace**, que es como un área de trabajo para objetos relacionados (páginas, código, etc.).

### 1.1.5. Entendiendo las Credenciales y Tipos de Permisos

La seguridad del código es una prioridad para los desarrolladores de aplicaciones.

*.NET Framework 4.0* es un componente de Windows que se ejecuta en el servidor. Este componente proporciona un entorno de ejecución de código (como *JavaScript*), lo que les ayuda a ejecutarse con relativamente pocos problemas. Y, además, cuenta con la ayuda del código transparente en seguridad durante el proceso, junto con un **Conjunto de permisos** y de **Permisos de identidad**.

El **conjunto de permisos** son, básicamente, los permisos de seguridad, por los que se rigen las transparencias.



Los **permisos de identidad** protegen al ensamblado y cada permiso de identidad representa un tipo particular de evidencia, o credenciales, que un ensamblado debe tener para poder ejecutarse.

## 1.2. Entendiendo y Gestionando el Estado de las Aplicaciones

La gestión de estado es el proceso de mantener la información de la página Web durante múltiples solicitudes para la misma o diferente página web. Cuando un usuario solicita por primera vez el acceso a una aplicación se crea el estado de sesión o **sesión state**. El estado finaliza cuando el usuario cierra la sesión.

Una alternativa al estado de sesión es el estado de aplicación. Se crea el estado de la aplicación o **application state** cuando el navegador web envía la primera solicitud de una página web al servidor web, y finaliza cuando el usuario cierra el navegador.

La información de estado persistente o **persistent state information** son los datos que una aplicación necesita una vez finalizada la sesión. Muchas aplicaciones Web necesitan almacenar datos (hacerlos persistentes) para que los usuarios puedan encontrar la página tal y como se dejó en la última sesión.

### 1.2.1. Almacenamiento de Datos de Estado mediante Almacenamiento Local y de Sesión

**Nota:** El **Hypertext Transport Protocol (HTTP)** es el protocolo que transfiere datos en el directorio World Wide Web. Define las acciones que los servidores web y los navegadores toman en respuesta a comandos por parte de los usuarios. HTTP es un protocolo sin estado, lo que significa que no retiene datos de una sesión a otra. Al cerrar un navegador Web después de visitar un sitio Web, la función no guarda los datos.

Para evitar las limitaciones del protocolo *HTTP*, los desarrolladores han utilizado históricamente cookies, que son pequeños archivos que contienen información sobre el usuario y el sitio web visitado y se guardan en el ordenador del usuario. Cuando un usuario regresa a un sitio visitado, el navegador envía el archivo de cookies de vuelta al servidor web. Las cookies ayudan a un servidor web a "recordar" a un usuario y personalizar la experiencia del usuario en ese sitio.

Sin embargo, las cookies han demostrado ser un riesgo para la seguridad. Además, si se utilizan grandes cantidades de datos, estos se envían entre el navegador y el servidor cada vez que se solicita, lo que causaría una notable disminución del rendimiento para el usuario.

En *HTML5*, los desarrolladores pueden usar el almacenamiento Web (Web storage), que ofrece más flexibilidad y puede manejar conjuntos de datos más grandes, además de proporcionar un mejor rendimiento.

El método **localStorage** permite a los usuarios guardar grandes cantidades de datos de sesión en sesión (datos persistentes), y no hay límite de tiempo en cuanto a la duración de la existencia de los datos. El método **sessionStorage** guarda los datos sólo durante una sesión (hasta que se cierra el navegador), que también se conoce como "*per-tab storage*".

Utilizando estos métodos, los datos específicos se transfieren sólo cuando se solicitan, por lo que es posible almacenar una cantidad relativamente grande de datos sin ralentizar la conexión o el sitio.



#### d. APPCACHE PARA ARCHIVOS EN MODO OFFLINE

Otra forma de utilizar el almacenamiento Web es almacenar datos localmente cuando un usuario está desconectado. La aplicación *Cache*, o **AppCache**, almacena recursos como imágenes, páginas *HTML*, archivos *CSS* y *JavaScript*, datos que normalmente se almacenarían en un servidor. Como los recursos se almacenan en la base de datos del cliente el disco duro o dispositivo, los recursos se cargan más rápido cuando se solicitan. Usando *AppCache*, un desarrollador usa un archivo de texto llamado "manifiesto de caché" para especificar los archivos que el navegador de Internet deberá tener en la cache offline. Incluso si un usuario pulsa el botón *Actualizar sin conexión*, la aplicación se cargará y funcionará correctamente. Un archivo de manifiesto de caché tiene un aspecto similar al siguiente:

```
index.html  
  
stylesheet.css  
  
images/dot.png  
scripts/main.js
```

### 1.3. Depurando y Probando las Aplicaciones de HTML5

La depuración de una aplicación (**Debugging**) implica la detección, búsqueda y corrección de datos lógicos o sintácticos. Un error sintáctico es un error tipográfico en el código o un error similar, que normalmente se revela durante la ejecución para aplicaciones interpretadas. Un error lógico hace que la aplicación se comporte de forma diferente a la esperada.

Testear y depurar código es una parte estándar del desarrollo de aplicaciones. Algunos errores son fáciles de cometer, detectar y corregir, mientras que otros pueden requerir horas o incluso días para resolverse, dependiendo de la complejidad de la aplicación.

De cualquier manera, la fase de prueba y depuración es muy importante por varias razones:

- Las aplicaciones de alta calidad obtienen altas calificaciones, lo que puede aumentar sus beneficios e impulsar las ventas de futuras aplicaciones.
- Si se planea publicar una aplicación a través de la Tienda de Windows o a través de otro sitio web de confianza.
- En el mercado de aplicaciones en línea, la tienda requerirá la validación o certificación de que la aplicación ha sido probada.

#### 1.3.1. Validación del Código HTML5

Uno de los primeros pasos en la fase de depuración y testeo es validar el código *HTML5*. Validación significa verificar la validez de su código. Un **validador** busca cualquier cosa que pueda causar que el código sea interpretado incorrectamente, como etiquetas faltantes o no cerradas, una declaración *DOCTYPE* incorrecta, una barra oblicua, código obsoleto, etc.



El W3C proporciona un servicio de validación de código para todas las versiones activas de *HTML* en su Markup Validation Service: la página Web en <http://validator.w3.org/>. El servicio es gratuito. Simplemente se debe hacer click en un enlace para subir su archivo al servicio, o copiando y pegando el contenido de su archivo en un cuadro de texto en el sitio Web. A continuación, haciendo click en el botón *Check* la validación comprueba su código e informa de cualquier error o problema que necesite solucionar.

Un validador no es lo mismo que un emulador o un simulador. Un validador realmente prueba el código e informa de inexactitudes, dándole la oportunidad de hacer cambios. Emuladores y simuladores simplemente proporcionan un entorno en el que ejecutar código.

**Nota:** El W3C también ofrece un buscador de enlaces en <http://validator.w3.org/checklink>. Este servicio comprueba que todos los enlaces de su archivo *HTML* sean válidos. El Servicio de Validación de *CSS* en <http://jigsaw.w3.org/css-validator/> comprueba sus archivos *CSS*.

### 1.3.2. Validación del Empaquetado

El Kit de certificación de aplicaciones de Windows está incluido en el Kit de desarrollo de software de *Windows* (*SDK*) para aplicaciones de estilo Metro, disponibles en el sitio web de *Microsoft*. Para usar el kit, se deberá instalar la aplicación localmente usando una herramienta de desarrollo de aplicaciones. Luego se abrirá el kit, se seleccionará la aplicación que desea validar y ejecute el validador. Aparecerá un informe que indica cualquier problema con la aplicación. El Kit de certificación de aplicaciones de *Windows* también puede estar disponible como una opción de menú dentro de su herramienta de desarrollo de aplicaciones.

## 1.4. Publicando una Aplicación en el Tienda de Aplicaciones

Antes de publicar una aplicación en la Tienda de Windows, se debe hacer lo siguiente:

- Se debe registrar y pagar por una cuenta de desarrollador de la Tienda de Windows y reservar un nombre para la aplicación. También se deberá editar el archivo de manifiesto de la aplicación.
- Revisar la lista de control de envío de aplicaciones en <http://bit.ly/HAPmbk>. La lista de control incluye tareas como ponerle un nombre a la aplicación, elegir detalles de venta como seleccionar precio apropiado y una fecha de lanzamiento, asignando una clasificación por edades, describiendo su aplicación, etc...
- Utilizar el Kit de certificación de Windows App para probar la aplicación, si aún no se ha hecho.
- Realizar algunas capturas de pantalla de características, significativas o únicas de la aplicación para mostrarlas en la tienda. Se puede usar la herramienta Recortes, que está integrada en Windows 7 y Windows 8, para capturar capturas de pantalla o puede utilizar otra herramienta de su elección.
- Hacer que otros probadores o desarrolladores prueben su aplicación en tantos dispositivos y plataformas diferentes como sea posible, en la medida de lo posible, especialmente si se ha probado sólo en un simulador o emulador.
- Incluir una declaración de privacidad si la aplicación recopila información personal o utiliza información protegida por derechos de autor.



## 2. Construyendo la Interfaz de Usuario (UI) Usando HTML5: Textos, Gráficos y Multimedia

### 2.1. Entendiendo HTML Esencial

El lenguaje de marcado de hipertexto (*HTML*) se denomina lenguaje de marcado porque se utiliza para describir (marcar) piezas de contenido para mostrar en una página Web. Una página Web con marcas de revisión significa que incluye **etiquetas** o **tags**, que son palabras clave que ayudan a dar una estructura de página *HTML*.

La clave para usar *HTML* es saber qué etiquetas usar y cuándo. La combinación de contenido, etiquetas y quizás gráficos, multimedia, etc., son los que construyen una página web.

Se puede identificar fácilmente un documento *HTML* porque tiene una extensión de archivo *.htm* o *.html*.

Cuando un navegador Web o un dispositivo móvil, como un smartphone, abre un archivo *HTML*, éste **renderiza** (interpreta y reproduce) el contenido de la página.

#### 2.1.1. Etiquetado Básico y Estructura de la Página

**Nota:** Cada página *HTML* incluye etiquetas. Una etiqueta es una palabra clave rodeada de corchetes angulares (<>). La mayoría de las etiquetas vienen en pares; una etiqueta se llama etiqueta de apertura o de inicio, y la otra es la etiqueta cierre o etiqueta final. Ambas etiquetas deben ser idénticas. Una etiqueta de cierre es idéntica a una etiqueta de apertura excepto que la etiqueta de cierre incluye una barra (/) antes de la palabra clave.

Las etiquetas rodean el contenido y le dan definición, por ejemplo:

```
<h1>Cuidado de mascotas</h1>
```

*HTML* también utiliza algunas etiquetas simples, como <br/> para un salto de línea y <hr/> para una etiqueta de línea horizontal. En *HTML 4*, estas etiquetas se llaman etiquetas vacías porque no requieren una etiqueta final.

*HTML5* es menos restrictivo que *HTML 4*. No es necesario incluir etiquetas de final para todos los elementos (aunque algunos elementos todavía requieren etiquetas de inicio y final), y se puede introducir etiquetas en mayúsculas o en minúsculas. Sin embargo, este libro utiliza etiquetas de inicio y final, y todas en minúsculas para el marcado, para una mayor consistencia.

Algunas de las etiquetas más usadas son:

Etiquetas	Descripción
<html>	Identifica la página como un documento <i>HTML</i> . La etiqueta <html> abarca todo lo que hay en la página excepto la declaración doctype en la parte superior.





<b>&lt;head&gt;</b>	Contiene marcas y códigos utilizados por el navegador, como secuencias de comandos para añadir interactividad y palabras clave para ayudar a los motores de búsqueda a encontrar la página. El contenido de la etiqueta <b>&lt;head&gt;</b> también puede incluir estilos de formato para la página.
<b>&lt;title&gt;</b>	Muestra el título de la página Web, que aparece en la parte superior de la ventana de diálogo Navegador web, normalmente en la pestaña de la página en un navegador con pestañas.
<b>&lt;body&gt;</b>	Envuelve el contenido que es visible en la página Web cuando se visualiza en un Navegador web.
<b>&lt;a href=URL&gt;</b>	Generalmente se usa para anclar una URL a un texto o a una imagen; también puede crear un archivo de un documento para permitir el enlace a secciones del documento.
<b>&lt;b&gt;</b>	Aplica negrita al texto.
<b>&lt;hx&gt;</b>	Crea un encabezado, que puede ser de primer nivel ( <b>h1</b> ) a sexto nivel ( <b>h6</b> ), x puede ser cualquier valor entre 1 y 6.
<b>&lt;img&gt;</b>	Inserta una imagen de un archivo o de otro sitio Web.
<b>&lt;p&gt;</b>	Define el texto como un párrafo.

Un par de etiquetas o una etiqueta vacía también se denomina **elemento**. Un elemento puede describir el contenido, insertar y crear hipervínculos.

#### a. Usando Atributos

No todas las etiquetas describen los datos por sí solas o al menos no con suficiente detalle para el renderizado, por lo que algunas deben incluir atributos, que son modificadores de elementos *HTML* que proporcionan información adicional.

Los atributos son fáciles de usar y son sólo extensiones de elementos. Se añaden atributos a los elementos de acuerdo con esta sintaxis básica:

```
<tag attribute="value">
```

Se debe notar que el atributo y su valor están dentro de una etiqueta. Se debe incluir un atributo dentro de una etiqueta para que el navegador web sepa cómo manejar el atributo. Un buen ejemplo de atributo es cuando se crea un hipervínculo, como se indica a continuación:

```
<a href="http://www.example.com">This is a link.</a>>
```

El navegador Web utiliza la combinación del elemento ancla y el atributo **href** para mostrar un hipervínculo. A continuación, se muestra cómo interpreta un navegador web esta marca.



Dos de los usos más comunes de los atributos son la creación de hipervínculos y la inserción de atributos sencillos gráficos. Más adelante en esta lección se aprenderá a trabajar con gráficos. *HTML5* incluye varios que se pueden utilizar con cualquier elemento *HTML5*. Ejemplos de **atributos globales** incluyen **id**, **lang**, y **class**, entre muchos otros.

#### b. Anidando Elementos

La forma en que un navegador Web muestra su *HTML* depende de la forma en que combine los elementos, sus atributos (si los hay) y contenido. Cuando dos o más elementos se aplican al mismo bloque de texto, se debe anidar las parejas de marcas apropiadamente para que hagan lo que se desea. **Anidar** significa colocar un elemento dentro de otro. He aquí un ejemplo de cómo anidar correctamente:

```
<p>Asegúrese de que su mascota tenga suficiente <i><b>agua dulce</b></i>  
en tiempo caluroso.</p>
```

En este caso, se quiere que las palabras "agua dulce" destaquen para que aparezcan en cursiva y en negrita utilizando las etiquetas `<i>` y `<b>`. Si se coloca la etiqueta final `</b>` después de la etiqueta final `</p>` (se muestra más abajo), las palabras "agua dulce en tiempo caluroso" aparecerían en negrita, pero sólo "agua dulce" estaría en cursiva. El texto se mostraría de una forma extraña, como se muestra en la siguiente imagen:

```
<p> Asegúrese de que su mascota tenga suficiente <i><b>agua dulce</i>  
en tiempo caluroso.</p></b>
```

La regla para anidar es que las marcas anidadas deben ser cerradas antes que sus marcas parentales. Echando una vista atrás en el ejemplo correcto, se observa que el elemento de párrafo se abre primero, seguido de la cursiva y luego el elemento de fuente. Luego se cierra el elemento en negrita, seguido por el elemento de la cursiva, y finalmente el elemento de párrafo. Los elementos en cursiva y negrita están completamente anidados dentro del elemento de párrafo.

### c. Entendiendo las Entidades

Una entidad es un caracter especial, como el símbolo del dólar, la marca registrada (una R mayúscula dentro de un círculo) y las letras acentuadas. El proceso de incorporación de entidades en una página Web se denomina codificación de caracteres. Las herramientas de edición Web y los navegadores de hoy en día hacen un buen trabajo de manejo de caracteres especiales que aparecen en el teclado, como los de la parte superior de las teclas numéricas. En la mayoría de los casos, esos caracteres se renderizan sin problemas.

Con algunos navegadores, el carácter que se esperaba no aparece y se obtiene un carácter de galimatías o símbolo en su lugar. Esas situaciones son fáciles de manejar. Cada carácter especial que puede ser reproducido en una página Web tiene un nombre de entidad y un código numérico. Sin embargo, generalmente es más seguro representar símbolos como la marca comercial usando un número para asegurar el correcto renderizado en una amplia variedad de navegadores.

Una entidad comienza con un ampersand (&) y termina con un punto y coma (;). Por ejemplo, la entidad **&reg;** representa el símbolo de marca registrada, y su código numérico es **&#174;**.

Cuando un navegador se encuentra con un ampersand, intenta hacer coincidir los caracteres que siguen con el símbolo entidad. Si el navegador encuentra una coincidencia, muestra el carácter especial en lugar de la entidad.

En la tabla siguiente se enumeran algunas entidades de uso común.

Símbolos	Descripción	Nombre de la entidad	Código
©	Copyright	&copy;	&#169;
°	Degree	&deg;	&#176;
\$	Dollar	&dollar;	&#36;
%	Percent	&percnt;	&#37;
®	Registered trademark	&reg;	&#174;

Otra cosa importante que se debe saber sobre la codificación de caracteres en *HTML5* es que se debe utilizar codificación UTF-8 siempre que sea posible, ya que la mayoría de los navegadores utilizan UTF-8. Esto significa que se añade la siguiente declaración al elemento de cabecera:

```
<meta charset="UTF-8">
```

La especificación *HTML5* requiere que todo elemento encaje en los primeros 1.024 bytes del documento, por lo que se incluye en la parte superior de la página en el elemento principal.

### d. Entendiendo DOCTYPE

El **doctype** es una declaración que se encuentra en la parte superior de casi todos los documentos *HTML*.

Cuando un navegador web lee una declaración de tipo *.doc*, el navegador asume que todo lo que se encuentra en el directorio de la página web utiliza el idioma o las normas especificadas en la declaración.



En *HTML 4*, todas las declaraciones `<!DOCTYPE>` requieren una referencia a una *DTD*, que significa definición del tipo de documento. La *DTD* es simplemente un conjunto de reglas que ayudan a un navegador web a convertir el contenido en las páginas que ve en la Web. Hay unos cuantos *DTDs* diferentes que una página web en *HTML4* puede utilizar. Debido a la forma en que se creó *HTML5*, no requiere una referencia a un *DTD*.

En *HTML4*, la declaración doctype especifica el idioma de la página *HTML* y la *DTD*, y parece bastante complejo. Aquí hay un ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.example.com/TR/xhtml11/DTD/
xhtml11.dtd">
```

El nuevo tipo de documento *HTML5*, en comparación, es muy simple:

```
<!doctype html>
```

El doctype *HTML5* no distingue entre mayúsculas y minúsculas, por lo que la palabra clave "doctype" puede ser mayúscula o minúscula.

Este tipo de documento simplificado es parcialmente responsable de por qué las páginas *HTML5* se prestan fácilmente para ser utilizadas en un navegador Web, en un ordenador o en un dispositivo móvil. *HTML5* está diseñado para ser ampliamente compatible con navegadores Web nuevos y antiguos, y con el entorno de dispositivos móviles.

#### e. Explorando el Etiquetado de una Simple Página Web

Un ejemplo de marcado y contenido para una página Web *HTML5* simple es el que aparece cuando:

```
<!doctype html>
<html>
  <head>
    <title>78704 Servicio de mascotas</title>
  </head>
  <body>
    <p>
      Su perro es un amigo de por vida. ¿Por qué no proporcionar la mejor atención
      posible?
    </p>
  </body>
</html>
```

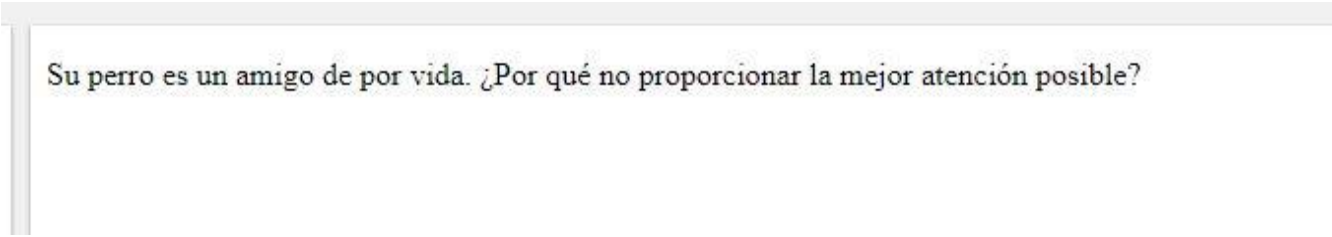


Las líneas en blanco entre partes de la página, como por ejemplo entre la declaración tipo documento y el archivo `<html>`, no aparecen en una página Web. Tampoco los sangrados, como los de los párrafos.

Se observa que los elementos de párrafo están un poco desplazados de las etiquetas `<body>`.

- Las líneas en blanco y sangrías simplemente ayudan a leer el marcado más fácilmente en una herramienta de edición.

En la imagen siguiente se muestra la página Web renderizada para la marcación HTML anterior.



Su perro es un amigo de por vida. ¿Por qué no proporcionar la mejor atención posible?

## 2.2. Elección y configuración de las etiquetas de HTML5 para mostrar textos

**Nota:** *HTML5* utiliza la mayoría de los mismos elementos y atributos especificados en *HTML4*, y tiene algunas etiquetas nuevas, modificó el uso preferido de otras y ya no soporta ciertos elementos. Los nuevos elementos relacionados con el texto incluyen comando, marca, tiempo, medidor y progreso. Algunos de los elementos obsoletos son la fuente base, el centro, la fuente y el tachado.

Todos los elementos tratados en la primera sección de esta lección funcionan bien en *HTML5*, incluso aunque se han utilizado durante años en versiones anteriores de *HTML*. En su mayor parte, *HTML5* reemplaza muy poca sintaxis *HTML*. Esto significa que los desarrolladores pueden seguir utilizando la mayoría de los mismos elementos que siempre han tenido. Algunos elementos tienen la misma etiqueta pero con una funcionalidad ligeramente modificada.

*HTML5* también incluye muchos elementos nuevos que aumentan la funcionalidad de las páginas Web o agilizan el marcado. Estos incluyen elementos multimedia tales como audio, vídeo y elementos que hacen que la estructura de una página Web parezca más intuitiva. Relacionadas con la estructura incluyen elementos para las secciones de la página, encabezados, pies de página, navegación e incluso barras laterales.

Si se crean formularios Web, las nuevas funciones de formularios facilitan mucho la creación y validación.

Esta sección, sin embargo, se centra en el marcado *HTML5* del texto.

### 2.2.1. Elementos de texto de HTML4 con nuevos significados y nuevas funcionalidades

**Nota:** Algunos elementos relacionados con el texto de *HTML4* ahora tienen un significado o funcionalidad ligeramente diferente en *HTML5*. Los elementos incluyen `<b>`, `<i>`, `<strong>`, `<em>`, y `<small>`. El elemento `<b>` debe usarse ahora para resaltar el texto sin transmitir importancia, como por ejemplo para palabras clave o nombres de productos. El elemento `<i>` muestra el texto en otro estado, como por ejemplo, texto hablado. El elemento `<strong>` indica una gran importancia, mientras que el elemento `<em>` indica que se quiere enfatizar. El elemento `<small>` debe ser usado para letra pequeña, como una línea de copyright.

Ahora se verán algunos de los elementos de texto heredados de *HTML 4* que tienen un valor ligeramente diferente en su significado o funcionalidad en *HTML5*:

- **<b>**: Este elemento de uso común siempre ha representado negrita, y se usaba con frecuencia para enfatizar o transmitir importancia. El *W3C* sugiere que ahora se use para indicar texto "compensado estilísticamente" sin transmitir importancia. Se usa **<b>** para palabras clave, nombres de producto y elementos procesables (como los elementos en los que se hace clic o se presiona en una lista de pasos de cómo hacerlo). Por ejemplo:

```
<p>Haz Clic en el botón <b>Validar</b>, y luego en <b>OK.</b> </p>
```

- **<i>**: El elemento cursiva se usa ahora para texto en una "voz o estado de ánimo alternativo". Puede ser texto hablado, pensamientos, o algo similar que no transmita importancia o énfasis. También se puede incluir términos técnicos y palabras extranjeras transliteradas. Por ejemplo:

```
<p><i> Él realmente tiene un corazón bondadoso, </i> por eso, pensó en ella.
```

- **<strong>**: El elemento strong es para la importancia fuerte, donde el contenido es más importante que las palabras cercanas. Por ejemplo:

```
<p>El equipo de Courtney se puso el <strong>mismo</strong> traje para trabajar tres días seguidos.</p>
```

- **<em>**: Este elemento se usa para dar énfasis a lo que se quiere decir. Por ejemplo:

```
<p>Deberías<em>siempre</em> validar tus etiquetas de HTML antes de compartirlo con los otros.</p>
```

- **<small>**: El elemento small, debe ser usado para letras pequeñas o comentarios laterales.

Este es útil para las líneas de copyright o para añadir una línea de origen a una imagen. Por ejemplo:

```
<p><small>Copyright 2018 por la empresa XYZ</small></p>
```

La funcionalidad prevista para algunos de estos elementos en *HTML5* puede ser confusa, como saber cuándo usar el elemento cursiva. El mejor enfoque es esforzarse por lograr consistencia dentro de una página o sitio Web, y observar cómo otros desarrolladores utilizan los mismos elementos.

## 2.2.2. Nuevos elementos de texto en HTML5

Se van a ver elementos nuevos de *HTML5*:



- **<command>**: El elemento de mando se utiliza para definir un botón de mando en el que los usuarios hacen clic para invocar un comando. El elemento de mando tiene muchos atributos que se pueden utilizar, como por ejemplo **type**, **label**, **title**, **icon**, **deactivated**, **marked** y tipo de **radiogroup**. Por ejemplo:

```
<menu label="Tipos de Pinturas">
  <command type="radio" radiogroup="Tipos de Pinturas" label="Barroco">
  <command type="radio" radiogroup="Tipos de Pinturas" label="Rococó">
  <command type="radio" radiogroup="Tipos de Pinturas" label="Renacentista">
</menu>
```

- **<mark>**: El elemento de marca es muy útil para resaltar texto en una página. Se podría usar en una página de resultados de búsqueda, por ejemplo, o para activar un bloque de texto que se desea resaltar para llamar la atención del lector. Por ejemplo:

```
<p>Desde que empecé a hacer ejercicio el otoño pasado, yo he <mark style="background-color:yellow;">perdido 35 kilos</mark>.</p>
```

- **<time>**: El elemento tiempo indica el contenido que es una hora o fecha, que se puede hacer legible por máquina con el atributo **datetime**. El elemento de tiempo define el tiempo en un reloj de 24 horas y una fecha en el calendario gregoriano. Una de las ventajas de hacer que las horas y fechas sean legibles por máquina en la página Web es que ayuda a los motores de búsqueda a producir mejor resultados de búsqueda. Por ejemplo:

```
<time datetime="2013"> significa año 2013
<time datetime="2013-04"> significa abril de 2013
<time datetime="04-15"> significa 15 de abril (cualquier año)
```

Otros dos nuevos elementos son la medición y el progreso. El elemento de medida indica el contenido que es una fracción de un rango conocido, como el uso del disco. El elemento de progreso indica el progreso de una tarea hacia su finalización.

### 2.2.3. Elementos de texto que no se usan en HTML5

La depreciación puede deberse a que un nuevo elemento sustituye a la funcionalidad de un elemento anterior, o bien la preferencia de un nuevo método de formateo sobre un elemento más antiguo. Un ejemplo de lo último es formatear con hojas de estilo en cascada (CSS). Usar CSS para cambiar el aspecto y la sensación de texto, imágenes y otros contenidos web separan el estilo del contenido. El objetivo de W3C ha sido empujar a los desarrolladores hacia el uso de CSS para controlar el formato de páginas Web en lugar de usar formato local durante bastante tiempo, y es claramente el método a utilizar en *HTML5*.



Esto tiene sentido ya que pueden cambiar fácilmente los estilos en CSS que se aplican a través de una página Web o incluso un sitio Web. Insertar estilos individuales incluso en una sola página Web puede implicar un gran gasto de tiempo a la hora de modificarlos.

Los siguientes elementos *HTML* se consideran obsoletos y no se soportan en páginas *HTML5*:

- **<acronym>**: Define acrónimos en *HTML4* que pueden ser pronunciados como si fueran una sola palabra, como GUI (Graphical User Interface) para la interfaz gráfica de usuario. Se utiliza en su lugar la etiqueta **<abbr>**.
- **<applet>**: Define un applet incrustado. Se utiliza en su lugar la etiqueta **<object>**.
- **<basefont>**: Define un color de fuente, un tamaño de fuente o una familia de fuentes predeterminado para todo el texto de un archivo documento. Se utiliza CSS para aplicar todas las fuentes.
- **<big>**: Aumenta el tamaño del texto en relación con el tamaño de la fuente actual. Se utiliza CSS en su lugar.
- **<center>**: Alinea el texto y el contenido en el centro. Se utiliza CSS en su lugar.
- **<dir>**: Define una lista de directorios. Se utiliza en su lugar la etiqueta **<ul>**.
- **<font>**: Especifica el tipo de la fuente, el tamaño de la fuente y el color de la fuente del texto. Se utiliza CSS en su lugar.
- **<frame>**: Define un marco en particular (una ventana) dentro de un conjunto de marcos (véase la siguiente sección) con viñetas.
- **<frameset>**: Define un conjunto de marcos para organizar múltiples marcos (ventanas).
- **<noframes>**: Muestra texto para navegadores que no admiten marcos.
- **<strike>**: Define el texto tachado. Se utiliza la etiqueta **<del>** en su lugar para pequeñas cantidades de texto, o usar CSS para bloques grandes de texto.
- **<tt>**: Define teletipo o texto monoespaciado. Se utiliza la etiqueta **<code>** o CSS en su lugar.

El hecho de que un elemento no esté soportado no significa que no funcione en ciertos navegadores. Muchos usuarios aún utilizan versiones anteriores de los navegadores, y muchos elementos obsoletos se muestran bien en esos navegadores. Sin embargo, una buena práctica es crear páginas asumiendo que los visitantes de la página Web usen la opción navegador actual o casi actual, lo que significa utilizar los últimos elementos *HTML*. Sin embargo, si necesita aplicar mucho formato a cualquier página Web, es mejor utilizar CSS para mejorar la eficiencia.

Los siguientes atributos no se utilizan en *HTML5*, aunque estos atributos no son realmente parte de cualquier especificación *HTML*:

- **Bgcolor**: Aplica un color de fondo específico a cualquier contenido que tenga asociado, que normalmente es una tabla o una página. Se utiliza la propiedad CSS para el color de fondo en su lugar.
- **Bordercolor**: Aplica un color específico a la celda de una tabla. Se utiliza el color del borde CSS en su lugar.
- **Bordercolorlight**: Aplica un color específico a las esquinas superiores e izquierdas de una tabla o celda. Se utiliza en su lugar la propiedad **border-color** CSS.
- **Bordercolordark**: Aplica un color específico a las esquinas inferior y derecha de una tabla o celda. Se utiliza en su lugar la propiedad **border-color** CSS.

Al igual que con los elementos obsoletos, se pueden utilizar estos atributos si se sabe que los visitantes de la página Web utilizan navegadores más antiguos. Hay que tener en cuenta que los intentos de





validar la página Web resultarán en caso de error, que se puede ignorar si se está seguro de que los navegadores de los visitantes soportan los atributos.

## 2.3. Elección y configuración de las etiquetas de HTML5 para gráficos

Se pueden mostrar diferentes tipos de imágenes en una página Web, la mayoría de las cuales se dividen en dos categorías principales:

- Raster (o bitmap) : Una imagen rasterizada se compone de píxeles, mientras que una imagen vectorial se compone de líneas y curvas basadas en expresiones matemáticas. Una fotografía es un tipo de raster y está más a menudo en formato *JPG*. Otros formatos de archivos raster que funcionan bien en las páginas Web son *PNG*, *GIF* y *BMP*.
- Vector : Una imagen vectorial es una ilustración, como un dibujo de línea. Los desarrolladores a menudo convierten formatos de archivos vectoriales de programas como *Adobe Illustrator* o *CorelDRAW*, que no son compatibles con los navegadores Web, en *PNG* o *GIF* para su visualización en la Web. Una diferencia importante entre los dos tipos de archivos es que las imágenes raster pierden calidad (se pixelan) cuando se amplían, pero las imágenes vectoriales mantienen la calidad incluso cuando se amplían.

La forma principal de añadir imágenes a un documento *HTML* es con el elemento **img**. Como la anchura, la etiqueta **img** no hace nada por sí misma y requiere atributos y valores que especifican la imagen que debe mostrar el navegador web.

Por ejemplo, para insertar una imagen llamada *bolaRoja.jpg* que está en una subcarpeta llamada imágenes, escriba este elemento:

```

```

La imagen se mostrará mientras la subcarpeta de imágenes esté accesible. Ambos atributos **src** y **alt** deben ser totalmente válidos. El valor del atributo **alt** (abreviatura de texto alternativo) aparece cuando el usuario pasa el puntero del ratón por encima de la imagen; en este caso, en el icono se mostraría la frase "Bola Roja". El *W3C* requiere el atributo **alt** para facilitar la lectura a personas con discapacidades. Las personas con visión limitada pueden usar un lector de pantalla, que lee en voz alta el texto alternativo de cada imagen. Los motores de búsqueda también utilizan el atributo **alt** para identificar tipos de imágenes y lo que hay en ellas, ya que los motores de búsqueda no pueden "ver" píxeles en las imágenes.

Como otro ejemplo, para insertar una imagen llamada *logoAzul.png* a la que se puede acceder desde otro archivo al sitio web, se escribirá el siguiente elemento:

```

```

A continuación, se van a ver los diferentes atributos que usa el elemento **img**:



Atributo	Valor	Descripción
<b>src</b>	URL	Especifica la ruta, URL o localización donde se encuentra la imagen
<b>alt</b>	Text	Especifica texto alternativo para la imagen, que se muestra cuando el usuario pasa el puntero del ratón u otro dispositivo señalador sobre la imagen
<b>height</b>	pixels	Especifica la altura de la imagen
<b>width</b>	pixels	Especifica la anchura de la imagen
<b>ismap</b>	ismap	Especifica que es una imagen sobre la que puedes hacer Click.
<b>usemap</b>	#mapname	Especifica sobre qué parte de la imagen puedes hacer Click

### 2.3.1. Usando el figure y elementos de tipo *figcaption*

El elemento de la **figura** especifica el tipo de figura que está añadiendo, como una imagen, un diagrama o una foto, y así sucesivamente. Este elemento proporciona una gran ventaja: la posibilidad de añadir fácilmente varias imágenes juntas al mismo tiempo. El **elemento figcaption** es opcional. Este añade un título a una imagen en una página Web, y puede mostrar el título antes o después de la imagen.

La siguiente marca de revisión utiliza el elemento de la figura y especifica la anchura y la altura de la imagen, y añade una leyenda. El resultado y el ejemplo de usarlos, se puede ver a continuación:

```
<figure>
  
  <figcaption>Perros felices son buenos perros</figcaption>
</figure>
```

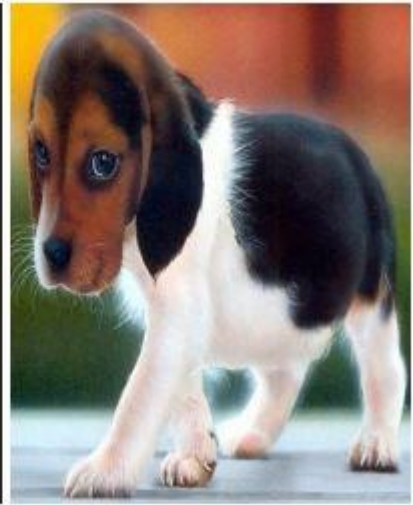
En el navegador se mostrará:



Y si se quiere mostrar múltiples imágenes se puede usar el siguiente código, con las imágenes que se tengan guardadas:

```
<!doctype html>
<html>
  <figure>
    
    
    
    <figcaption>Perros felices son perros buenos</figcaption>
  </figure>
</body>
</html>
```

Y se mostrará como:



Perros felices son perros buenos

### 2.3.2. Creando gráficos con canvas

**Nota:** El elemento canvas, es nuevo en *HTML5* y crea un contenedor para gráficos, y utiliza *JavaScript* para dibujar los gráficos dinámicamente.

Con canvas, la página Web se convierte en un bloc de dibujo y se utilizan comandos *JavaScript* para dibujar formas basadas en píxeles en un lienzo que incluye color, degradados y rellenos de patrones. Canvas también permite renderizar texto con varios adornos, y animar objetos haciendo que se muevan, cambien de escala, y así sucesivamente.

#### a. Canvas básico

Para utilizar el lienzo, se define primero un lienzo en *HTML*. La sintaxis básica del elemento de lienzo es la siguiente:

```
<canvas id="smlRectangle" height="100" width="200"></canvas>
```

Este elemento crea un bloc de dibujo. El elemento de canvas requiere el atributo id para referenciarlo en *JavaScript* y añadirlo al modelo de objeto de documento (*DOM*). También se deben especificar las dimensiones del canvas (altura y anchura) que están en píxeles. Javascript funciona con la interfaz de programación de aplicaciones (*API*) bidimensional (2D) para dibujar en el canvas.

#### b. Creando contorno de una forma

Para crear un contorno de un rectángulo sin color de relleno, utilice el método `context.strokeRect`. Utiliza los mismos valores que `context.fillRect`. Para modificar el color del contorno (el color del trazo), se usa `context.strokeStyle`. Por ejemplo, para crear una imagen de 200 x 100 píxeles con contornos rectangulares en rojo, se utilizan estos métodos de *JavaScript*:

```
context.strokeStyle = "red";  
context.strokeRect(10,20,200,100);
```

### c. Proporcionando una forma alternativa para imágenes y textos para viejos navegadores

Algunos navegadores antiguos no pueden renderizar dibujos en canvas o animaciones. Por lo tanto, se debe añadir una imagen, texto u otro contenido *HTML* dentro del elemento de canvas que se mostrará si el dibujo no puede. El contenido de la "copia de seguridad", también conocido como contenido alternativo, no se mostrará si el canvas está soportado. Este ejemplo muestra una imagen (*Rectangulo.jpg*) similar a la que crearía un canvas rectangular con relleno:

```
<canvas id="Rectangulo" height="100" width="200">  
    
</canvas>
```

Para mostrar texto en lugar de una imagen, se debe insertar texto en lugar de la etiqueta `<img>`.

### 2.3.3. Creando gráficos con SVG

El objetivo principal de *SVG* (Scalable Vector Graphics o Vectores Gráficos Escalables), como su nombre indica, es crear formas gráficas vectoriales escalables, pero también se puede crear imágenes y texto. Al igual que el canvas, se pueden aplicar colores sólidos, y rellenos de patrones a objetos *SVG*, y copiar y clonar objetos. También se puede utilizar *SVG* en cualquier lugar en el que se desee insertar un *PNG*, *JPG* o *GIF*. Con *SVG*, se proporciona el dibujo en lugar de un archivo de imagen.

Una de las principales ventajas de *SVG* es su flexibilidad. Su gráfico vectorial cambia de tamaño para adaptarse a las pantalla en la que se muestra, tanto si la pantalla está en un monitor de ordenador de 32 pulgadas como en un teléfono inteligente. Porque sólo el *XML* que describe el gráfico *SVG* se transmite, incluso las imágenes grandes no requieren mucho ancho de banda. Esto hace que *SVG* sea útil como fondo de página Web sin tener que usar la propiedad *repeat*. Además, *SVG* puede ser indexado por los motores de búsqueda porque es creado por *XML*.

Se pueden incluir atributos como el color, la rotación, el color y el tamaño de trazo, etc., para cada uno de los objetos *SVG*. Las siguientes marcas de revisión pueden incluirse en un archivo *HTML* para crear una bola púrpura:

```
<svg id="svgpurpball" height="200" xmlns="http://www.w3.org/2000/svg">  
  <circle id="purpball" cx="40" cy="40" r="40" fill="purple" />  
</svg>
```



Los atributos **cx**, **cy**, y **r** ayudan a definir el círculo al definir los puntos **x** e **y** del centro y radio. **SVG** tiene un conjunto de atributos que le ayudan a crear todo tipo de formas.

Se puede encontrar más información en <http://www.w3.org/TR/SVG/attindex.html>.

### 2.3.4. Cuando se usa canvas en lugar de SVG

Las siguientes son algunas consideraciones que le ayudarán a tomar la decisión correcta:

- Si el dibujo es relativamente pequeño, se utiliza canvas.
- Si el dibujo requiere un gran número de objetos, se utiliza canvas. **SVG** comienza a degradarse a medida que añade continuamente objetos al *DOM*.
- Por lo general, se utiliza canvas para pantallas pequeñas, como las de los dispositivos móviles. Cuando el tamaño de la pantalla aumenta y se necesitan más píxeles, el canvas comienza a pixelarse, así que debe utilizarse **SVG**.
- Si se deben crear documentos vectoriales muy detallados que deben escalarse bien, se utilizará **SVG**.
- Si se están mostrando datos en tiempo real, como mapas, superposiciones de mapas, datos meteorológicos, y así sucesivamente, se usará lona.

## 2.4. Elección y configuración de etiquetas de HTML5 para Play Media

Los elementos de audio y vídeo son dos de los principales cambios en **HTML5**, lo que le permite proporcionar contenido multimedia desde un navegador web sin necesidad de plug-ins, como los de *Microsoft Windows Media Player*, *Microsoft Silverlight*, *Adobe Flash* o *Apple QuickTime*.

Esto significa que los usuarios pueden simplemente abrir un archivo compatible con **HTML5** para escuchar música o audiolibros, disfrutar de efectos de sonido y ver videoclips o películas. La especificación **HTML5** incluye las etiquetas **<video>** y **<audio>** para incorporar contenido multimedia. Las siguientes secciones cubren cada una de ellas en detalle.

### 2.4.1. Entendiendo y usando etiquetas de vídeo

**Nota:** Utilizar el elemento de *video* junto con el atributo *src* para designar un archivo de vídeo como en un documento **HTML**. Incluir los atributos de altura y anchura permite controlar el tamaño de la ventana en la que se muestra el vídeo.

El **elemento de vídeo** permite incorporar vídeos en documentos **HTML** con una mínima cantidad de código. La estructura para incrustar vídeo es simple. El siguiente es un ejemplo para añadir un archivo *MP4* a una página Web:

```
<video src="intro.mp4" width="400" height="300"></video>
```

El atributo **src** apunta al nombre del archivo de vídeo (en este caso, *video.mp4*) que será reproducido. Los atributos de altura y anchura especifican el tamaño de la ventana en la que el vídeo se mostrará.

Hay otros atributos disponibles que se pueden añadir para controlar el vídeo:

- **poster:** Muestra un archivo de imagen estático antes de que se cargue el vídeo.



- **autoplay**: Comienza a reproducir el vídeo automáticamente al cargar la página.
- **controls**: Muestra un conjunto de controles para reproducir, pausar y detener el vídeo, y controlar el volumen.
- **loop**: Repite el vídeo.

Usando todos los controles listados arriba, el marcado se vería similar a éste:

```
<video src="/videos/intro.mp4" width="400" height="300" poster="78704-splash.jpg"
autoplay="autoplay" controls="controls" loop="loop"></video>
```

Otros formatos de vídeo web populares también incluyen *H.264*, *OGG* y *WebM*, aunque WebM se utiliza menos del 10 por ciento del tiempo.

Junto con el formato de vídeo, también se debe especificar el códec, que es una tecnología utilizada para comprimir datos. La **compresión** reduce la cantidad de espacio necesaria para almacenar un archivo, y esto reduce el ancho de banda necesario para transmitir el archivo. La **compresión de vídeo** reduce el tamaño de imágenes de vídeo, conservando al mismo tiempo la máxima calidad de vídeo con la mínima tasa de bits. Todo esto permite un mejor rendimiento.

En pocas palabras, los principales formatos de vídeo junto con los codecs (para los dos últimos) son:

- *MP4* o *H.264*
- *OGG* + *Theora with Vorbis audio*
- *WebM* + *VP8*

Una buena práctica es utilizar el atributo `type` para especificar el formato de vídeo. También se debe utilizar el atributo `codecs` para especificar los codecs, si procede. Por ejemplo:

```
<video width="400" height="300" poster="78704-splash.jpg" autoplay="autoplay"
controls="controls" loop="loop"><source src="intro.mp4" type="video/mp4" /></video>
```

La etiqueta `<source>` se utiliza como contenido del elemento de vídeo de forma que el atributo `type` pueda usarse para que la opción de formato múltiple esté disponible.

No todos los formatos de vídeo son compatibles con todos los navegadores, aunque *MP4/H.264* es el formato más utilizado tanto por los navegadores Web como por los dispositivos móviles. Si es necesaria más información acerca de este tema, se puede mirar la página web de vídeo de *HTML5* en [http://www.w3schools.com/html5/html5\\_video.asp](http://www.w3schools.com/html5/html5_video.asp), donde se muestra una tabla con los formatos de vídeo que se usa para cada navegador, este sitio se actualiza regularmente. Para ayudar a hacer el vídeo visualizable por la mayoría de los navegadores y dispositivos, se puede utilizar el atributo `source` para incluir múltiples formatos en su marcado. Este ejemplo muestra el mismo vídeo disponible en dos formatos, y el formato OGG especifica codecs:

```
<video width="400" height="300" poster="image.png" autoplay="autoplay" controls="controls"
loop="loop">
  <source src="video.mp4" type="video/mp4">
```



```
<source src="video.ogg" type='video/ogg; codecs="theora, vorbis"'>
</video>
```

### 2.4.2. Entendiendo y usando etiquetas de audio

**Nota:** El elemento de audio *HTML5* funciona de forma muy parecida al elemento de vídeo, pero sólo para el sonido. Para usar el elemento de audio, se incluye la etiqueta `<audio>` y una ruta al archivo o una etiqueta localizador de recursos uniforme (URL) que apunta al archivo de audio.

El **elemento de audio** permite incorporar audio, como música y otros sonidos, en documentos HTML. Se puede incluir los mismos atributos relacionados con el control que el vídeo **autoplay**, **controls** y **loop**. El siguiente ejemplo muestra sólo el elemento atributo de control incluido:

```
<audio src="sample.mp3" controls="controls"></audio>
```

Los tres tipos principales de archivos de audio compatibles con los navegadores más populares son *OGG*, *MP3* y *WAV*.

Sin embargo, no todos los navegadores soportan todos los formatos de archivos de audio, al menos hoy en día. Para asegurarse de que el audio se reproduce en la mayoría de los navegadores y dispositivos, se utiliza la fuente para incluir múltiples formatos en el marcado. Este ejemplo muestra el mismo audio disponible en dos formatos:

```
<audio controls="controls">
  <source src="sample.ogg" type="audio/ogg" />
  <source src="sample.mp3" type="audio/mp3" />
</audio>
```

Se puede encontrar una gran cantidad de archivos de audio gratuitos, que también están libres de derechos de autor, en <http://flashkit.com>. Este es un buen recurso para los estudiantes y para los desarrolladores que pueden necesitar un efecto de sonido para un proyecto. Otra fuente es el sitio Web de los Sherpa de Dominio Público en <http://www.publicdomainsherpa.com/public-domain-recordings.html>. También se pueden hacer grabaciones propias utilizando un ordenador y software de grabación. *Windows 7* incluye el *Sound Recorder*, que le permite guardar archivos de audio en formato *WAV*.





## 3. Construyendo la Interfaz de Usuario con HTML5: Organización, Elementos de Entrada y Validación

---

### 3.1. Eligiendo y Configurando Etiquetas de HTML5 para Organizar Contenido y Formularios

**Nota:** *HTML5* introduce varios elementos nuevos para organizar el contenido y los formularios. Representan la nueva forma de marcado que es una parte importante de *HTML5*.

El marcado *HTML5* introduce muchas etiquetas de marcado nuevas para organizar la estructura de HTML lo que facilita la creación y modificación de documentos. Las nuevas etiquetas tienen nombres más intuitivos que construcciones similares en especificaciones *HTML* anteriores; las etiquetas se denominan más apropiada para la parte de la página a la que se aplican, como `<header>`, `<section>` y `<footer>`.

*HTML5* también ha agilizado la creación de tablas, trasladando muchos de los atributos de las tablas que afectan a ancho, relleno de celdas y alineación vertical y horizontal al archivo CSS.

#### 3.1.1. Entendiendo la sintaxis de HTML

Una de las nuevas características más útiles de *HTML5* es el uso de marcado semántico, que da mejor significado, o definición, a varias etiquetas para que tengan más sentido para los programas, los humanos, y navegadores web. Como se mencionó en la Lección 2, no todas las etiquetas *HTML* han sido reemplazadas o actualizadas para *HTML5*, pero algunas etiquetas nuevas que han sido introducidas en *HTML5* hacen que el trabajo de crear las páginas web sea mucho más fácil.

En *HTML 4.01* y especificaciones previas, un desarrollador que crea la estructura de un *HTML* usa la etiqueta `<div>` frecuentemente en todo el documento. La etiqueta `<div>` a menudo incluye una clase o ID, que también puede incluir estilos CSS como el `background-color`, `height` y el `width`. Un ejemplo simple de una etiqueta `<div>` es:

```
<div id="header" > Esto es un encabezado </div>
```

*HTML5* introduce varios elementos nuevos para organizar el contenido y los formularios. Representan la nueva marcación semántica que es una parte importante de *HTML5*.

**Nota:** `class` e `id` son **atributos globales**, lo que significa que pueden ser usados con cualquier elemento *HTML*. Se puede ver la lista completa de atributos *HTML* globales en <http://dev.w3.org/html5/markup/global-attributes.html>.

El elemento `div` por sí solo no tiene mucho significado sin el atributo `id` o `class`. Incluso se puede asignar al ID un valor de su elección, como `"header"`, `"header_inner"`, `"slogan"`, `"content"`, `"style"`, y muchos más. Un ejemplo de un documento *HTML 4.01* se muestra de la siguiente manera:

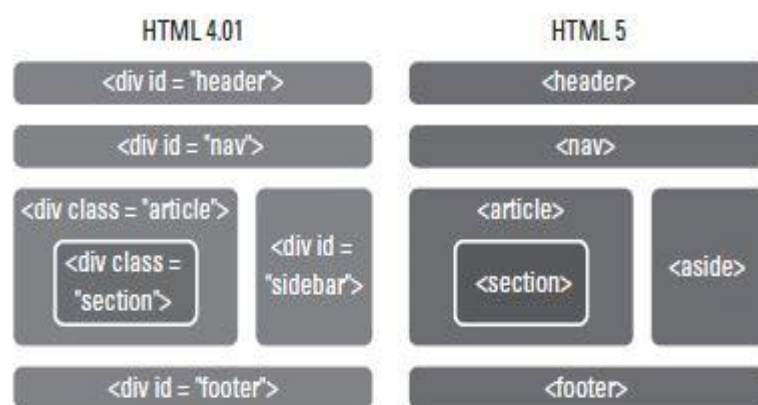


```

<div id="header">
  <div id="header_inner">
    
    <div id="slogan">Perros Felices son Buenos Perros</div>
  </div>
</div>

```

*HTML5* utiliza etiquetas más simples para reemplazar muchas de las etiquetas **div**, algunas de las cuales se muestran en la figura siguiente:



Nótese cómo el marcado semántico de *HTML5* da un significado más específico a partes de un documento *HTML*, haciendo que la estructura sea más fácil de entender.

### 3.1.2. Usando Etiquetas para Añadir Estructura a un Documento de HTML

En la siguiente tabla se pueden ver algunos de los nuevos elementos de *HTML5* para organizar los documentos:

Etiquetas	Descripción
<b>&lt;address&gt;</b>	Define un área para la información de contacto de una página o sección
<b>&lt;article&gt;</b>	Define un artículo, como un artículo de revista o periódico, una entrada de blog, o contenido similar
<b>&lt;aside&gt;</b>	Define el contenido que está separado pero relacionado con el contenido de la página; similar a una barra lateral en capítulos de libros y artículos de revistas
<b>&lt;details&gt;</b>	Contiene detalles adicionales pertinentes al texto que lo rodea; crea un archivo interactivo que un usuario puede mostrar u ocultar

<b>&lt;footer&gt;</b>	Define un pie de página para un documento o sección; puede incluir el nombre del autor del documento, información de contacto, información de copyright y/o enlaces a términos de uso
<b>&lt;header&gt;</b>	Define un encabezado para un documento o sección; puede contener contenido introductorio o enlaces de navegación
<b>&lt;hgroup&gt;</b>	Encabezados y subencabezados de grupos (usando las etiquetas <b>&lt;h1&gt;</b> a <b>&lt;h6&gt;</b> ) para headers multiniveles
<b>&lt;nav&gt;</b>	Define un bloque de enlaces de navegación
<b>&lt;section&gt;</b>	Define una sección en un documento, como capítulos, partes de una tesis, o las partes de una página web cuyo contenido es distinto de los demás
<b>&lt;summary&gt;</b>	Define un encabezado visible para un elemento de detalle; el usuario puede hacer clic para visualizar o ocultar información
<b>&lt;wbr&gt;</b>	Define un posible salto de línea; cuando una palabra o línea es muy larga el navegador las podría romper en el lugar equivocado; se puede usar el comando <b>&lt;wbr&gt;</b> para romper la palabra o línea apropiadamente

#### a. Encabezados y comentarios al pie de página

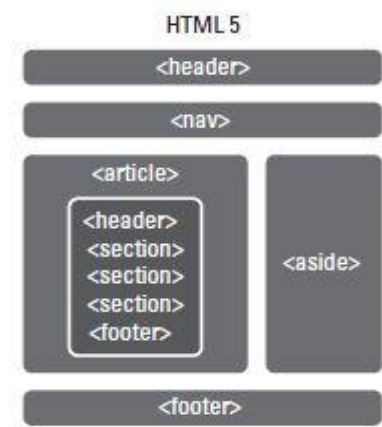
El **elemento header** define una cabecera para un documento, sección o artículo. El elemento **footer** define un pie de página para un documento o sección, y normalmente contiene información sobre el documento o la sección, como el nombre del autor, los datos de copyright, los enlaces a sitios web relacionados, etc. documentos, etcétera. El elemento **footer** no aparece automáticamente en la parte inferior (o pie) del documento: es necesario utilizar CSS para indicar al navegador dónde se debe mostrar el pie de página. Los pies de página que aparecen en la parte inferior de cada página web o documento se conocen como "sticky footers".

Un ejemplo de un artículo con una etiqueta de encabezado y una etiqueta de pie de página es el siguiente:

```
<article>
  <header>
    <h1>Aprendiendo HTML5</h1>
    <h2>Los Nuevos Elementos</h2>
  </header>
  <p>
    Nuevas etiquetas de HTML5 para hacer una página Web y desarrollar una aplicación
    Más fácilmente.
  </p>
  <footer>
    <p>
      Publicado: <time datetime="2012-09-03" Septiembre 3, 2018</time>
    </p>
  </footer>
</article>
```



Al igual que el elemento **div**, se pueden utilizar los elementos de cabecera y pie de página varias veces en un *HTML*, como se muestra a continuación:



#### b. Elemento section

El **elemento section** define una sección en un documento, como un capítulo o como partes de una tesis, o partes de una página web cuyo contenido es distinto de los demás. El W3C especifica los usos para que el elemento de sección lo diferencie de otros elementos relacionados con la estructura, principalmente contiene al menos un encabezado y define algo que aparecería en el esbozo del documento. Por ejemplo, debería utilizar el elemento de **section** para dividir diferentes partes de un sitio Web de una página o para crear un portafolio de imágenes. El siguiente es un ejemplo de una sección simple:

```

<section>
  <h1>Eight Count</h1>
  <p> Los instructores de baile hip-hop a menudo enseñan movimientos
    que tienen ocho tiempos por movimiento.</p>
</section>

```

A continuación, se enumeran las situaciones en las que se debe evitar el uso del elemento **section**, y se proporciona una mejor forma:

Situación	Uso
Contenido independiente del resto del artículo de contenido en la página Web o documento	<b>article</b>
Plan para organizar un artículo de contenido en bloques	<b>article</b>
Crear barra lateral	<b>aside</b>
Envolver y posicionar varias secciones que no están relacionadas entre sí	<b>div</b>
Agregar una sombra o un borde alrededor de un elemento	<b>div</b>

Saber cuándo usar la etiqueta `<section>` frente a un elemento diferente puede ser difícil a veces.

Cuando se trabaja en un documento *HTML* y no se está seguro de qué elemento usar, se pueden buscar las especificaciones de *HTML5* del *W3C* o investigar la Web para ver cómo han manejado otros desarrolladores una situación similar. Al definir una cabecera de sección, que puede contener encabezados de **h1** a **h6**, se puede utilizar la función **hgroup** en las cabeceras de los grupos. El elemento **hgroup** afecta a la organización, pero no la presentación.

Se puede considerar la posibilidad de usar **hgroup** cuando se tiene un título y un subtítulo, uno inmediatamente después del otro, como se puede ver a continuación:

```
<section>
  <hgroup>
    <h1>Los movimientos de los bailarines de Hip-Hop</h1>
    <h3>Método de contar ocho</h3>
  </hgroup>
  <article>
    <p> Los instructores de baile hip-hop a menudo enseñan movimientos que tienen ocho
    tiempos por movimiento.
    </p>
  </article>
</section>
```

El código anterior se mostrará en una página Web como se muestra en la figura de a continuación.

## Los movimientos de los bailarines de Hip-Hop

### Método de contar ocho

Los instructores de baile hip-hop a menudo enseñan movimientos que tienen ocho tiempos por movimiento.

#### c. Elemento nav

El **elemento nav** define un bloque de enlaces de navegación. El **nav** es útil para crear un conjunto de enlaces de navegación como un índice de contenidos, un pequeño menú en un pie de página o en los enlaces Previous-Home-Next.

El *W3C* menciona que no se tienen que usar etiquetas `<nav>` para todos los enlaces de navegación, sólo los principales bloques de enlaces. Como las etiquetas `<nav>` son interpretadas por el software de lectura de pantalla, visualmente el software puede determinar si debe hacer que los enlaces de navegación estén disponibles para el usuario inmediatamente o no, dependiendo de su importancia.

El siguiente ejemplo muestra la etiqueta `<nav>` en uso:

```
<nav>
  <a href="/ Natación /">Natación</a>
  <a href="/ Baile /">Baile</a>
  <a href="/ Fútbol /">Fútbol</a>
```



```
<a href="/ Baloncesto /">Baloncesto</a>
</nav>
```

Los enlaces se mostrarán en una página Web como se muestra en la figura siguiente:



#### d. Elemento article

El **elemento article**, define una parte de un documento *HTML* que consiste en una "composición de contenido propio" que es independiente del resto del contenido del documento. El contenido desencadenado por **<article>** puede ser distribuido en sindicación, así que se puede pensar en él como contenido que tiene sentido por sí mismo. (La sindicación web es el proceso de crear contenido a partir de un sitio Web disponible para muchos sitios Web).

Ejemplos de elementos que tienen cabida en el etiquetado con **<article>** son artículos de revista, un blog o contenido para un RSS feed.

#### e. Elemento aside

El **elemento aside** se utiliza para resaltar el contenido que está relacionado con el tema actual, pero interrumpiría el flujo del documento si se deja en línea. Esencialmente, el elemento **aside** se utiliza para información que se presta a las barras laterales y notas. Este contenido puede dar una visión más detallada a un tema, ofrecer enlaces de lectura relacionados o mostrar definiciones para palabras clave en el párrafo. El elemento **aside** no cambia la posición del contenido o cómo se muestra el contenido; simplemente hace saber al navegador y a los motores de búsqueda que es contenido relacionado.

```
<article>
  <header>
    <h1>Aprendiendo HTML5</h1>
    <h2>Los Nuevos Elementos</h2>
  </header>
  <p>Nuevas etiquetas de HTML5 para hacer páginas Web y
  Desarrollar aplicaciones más fácilmente Una de las nuevas características más útiles
  de HTML5 es el uso de marcado semántico.</p>
  <aside>
    <h4><b> marcado semántico </b></h4>
    <p> da mejor significado, o definición, a las etiquetas para que tengan
    más sentido para las personas,
    y navegadores web.
    </p>
  </aside>
  <p>No todas las etiquetas de HTML tienen que ser remplazadas o actualizadas
```

```

en HTML5, pero algunas nuevas etiquetas de HTML5,
se han creado para hacer más fácil lo que es la creación de páginas Web.</p>
<footer>
  <p>Publicado: <time datetime="2018-12-
    03">Septiembre 3, 2018</time></p>
</footer>
</article>

```

El navegador nos mostrará lo que hay en la siguiente imagen:

## Aprendiendo HTML5

### Los Nuevos Elementos

Nuevas etiquetas de HTML5 para hacer páginas Web y Desarrollar aplicaciones más fácilmente Una de las nuevas características más útiles de HTML5 es el uso de marcado semántico.

#### marcado semántico

da mejor significado, o definición, a las etiquetas para que tengan más sentido para las personas, y navegadores web.

No todas las etiquetas de HTML tienen que ser remplazadas o actualizadas en HTML5, pero algunas nuevas etiquetas de HTML5, se han creado para hacer más fácil lo que es la creación de páginas Web.

Publicado: Septiembre 3, 2018

Podemos ver que el contenido aparece separado, aunque podría no ser lo suficiente. Si se quiere hacer que esté más diferenciado, lo que se debe hacer es poner, antes y después de las etiquetas **aside**, la etiqueta `<hr/>` y quedaría como se puede ver en la siguiente imagen:

## Aprendiendo HTML5

### Los Nuevos Elementos

Nuevas etiquetas de HTML5 para hacer páginas Web y Desarrollar aplicaciones más fácilmente Una de las nuevas características más útiles de HTML5 es el uso de marcado semántico.

#### marcado semántico

da mejor significado, o definición, a las etiquetas para que tengan más sentido para las personas, y navegadores web.

No todas las etiquetas de HTML tienen que ser remplazadas o actualizadas en HTML5, pero algunas nuevas etiquetas de HTML5, se han creado para hacer más fácil lo que es la creación de páginas Web.

Publicado: Septiembre 3, 2018

También se puede utilizar CSS para ajustar los márgenes del contenido lateral de modo que quede sangría a izquierda y derecha.



### 3.1.3. Usando Etiquetas para Crear Tablas y Listas

Las tablas y listas estructuran la información específica de los documentos *HTML*. Una tabla contiene filas y columnas, y muestra los datos en una cuadrícula. En *HTML*, se pueden crear listas ordenadas y desordenadas. Cada artículo de una lista ordenada está marcado con un número o una letra, mientras que una lista no ordenada es una lista con viñetas.

Esta sección se centra en cómo crear tablas y listas utilizando elementos *HTML*. *HTML5* presenta algunos elementos nuevos, tanto para las tablas como para las listas.

#### a. Creando Tablas

Una tabla *HTML* contiene filas y columnas y se utiliza para organizar y mostrar información en formato de cuadrícula. Algunos desarrolladores utilizan tablas con fines de diseño, como por ejemplo para posicionar o alinear con imágenes, pero no es el mejor uso para las tablas.

Con respecto al marcado, cada tabla *HTML* comienza con la etiqueta `<table>`. Las filas están marcadas con la etiqueta `<tr>`, las cabeceras de columna usan la etiqueta `<th>`, y las celdas son definidas por la etiqueta `<td>`. Se han añadido comentarios informativos que no aparecen cuando el documento se visualiza en un navegador, se muestra en la a continuación:

```
<table border="1">
  <tr> <!--fila 1-->
    <th>Trimestre</th> <!--Primera fila en la primer columna-->
    <th>Total Ventas</th> <!--primera fila, segunda columna-->
  </tr>
  <tr> <!--segunda fila-->
    <td>T1</td>
    <td>$4,349</td>
  </tr>
  <tr> <!--tercera fila-->
    <td>T2</td>
    <td>$2,984</td>
  </tr>
  <tr> <!--cuarta fila-->
    <td>T3</td>
    <td>$3,570</td>
  </tr>
  <tr> <!--quinta fila-->
    <td>T4</td>
    <td>$7,215</td>
  </tr>
</table>
```

Y se verá:





Trimestre	Total Ventas
T1	\$4,349
T2	\$2,984
T3	\$3,570
T4	\$7,215

Basándose en una tabla sencilla, se puede utilizar la etiqueta `<caption>` para añadir un subtítulo arriba o abajo la tabla. Para aplicar estilos en línea utilizando *HTML* en lugar de *CSS*, se puede utilizar la etiqueta `<col>` para aplicarlo a una columna entera.

El `<colgroup>` agrupa las columnas dentro de una tabla de modo que se pueda aplicar formato al grupo en lugar de sólo una etiqueta columna.

Cuando se cree una tabla larga que requiera desplazarse dentro de un navegador, se deben usar las etiquetas `<thead>`, `<tfoot>`, y `<tbody>` etiquetas. El contenido dentro del encabezado y pie de tabla permanecerá en la página mientras que el contenido marcado por `<tbody>` se desplazará entre ellos.

La etiqueta `<thead>` crea encabezados de columna (en negrita por defecto), y la etiqueta `<tfoot>` se utiliza para visualizar la última línea, como una línea de totales. La etiqueta `<tbody>` define todo el contenido entre el encabezado y el pie de página.

A continuación, se muestra un ejemplo del marcado para una tabla con tres columnas y cinco filas, la primera fila es el encabezado de la columna y la última fila el pie de la tabla. El marcado también incluye una leyenda sobre la tabla.

```
<table>
  <caption>Ventas por el Empleado ID 2387</caption>
  <colgroup
    span="2"
    style="background-color:#EEE8AA;">

  </colgroup>
  <colgroup
    style="background-color:#00FA9A;">
  </colgroup>
  <thead>
    <tr>
      <th scope="col">Trimestres</th>
      <th scope="col">Total Ventas</th>
      <th scope="col">Objetivo Conseguido?</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <th scope="col">Total</th>
      <th scope="col">$18,118</th>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <td>T1</td>
```

```

        <td>$4,349</td>
        <td>Sí</td>
    </tr>
    <tr>
        <td>T2</td>
        <td>$2,984</td>
        <td>No</td>
    </tr>
    <tr>
        <td>T3</td>
        <td>$3,570</td>
        <td>Sí</td>
    </tr>
    <tr>
        <td>T4</td>
        <td>$7,215</td>
        <td>Sí</td>
    </tr>
</tbody>

```

Y da como resultado:

Trimestres	Total Ventas	Objetivo Conseguido?
T1	\$4,349	Sí
T2	\$2,984	No
T3	\$3,570	Sí
T4	\$7,215	Sí
<b>Total</b>	<b>\$18,118</b>	

Se observa en el ejemplo anterior el uso del color de fondo para las columnas agrupadas. Éste es un ejemplo de formato en línea. El atributo **style** usa una o más propiedades CSS, separados por punto y coma. Para el color *HTML*, puede utilizar el nombre del color o el código hexadecimal. El código hexadecimal **#EEE8AA** produce el color dorado pálido. El código hexadecimal **#00FA9A** produce el color verde primavera. Un conjunto de colores estándar *HTML* está disponible en [http://www.w3schools.com/html/html\\_colonames.asp](http://www.w3schools.com/html/html_colonames.asp).

También puede centrar el contenido en una celda, columna o grupo de columnas usando **style="text-align:center"**. Para encadenar múltiples propiedades y valores en el mismo atributo de estilo, utilice una sintaxis similar a **style="color:blue;text-align:center"**. CSS y sus muchas propiedades se tratan en las Lecciones posteriores en este libro.

En la tabla siguiente podemos ver los elementos que se usan para fabricar tablas en *HTML5*:

Elemento	Descripción
<b>col</b>	Define una columna de una tabla
<b>colgroup</b>	Define un grupo de columnas en una tabla
<b>caption</b>	Marca el texto como un título de una tabla
<b>table</b>	Define una tabla
<b>tbody</b>	Define un grupo de filas en una tabla para formatear y desplazarse
<b>td</b>	Define una celda de una tabla
<b>tfoot</b>	Define un grupo de filas de pie de página en una tabla para formatear y desplazarse
<b>th</b>	Define una celda de encabezado de una tabla
<b>thead</b>	Define un grupo de filas de encabezado en una tabla para fines de formato y desplazamiento
<b>tr</b>	Define una línea de una tabla

## b. Creando Listas

La creación de listas en *HTML5* es sencilla. *HTML5* introduce algunos atributos nuevos.

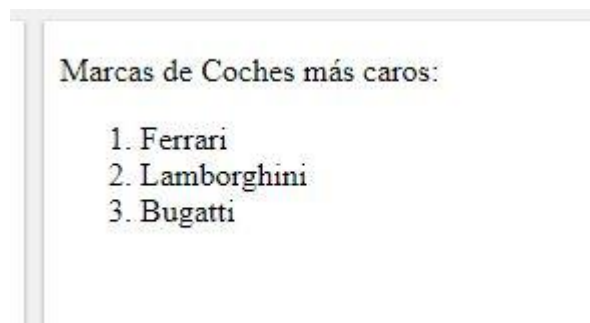
Hay dos tipos principales de listas *HTML*:

- **Listas ordenadas:** Ordena las entradas de la lista utilizando números, por defecto. Se utiliza la etiqueta `<ol>`. Con una lista ordenada se puede utilizar los siguientes atributos:
  - a) **reversed:** Invierte el orden de la lista. Este atributo no es soportado en la mayoría de los navegadores en el momento de escribir este documento.
  - b) **start number:** Especifica el valor inicial de la lista ordenada.
  - c) **type:** Especifica el tipo de marcador que se utilizará al principio de cada elemento de la lista; el valor "1" es el valor por defecto y muestra números decimales, el valor "A" usa letras mayúsculas, el valor "a" utiliza letras minúsculas, el valor "I" utiliza números romanos en mayúsculas, y el valor "i" utiliza números romanos en minúsculas.
- **Listas no ordenadas:** Muestra las entradas de la lista en una lista con viñetas. Utiliza una etiqueta `<ul>`. Los elementos de una lista están marcados con `<li>`, que indica un elemento ordinario de la lista. Echemos un vistazo a algunos ejemplos, en el primer código elaboramos una lista ordenada y luego se muestra un código para lista una desordenadas.

```
<p>Marcas de Coches más caros:</p>
<ol>
  <li>Ferrari</li>
  <li>Lamborghini</li>
  <li>Bugatti</li>
</ol>
```



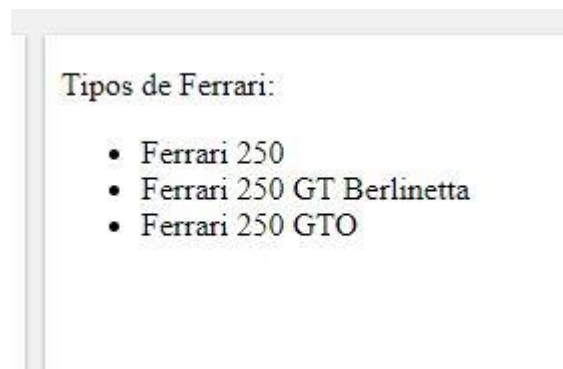
Y se mostrará:



Y para la lista desordenada:

```
<p>Tipos de Ferrari:</p>
<ul>
  <li> Ferrari 250</li>
  <li> Ferrari 250 GT Berlinetta</li>
  <li> Ferrari 250 GTO </li>
</ul>
```

Se mostrará:



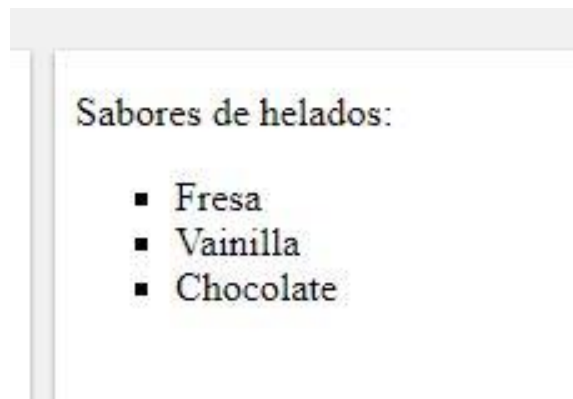
### 3.2. Eligiendo y Configurando Etiquetas de HTML5 para Elementos de Entrada y Validación

Se pueden cambiar los símbolos de viñetas redondas en una lista desordenada simplemente añadiendo un atributo para cambiar la naturaleza de los símbolos.

Para símbolos cuadrados, añade **type="square"** al **<ul>**. Para círculos vacíos agregar **type="circle"**. También se pueden añadir los atributos al listar ítems (marcándolos con **<li>**) para afectar puntos individuales. Por ejemplo, para mostrar todos los símbolos de los apartados como cuadrados rellenos:

```
<p>Sabores de helados:</p>
<ul type="square">
  <li> Fresa </li>
  <li>Vainilla</li>
  <li>Chocolate</li>
</ul>
```

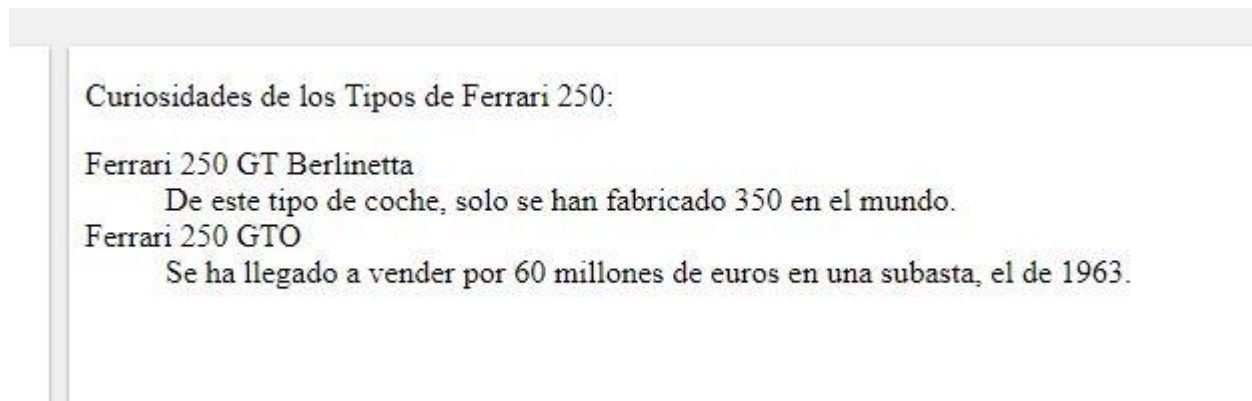
Y se verá:



Otro tipo de lista es la lista de definición. Muestra los ítems con sus definiciones debajo de la lista y con sangría. La etiqueta `<dl>` define la lista, la etiqueta `<dt>` marca cada término en el elemento, y la etiqueta `<dd>` define cada descripción. He aquí un ejemplo de la marcación:

```
<p>Curiosidades de los Tipos de Ferrari 250:</p><dl>
  <dt>Ferrari 250 GT Berlinetta </dt>
  <dd> De este tipo de coche, solo se han fabricado 350 en el mundo.</dd>
  <dt> Ferrari 250 GTO </dt>
  <dd> Se ha llegado a vender por 60 millones de euros en una subasta, el de
1963.</dd>
</dl>
```

Y se mostrará como:



*HTML5* define un grupo de filas de encabezado en una tabla para fines de formato y desplazamiento con botones. El *W3C* prefiere que utilice el elemento de menú sólo para menús contextuales, listas de controles y comandos de formulario, barras de herramientas y elementos similares.

### 3.2.1. Entendiendo los Elementos de Entrada y Formularios

En *HTML*, la entrada y la validación se aplican a los formularios. Un **formulario Web** es una página Web que proporciona campos de entrada para que un usuario introduzca datos, que se envían a un servidor para su tratamiento posterior. A partir de ahí, la información se almacena en una base de datos o se envía a un destinatario.

Los formularios web se utilizan como interfaz para muchas tareas diferentes:

- Para iniciar sesión en un sitio web, servidor o red.
- Para recopilar información de contacto, como nombre, dirección de correo electrónico, número de teléfono y dirección postal.
- Para suscribirse a los correos electrónicos o boletines de una organización.
- Para capturar los comentarios de los usuarios después de un artículo en un sitio Web
- Para seleccionar preferencias en una página Web
- Para introducir los datos de la reserva

Muchas aplicaciones cliente utilizan algún tipo de formulario Web para interactuar con el usuario. Los elementos de entrada *HTML* se utilizan para crear la interfaz de un formulario y garantizar que se recopila la información de los usuarios de forma consistente. La validación asegura que la información introducida está en el formato correcto y utilizable antes de enviar los datos al servidor.

Las **entradas de los formularios** son la información que un usuario introduce en los campos de un formulario Web o de una solicitud al cliente.

*HTML5* introduce varios atributos nuevos de formularios y elementos de entrada, tales como **url** para introducir una sola dirección Web, correo electrónico para una sola dirección de correo electrónico o una lista de correo electrónico y buscar para pedir a los usuarios que introduzcan el texto que desean buscar. Los nuevos atributos hacen que el desarrollo de formas sea mucho más fácil que en el pasado, esto solía requerir un montón de scripts y ahora puede realizarse mediante etiquetas *HTML5*.

Por otro lado, lamentablemente, muchos de los nuevos atributos aún no son compatibles con todos los navegadores principales.

Sin embargo, si se utiliza un nuevo elemento o atributo que aún no está soportado, el navegador hace lo que se llama "**falls back**" a una pantalla alternativa, una forma diferente de entrada, u otros.

*HTML5* introduce dos nuevos atributos para el elemento de formulario: **autocomplete** que se llama **autocomplete** y el de no validar que se llama **novalidate**.

Todos los atributos para el elemento de formulario se enumeran en la siguiente, con los nuevos atributos indicados con un doble asterisco.



Atributo	Valor	Descripción
<b>accept-charset</b>	<b>character_set</b>	Especifica un conjunto de codificaciones de caracteres en el directorio que el servidor acepta
<b>action</b>	<b>URL</b>	Especifica la dirección Web a la que se enviará el formulario los datos se envían
<b>autocomplete**</b>	<b>on</b> <b>off</b>	Especifica si la función de autocompletar está activada o desactivada
<b>enctype</b>	<b>application/x-www-form-urlencoded</b> <b>multipart/form-data</b> <b>text/plain</b>	Especifica el tipo de codificación para los datos de formulario al enviar los datos a un servidor; multipart/ se usa sólo para <b>method="post"</b>
<b>method</b>	<b>get</b> <b>post</b>	Especifica el método <i>HTTP</i> (transmisión) usado cuando se envían los datos del formulario; use "get" para recuperar datos y utilizar "post" para almacenamiento o actualización de datos o envío de correo electrónico
<b>name</b>	<b>text</b>	Especifica el nombre de un formulario, que se utiliza para hacer referencia a los datos del formulario
<b>novalidate**</b>	<b>novalidate</b>	Un atributo booleano que especifica que el atributo los datos del formulario (entrada del usuario) no deben ser validados cuando se envía; HTML5 también permite a los atributos booleanos que se establezcan mencionando el atributo sin signo igual o valor asignado
<b>Target</b>	<b>_blank</b> <b>_self</b>	Especifica dónde mostrar la respuesta recibido después de enviar el formulario:

	<b>_parent</b> <b>_top</b>	_ <b>blank</b> carga la respuesta en un nuevo archivo, ventana del navegador sin nombre _ <b>self</b> carga la respuesta en la corriente es la ventana por defecto, por lo que se usa no es necesario _ <b>parent</b> carga la respuesta en la ventana del formulario _ <b>top</b> carga la respuesta en su totalidad ventana del navegador
--	-------------------------------	--

*HTML5* introduce numerosos atributos de elementos de entrada. Los atributos para el elemento de entrada se enumeran en la tabla siguiente; los nuevos atributos en *HTML5* se indican con un doble asterisco.

Atributo	Valor	Descripción
<b>accept</b>	audio/* video/* image/* MIME_type	Especifica los tipos de archivo que acepta el servidor; se usa sólo para <b>type = "file"</b>
<b>alt</b>	<b>text</b>	Especifica que es un texto alternativo para las imágenes; sólo se utiliza para <b>type = "image"</b> ; se usa comúnmente cuando crear un botón Enviar personalizado desde su propio archivo de imagen
<b>autocomplete**</b>	<b>on</b> <b>off</b>	Especifica si la función de autocompletar está activada o desactivada
<b>autofocus**</b>	<b>autofocus</b>	Un atributo booleano, especifica que un control se centrará, o se seleccionará, tan pronto como el cargas de página
<b>checked</b>	<b>checked</b>	Especifica que se debe preseleccionar un elemento de entrada al cargar la página;



		se usa sólo para <b>type="checkbox"</b> o <b>type="radio"</b>
<b>disabled</b>	<b>disabled</b>	Desactiva un elemento de entrada
<b>form**</b>	<b>form_id</b>	Especifica el formulario al que pertenece un elemento de entrada
<b>formaction**</b>	<b>URL</b>	Especifica la dirección Web del archivo que se utilizará para procesar el control de entrada cuando el formulario es remitido
<b>formenctype**</b>	<b>application/x-www-form-urlencodedmultipart/form-data/text/plain</b>	Especifica el tipo de codificación para los datos del formulario cuando se envían los datos a un servidor; se utiliza sólo para <b>method="post"</b> .
<b>formmethod**</b>	<b>get</b> <b>post</b>	Especifica el método <i>HTTP</i> (transmisión) que se utiliza para enviar los datos del formulario a una dirección web
<b>formnovalidate**</b>	<b>formnovalidate</b>	Un atributo booleano que impide la validación al enviar información
<b>formtarget**</b>	<b>_blank</b> <b>_self</b> <b>_parent</b> <b>_top</b> <b>framename</b>	Especifica una palabra clave que indica dónde mostrar la respuesta recibida después de enviar el formulario
<b>height</b>	<b>pixels</b>	Especifica la altura de un elemento de entrada; se utiliza sólo con entrada <b>type="image"</b>
<b>list**</b>	<b>datalist_id</b>	Se refiere a un elemento de la lista de datos que contiene un



		contenido predefinido para autocompletar la entrada, como seleccionar una posición de un menú desplegable
<b>max**</b>	<b>number date</b>	Especifica el valor máximo de una entrada aspecto
<b>min**</b>	<b>number date</b>	Especifica el valor mínimo para un elemento de entrada
<b>multiple**</b>	<b>multiple</b>	Un atributo booleano que especifica que el usuario puede introducir varios valores
<b>pattern**</b>	<b>regexp</b>	Proporciona un formato (una expresión regular) para el campo de entrada; el valor del elemento de entrada es comprobado contra la expresión regular
<b>placeholder**</b>	<b>text</b>	Muestra una palabra clave o frase corta que describe el valor esperado de una entrada como, por ejemplo, "Email" para una entrada de correo electrónico  el campo; el marcador de posición desaparece cuando el usuario introduce datos
<b>readonly</b>	<b>readonly</b>	Restringe un campo de entrada a sólo lectura
<b>required**</b>	<b>required</b>	Un atributo booleano que requiere un campo de entrada que debe rellenarse antes de enviar el formar
<b>size</b>	<b>number</b>	Especifica el ancho de un elemento de entrada, en número de caracteres
<b>src</b>	<b>URL</b>	Especifica la dirección Web de la imagen usada como un botón de envío; se usa sólo para <b>type="imagen"</b>



<b>step**</b>	<b>number</b>	Especifica el número de intervalos aceptados para un elemento de entrada; se puede utilizar con el atributos <b>min</b> y <b>max</b> para crear un rango de valores
<b>type</b>	button, checkbox, color, date, datetime, datetime-local, email, file, hidden, image, month, number, password, radio, range, reset, search, submit, tel, text, time, url, week	Especifica el tipo de elemento de entrada a mostrar
<b>value</b>	<b>text</b>	Especifica el valor de un elemento de entrada
<b>width</b>	<b>pixels</b>	Especifica el ancho de un elemento de entrada; se utiliza sólo con entrada <b>type="image"</b>

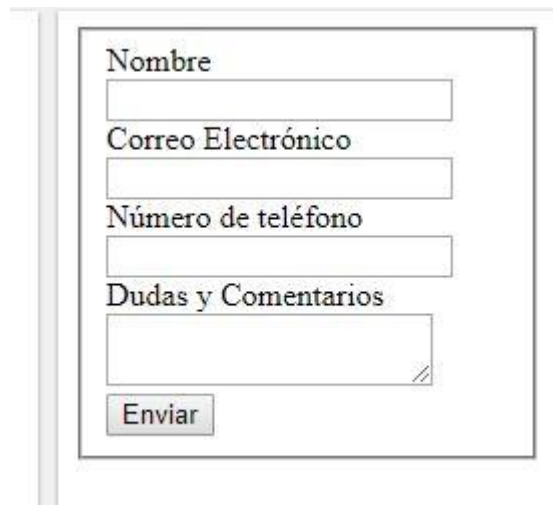
#### a. Explorando Formas de Creación, Atributos para Elementos de Entrada y Valores

Para crear un formulario, se deben utilizar las etiquetas `<form>` de **inicio** y **final**. Todos los contenidos y campos del formulario van entre las dos etiquetas `<form>`. La mayoría de los formularios también incluyen el atributo `id` en la etiqueta de inicio, como se indica a continuación:

```
<formulario id="palabra clave">
  <Contenido y campos> <contenido>
</form>
```

El elemento **fieldset** se utiliza con muchos formularios para agrupar elementos relacionados.

A continuación, podemos ver un ejemplo:



Si el formulario está incluido en un documento *HTML* con otros elementos, se puede utilizar la etiqueta `<div>` al principio y al final del formulario para separarlo de otros contenidos. Usando la etiqueta `<div>` también permite incluir el formato en línea, si el formulario utiliza etiquetas para alinear campos verticalmente cortos y simples.

La etiqueta `<div>` utiliza el atributo `id` y aparece antes de la primera etiqueta `<form>`.

A continuación, un ejemplo:

```
<div id="contact-form" style="font-family:'Arial Narrow','Nimbus Sans L',sans-serif;">
  <form id="contact" method="post" action="">
    <fieldset>
      <label for="name">Nombre</label>
      <input type="text" name=" Nombre " />
    </fieldset>
    <fieldset>
      <label for="email">Correo electrónico</label>
      <input type="email" name=" Correo electrónico " />
    </fieldset>
  </form>
</div> <!--fin del formulario-->
```

Y se mostrará:

Nombre	<input type="text"/>
Correo electrónico	<input type="text"/>

El **atributo required** obliga a que en ese campo exista información cuando se envía el formulario. El **atributo email** (mostrado en el ejemplo siguiente) requiere que el usuario introduzca un email. El navegador alertará al usuario con un mensaje de error para solucionar estos problemas.

Un ejemplo de un elemento de entrada con los atributos **required** y **email** es:

```
<input type="email" required />
```

Para hacer un formulario más fácil de usar, es útil agregar texto de marcador de posición. El **Placeholder text**, es decir, marcador de posición es texto mostrado dentro de un campo de entrada cuando el campo está vacío. Ayuda a los usuarios a entender el tipo de información que deben introducir o seleccionar. Al hacer click o saltar al campo de entrada y empezar a escribir, el texto recién ingresado reemplaza el texto del marcador de posición. Un ejemplo del marcador de posición es el atributo:

```
<input name="fName" placeholder="First Name" />
```

El **atributo pattern** proporciona un formato (una expresión regular) para un campo de entrada, que se utiliza para validar lo que se ha introducido en el campo. Por ejemplo, supongamos que se tiene un campo de entrada para el ID de empleado. Cada identificación de empleado comienza con dos letras mayúsculas seguidas de cuatro dígitos. Se deberá usar un campo de entrada de texto con los atributos requeridos y el patrón para asegurarse de que el campo (1) se rellena cuando el usuario hace clic en el botón de enviar y (2) contiene un valor que coincide con el formato correcto para un ID de empleado. Si el usuario pasa el ratón por encima de la tecla se muestra el mensaje en el atributo **title**, que se añade por separado. Un ejemplo del atributo **pattern** es:

```
<input type="text" id="empID" name="EmpleadoID"
  required pattern="[A-Z]{2}[0-9]{4}"
  title=" La identificación del empleado es seguida de dos letras mayúsculas por cuatro
  dígitos ">
```

**Pattern** se puede usar con los atributos de entrada como: **text**, **search**, **url**, **telephone**, **email** y **password**.

El **elemento datalist** permite presentar al usuario una lista desplegable de opciones para seleccionar. Sólo se pueden seleccionar las opciones de la lista. Alternativamente, se puede insertar

**type="text"** en el **elemento de entrada** para crear un cuadro de texto en el que el usuario introduce el texto.

El siguiente ejemplo permite al usuario seleccionar uno de los tres países:

```
<input id="país" name="país"
size="30" list="países" />
<datalist id=" países ">
  <option value="Estados Unidos">
  <option value="Canadá">
  <option value="Reino Unido">
</datalist>
```

El valor de búsqueda para el atributo **type** permite crear una función de búsqueda para una Web página. Un ejemplo de esta etiqueta es:

```
<form>
  <input name="search" required>
  <input type="submit" value="Buscar">
</form>
```

Finalmente, el atributo **autofocus** mueve el foco a un campo de entrada en particular. Un ejemplo de **autofocus** es cuando se abre una página Web de un motor de búsqueda y aparece automáticamente el cuadro de entrada para que se puedan escribir los términos de búsqueda. Un ejemplo de la marca de revisión para colocar el foco en un campo llamado **fname** cuando se carga una página:

```
<input type="text" name="fname" autofocus="autofocus" />
```

El autoenfoco ha sido manejado históricamente por *JavaScript*, y si un usuario desactiva *JavaScript* en una Web la función de enfoque automático no funciona. Para solucionar este problema, el autoenfoco *HTML5* es compatible con todos los principales navegadores y se comporta de forma coherente en todos los sitios web.

### 3.2.2. Entendiendo las Validaciones

La validación es el proceso de verificar que la información introducida o capturada en un formulario se encuentra en el formato correcto y utilizable antes de enviar los datos al servidor. Algunas cosas que se verifican durante la validación:

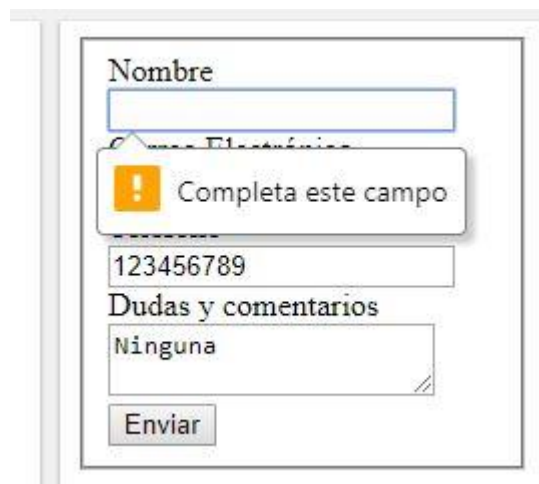
- Los campos obligatorios están vacíos.
- Las direcciones de correo electrónico son válidas.
- Las fechas son válidas.
- El texto no aparece en un campo numérico o viceversa.

En *HTML5*, varios de los tipos de elementos de entrada sobre los que se aprendió en la última sección ofrecen **validación automática** desde la entrada, lo que significa que el navegador comprueba los datos que el usuario introduce. Esto se denomina **validación del lado del cliente**, ya que los datos de entrada se validan antes



de su presentación al servidor. (En los casos en que el servidor valida los datos recibidos de un formulario de entrada, se conoce como **validación del lado del servidor**). Si el usuario introduce un tipo incorrecto de datos en un campo, como una dirección de correo electrónico en un campo con el atributo **url**, el navegador instruye al usuario para que introduzca una URL válida. A continuación, se verán ejemplos de los mensajes de error predeterminados que se generan durante el proceso de validación automática.

El atributo **required** evita el problema de los campos vacíos que deben rellenarse. Cuando un usuario omite un campo obligatorio y hace clic en el botón *Enviar*, aparece un mensaje de error como el que se muestra a continuación en la figura. Este ejemplo utiliza el navegador *Web Mozilla Firefox*.




*HTML5* también ofrece la validación de las direcciones Web introducidas en los campos con la directiva `<input type="url">`, y los números introducidos en los campos con `<input type="number">`. Si se utilizan los atributos **min** y **max** con `<input type="number">`, se recibirá un mensaje de error del navegador si se introduce un número demasiado pequeño o demasiado grande.

Finalmente, el atributo **pattern** evita que el usuario ingrese datos que no siguen la expresión del patrón. En este ejemplo, el atributo **pattern** valida un código zip de cinco dígitos:

```
<input type="text" name="zipcode" pattern="[0-9]{5}" title=" El código zip debe contener 5 dígitos" />
```

La introducción incorrecta de datos en el campo Zip Code del navegador *Firefox* provoca el mensaje de error que se muestra en la siguiente imagen:

Introduce archivo zip:

**Nota:** El  Utiliza un formato que coincida con el solicitado para hacer la validación de un campo.  
El código zip debe contener 5 dígitos

Como se mencionó anteriormente, no se requiere ninguna marca de revisión para activar la validación de formularios *HTML5*; está activada de forma predeterminada. Para desactivarlo, se utiliza el atributo **novalidate** para campos de entrada específicos.



## 4. Entendiendo CSS Básico: Contenido Flotante, Posicionamiento y Estilos

---

### 4.1. Breve Historia de CSS

Las hojas de estilos aparecieron poco después de que el lenguaje de etiquetas SGML, alrededor del año 1970. Desde la creación de SGML, se observó la necesidad de definir un mecanismo que permitiera aplicar de forma consistente diferentes estilos a los documentos electrónicos.

El gran impulso de los lenguajes de hojas de estilos se produjo con el boom de Internet y el crecimiento exponencial del lenguaje *HTML* para la creación de documentos electrónicos. La guerra de navegadores y la falta de un estándar para la definición de los estilos dificultaban la creación de documentos con la misma apariencia en diferentes navegadores.

El organismo *W3C* (World Wide Web Consortium), encargado de crear todos los estándares relacionados con la web, propuso la creación de un lenguaje de hojas de estilos específico para el lenguaje *HTML* y se presentaron nueve propuestas. Las dos propuestas que se tuvieron en cuenta fueron la *CHSS* (*Cascading HTML Style Sheets*) y la *SSP* (*Stream-based Style Sheet Proposal*).

La propuesta *CHSS* fue realizada por Håkon Wium Lie y *SSP* fue propuesto por Bert Bos. Entre finales de 1994 y 1995 Lie y Bos se unieron para definir un nuevo lenguaje que tomaba lo mejor de cada propuesta y lo llamaron *CSS* (*Cascading Style Sheets*).

En 1995, el *W3C* decidió apostar por el desarrollo y estandarización de *CSS* y lo añadió a su grupo de trabajo de *HTML*. A finales de 1996, el *W3C* publicó la primera recomendación oficial, conocida como "CSS nivel 1".

A principios de 1997, el *W3C* decide separar los trabajos del grupo de *HTML* en tres secciones: el grupo de trabajo de *HTML*, el grupo de trabajo de *DOM* y el grupo de trabajo de *CSS*.

El 12 de mayo de 1998, el grupo de trabajo de *CSS* publica su segunda recomendación oficial, conocida como "CSS nivel 2". La versión de *CSS* que utilizan todos los navegadores de hoy en día es *CSS 2.1*, una revisión de *CSS 2* que aún se está elaborando (la última actualización es del 8 de septiembre de 2009). Al mismo tiempo, la siguiente recomendación de *CSS*, conocida como "CSS nivel 3", continúa en desarrollo desde 1998 y hasta el momento sólo se han publicado borradores.

La adopción de *CSS* por parte de los navegadores ha requerido un largo periodo de tiempo. El mismo año que se publicó *CSS 1*, *Microsoft* lanzaba su navegador Internet Explorer 3.0, que disponía de un soporte bastante reducido de *CSS*. El primer navegador con soporte completo de *CSS 1* fue la versión para *Mac* de *Internet Explorer 5*, que se publicó en el año 2000. Por el momento, ningún navegador tiene soporte completo de *CSS 2.1*.

### 4.2. Entendiendo CSS

*CSS* es una herramienta crucial para conseguir mucha de la apariencia e incluso comportamiento de las aplicaciones modernas de móvil, así como sitios Web. Para construir el "front end" de una app o sitio Web, y, especialmente, para mantener una apariencia correcta y "original", cuando se realizan cambios funcionales a la app o sitio Web durante su uso, se necesita entender bien *CSS* y cómo *CSS* coopera con otras herramientas incluidas en *HTML* y *JavaScript*. Se estará también en una posición mucho mejor para estimar el esfuerzo



requerido por proyectos particulares cuando se asimilen por completo los conceptos de “estilo” de interfaz de usuario como los usa *CSS*.

Se puede crear una apariencia detallada, mostrando letras individuales en *cursiva*, **negrita**, o incluso en color; las letras tendrán un **tamaño** particular, y se eligen de una fuente específica; y más efectos especiales están al alcance de los desarrolladores Web más avanzados. Todos esos elementos de presentación, en contraposición al contenido, son parte del estilo del sitio Web. **Cascading Style Sheets (CSS)** es un lenguaje que define el estilo de los elementos *HTML*.

¿Qué significa “cascading” en este caso? Uno de los principios originales de *HTML* es que la apariencia de un elemento es controlada, no solo por *CSS*, si no también por la forma que el usuario configura su navegador o escritorio. un usuario con una deficiencia visual puede, por ejemplo, pedir que el contenido sea mostardo en un tamaño de fuente particular.

**Nota:** es común para los diseñadores Web llamar al archivo que tiene las reglas *CSS* como hoja de estilos o “el *CSS*”. Algunos códigos de estilos y programadores le llaman “fuente *CSS*” o “archivo *CSS*”.

Como se ha aprendido en lecciones anteriores, la responsabilidad de *HTML* es estructurar el contenido; la responsabilidad de *CSS* es dar estilo al contenido. **CSS3** es la versión de *CSS* que corresponde con *HTML5*, y muchos navegadores webs modernos soportan *CSS3*, muchos desarrolladores ya están incorporando *CSS3* en sus sitios Web y aplicaciones.

La base de este libro es que en las herramientas modernas incluyen los mismos estándares de construcción que las aplicaciones móviles: *HTML*, *CSS* y *JavaScript* te ayudan a construir apps también.

La ventaja de *CSS3* es que es compatible con versiones anteriores de *CSS*, así que se puede usar *CSS3* con la página web existente sin tener que hacer cambios. *CSS3* generalmente añade características y funcionalidades más que cambiar como se ha usado *CSS*.

Algunas de las adiciones más importantes a *CSS3* son los selectores, los modelos *box*, las transformaciones 2D y 3D, las animaciones y los múltiples layout de columnas. *CSS3* también permite crear bordes redondeados, añadir sombras a cajas y texto, usar imágenes múltiples de fondo, y usar cualquier fuente que se quiera, tanto si está en el ordenador del usuario como si no.

### 4.3. Usando las Herramientas Apropriadas

**Nota:** se pueden crear ficheros *CSS* enteramente con un simple editor de texto como Notepad, aunque tiene sus limitaciones ya que de origen no fue diseñado para ser un editor. Muchos editores *HTML* y herramientas de desarrollo de aplicaciones tienen una aplicación de depuración que ayudan rápidamente a encontrar errores en el código. Estas herramientas usualmente también incluyen un botón para abrir un navegador web más que tener que hacerlo manualmente.

Cuando se empieza a trabajar con *CSS*, se decide qué herramientas de edición se necesita, se puede trabajar tanto con un programa construido con Notepad en Windows hasta con un entorno de desarrollo integrado (IDE) para móviles.

Como cuando se trabaja con *HTML*, en algún punto, habrán, al menos, dos aplicaciones abiertas:

- Un editor que puede ser *Notepad*, *Microsoft Visual Studio*, *Microsoft Expression Blend*, *Expression Studio*, *notepad++* para Windows o *textwrangler* para Mac OS, *Microsoft Web Matrix*, u otras herramientas.
- Un navegador web, como *Internet Explorer* o *Firefox*.



#### 4.4. Explorando la Conexión entre HTML y CSS

**Nota:** el elemento `<link>` conecta un fichero *HTML* con un fichero *CSS*. Esta sección cubre brevemente estilos *CSS* y cómo los ficheros *HTML* y *CSS* se conectan.

Cuando se crea una página *HTML* y se quieren añadir estilos de un fichero *CSS*, se debe incluir el elemento `<link>` al fichero *CSS* en la página *HTML* (se pueden referenciar más de un *CSS* en una página *HTML*). Un ejemplo del uso de `<link>`:

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

Un fichero *HTML* debe tener un formato como *myproject.HTML* o *file1.htm*; un archivo *CSS* suele tener *myproject.css*. El contenido también es diferente: la fuente de *HTML* está organizada por etiqueta, mientras que *CSS* se organiza por reglas.

En un proyecto, se pueden crear diferentes archivos *CSS* para cumplir los requisitos. Para referenciar diferentes archivos como `href = "mytheme.css"` es importante recordar que, si se escribe incorrectamente el archivo *CSS* o el estilo de página, o `"text/css"` dentro del código *HTML*, la página web no aplica ninguno de los estilos, ya que trata la conexión como perdida. El comportamiento por defecto es no advertir el error.

#### 4.5. Separando Contenido de Estilo

Se pueden crear páginas "puramente" *HTML*, es decir, sin fichero *CSS*. *HTML* tiene la habilidad de especificar cursiva, color y más. Los sitios web simples, algunas veces, no usan *CSS*.

**Nota:** incluso algo tan simple como colorear un bloque de texto se puede hacer de muchas maneras. Se puede observar en *HTML* algo como `<div style = 'color: green'...>`, llamado "in-line", o `div {color: green...}` en *CSS*, u otras muchas posibilidades y combinaciones.

No obstante, separar *HTML* y *CSS* en diferentes ficheros es un concepto fundamental que se debe comprender, el contenido se maneja por *HTML* y los estilos por *CSS*. En muchos proyectos, hay personas que se encargan de los ficheros *HTML* y otros de los ficheros *CSS*, lo que permite trabajar simultáneamente en ambos ficheros sin interferir el uno en el otro.

**Nota:** se debe tener en cuenta que para aprender las correspondencias de *CSS* a una página web, se debe tener como mínimo un archivo *HTML*. No se puede aprender *CSS* en un archivo aislado de las otras partes.

#### 4.6. Entendiendo Selectores y Declaradores

Cada regla tiene dos partes: un o varios selectores y la declaración. El *selector* es el elemento *HTML* que se desea estilizar. La *declaración* es el estilo para el selector, que contiene una propiedad, que es el atributo de estilo, y un valor. La siguiente imagen muestra un ejemplo de declaración:

Selector	Declaración
p	{color:brown;}
	Property Value

A continuación, se considera el siguiente ejemplo:



```
/* Este es el contenido del fichero e1.css. */
p {color: brown;}
#eslogan {
    font-size:20px;
    color: green;
    font-style: italic;
}
```

**Nota:** el contenido entre `/*...*/` en un fichero CSS está comentado, es decir, contiene información, insertada por el desarrollador, que no afecta al código.

El ejemplo anterior tiene dos reglas, una para `p` y otra para `#eslogan`. La primera se aplica a todo el contenido con la etiqueta `<p>` en la página web. La segunda se aplica solo al elemento *HTML* con el ID "eslogan". La parte del selector de una regla puede ser complicada, de momento, cuando se piense en selectores se debe pensar en algunos de los siguientes casos:

- Etiquetas *HTML* como `<p>`.
- Selectores ID como `#eslogan`. El símbolo `#` es el prefijo que determina que las selecciones son por ID. Se debe tener en cuenta que un elemento ID deber ser único dentro de una instancia *HTML* particular.
- Selectores de clase, como los del siguiente ejemplo:  
`class` es un atributo que el desarrollador web usa para proporcionar estructura al documento y que puede ser compartida por varios elementos. El selector es `.class`.

```
<!doctype html>
<html>
<head>
    <title>Un ejemplo de class</title>
    <link href = "e2.css" rel = "stylesheet" type = "text/css">
</head>
<body>
    <h1>Acerca de estados</h1>
    <p class = "fact"> Alaska es el estado mas grande de los EE.UU en superficie.</p>
    <p class = "opinion"> Nueva Jersey merece su apodo "Garden State."</p>
    <p class = "fact"> Un solo congresista representa Wyoming en la Camara de Representantes
nacional.</p>
</body>
</html>
```

Este es el contenido de un archivo llamado `e2.css`:

```
p {color: black;}
/* El prefijo para un selector de clase es un punto: '.' */
.opinion {color: gray;}
```

Con estos dos archivos, abriendo `e2.html` se produce la siguiente vista:



## Acerca de estados

Alaska es el estado mas grande de los EE.UU en superficie.

Nueva Jersey merece su apodo "Garden State."

Un solo congresista representa Wyoming en la Camara de Representantes nacional.

**Nota:** se debe tener cuidado con los IDs, ya que deben ser únicos en el HTML. Si se usan muchos ID a lo largo de los diferentes archivos, puede ser una tarea complicada encontrar aquel ID al que se hace referencia.

### 4.7. Entendiendo Fuentes y Familias de Fuentes

Una **fuer**te es un conjunto de caracteres de un tamaño y estilo particular. **Monospace** suele usarse para material técnico como fórmulas, números, código, etc. Serif son los detalles al final de una fuente, como por ejemplo **Sans serif** son tipos de estilos dibujados sin serifs, como por ejemplo la fuente **Arial**.

La forma principal de especificar fuentes en un archivo CSS es usar la propiedad **font-family**. Esta propiedad puede declarar tanto una fuente específica como en una familia más grande que incluya diferentes fuentes. Suele ser más seguro especificar una familia amplia cuando se trabaja, ya que es difícil predecir con qué fuente particular se trabajará.

Por ejemplo, cuando se incluye **font-family: monospace** en un fichero CSS, se le dice al navegador que elija caracteres donde cada letra ocupa el mismo ancho en una línea, ya sea una "m" o una "i". No se está especificando una fuente monospace única, se está especificando una familia.

Antes de CSS3, los desarrolladores tenían que usar fuentes seguras o fuentes que el desarrollador sabía que estaba instalada en la página web.

CSS3 aporta la regla **@font-face**, que permite al desarrollador usar cualquier la fuente que elija. Primeramente, se debe crear una regla **font-face**, asignando un nombre a la fuente, que debe estar localizado en el servidor web, o se debe incluir una URL donde la fuente está localizada en un servidor web diferente. Aquí hay un ejemplo de una regla para una fuente llamada Euphemia que está localizada en el servidor web:

```
@font-face
{
    font-family: TrustyHomePage;
    src: url('Euphemia.ttf'),
}
```

Al igual que en imágenes, se debe tener una copia legal de la fuente que se crea en la página web.

### 4.8. Manejando Contenido Flotante

El contenido flotante es un concepto **HTML** fundamental, tiene que ver con llenar las líneas horizontales de izquierda a derecha a lo largo de la pantalla, y separar líneas de arriba a bajo a medida que uno se mueve por la pantalla.



Se consideran dos alternativas cuando se muestra un elemento visual:

- **Inline flow:** rellena el ancho que se requiere.
- **Block flow:** rellena solamente ancho que esté disponible.

Es importante el control de la geometría de la interfaz de usuario, y particularmente de la extensión horizontal de la pantalla, así como entender el concepto de flotante.

**Inline flow** no fuerza nuevas líneas antes o después del elemento incrustado, sino que simplemente coloca el elemento entre el contenido antes y después del elemento inlineado.

Si se observa que en una misma frase, una palabra aparece en **negrita** y otra está en *cursiva*, en un código **HTML** las palabras en negrita y cursiva se ponen con las etiquetas **<b>** e **<i>**. Estos elementos corresponden al tipo *inline* y ocupan tanto espacio en las líneas de texto como sean necesarias, y no se muestran en líneas nuevas.

En un **block flow**, en contraste a un *inline flow*, un elemento se separa de otros elementos por nuevas líneas arriba y abajo, y rellena de izquierda a derecha la extensión horizontal donde aparece.

El siguiente párrafo es un elemento *block-flow*: arriba y debajo de él hay nuevas líneas, y el párrafo se rellena de izquierda a derecha.

## 4.9. Posicionando Elementos Individuales

El método de posicionamiento por defecto para todos los elementos es "static", es decir, inmediatamente después del anterior elemento del documento. Para posicionar cualquier elemento en otra posición, se usa **CSS** para cambiar a flotante o a absoluta.

### 4.9.1. Aplicando Posicionamiento Flotante

**Float positioning** se usa a menudo cuando un layout (concepto que se explicará más adelante) está en columnas, o al menos, en parte. Tener un elemento como flotante significa moverlo lo más lejos posible tanto a derecha como a izquierda; entonces el texto se envuelve alrededor del elemento.

Simple columnas se construyen haciendo flotantes diferentes elementos uno detrás de otro. Si se supone que se necesita crear un layout de cuatro columnas de un contenido de texto, se estiliza cada una de las piezas del contenido que deberían aparecer en columnas sucesivas como posicionadas flotantes; cada elemento "flota" al lado, pero mantiene la separación de los anteriores y posteriores. En este tipo de columna, el texto que sobresale de una columna no flota hasta el inicio de la siguiente columna.

### 4.9.2. Aplicando Posicionamiento Absoluto

Con **absolute positioning**, un elemento se elimina de su posición dentro del cuerpo del elemento y se posiciona en otra posición geométrica de la vista. En este caso, posición geométrica quiere decir una localización a una distancia definida de los lados de la pantalla.

El posicionamiento absoluto no suele usarse en proyectos web tanto como el posicionamiento flotante. No obstante, en aplicaciones móviles sí que suele usarse posicionamiento absoluto debido al tamaño de la ventana.

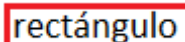
Para mayor información acerca de flotamiento en **CSS** y propiedades de posición ir a las páginas:



[http://www.w3schools.com/css/css\\_float.asp](http://www.w3schools.com/css/css_float.asp)  
[http://www.w3schools.com/css/css\\_positioning.asp](http://www.w3schools.com/css/css_positioning.asp)  
[http://www.w3schools.com/cssref/pr\\_class\\_position.asp](http://www.w3schools.com/cssref/pr_class_position.asp)

## 4.10. Manejando Contenido Desbordante

Las letras y caracteres de una página se deben ver como marcas, en vez de como letras, ya que cada elemento *HTML* ocupa un rectángulo. Tómese la palabra "rectángulo" como ejemplo, algunas letras sobresalen como las letras "l" o "t" por arriba, y "g" por debajo. Para el propósito del layout *HTML*, "rectángulo" cabe dentro de un pequeño rectángulo (**bounding box**) que incluye todas las letras de la palabra junto con su fondo, como se muestra en la siguiente imagen. El estilo *CSS* se expresa en términos de esta caja.



En particular, *CSS* hace posible limitar el ancho de un elemento, pero, ¿qué pasa si un elemento no cabe en el espacio que *CSS* define para él? La regla **overflow** lo controla.

### 4.10.1. Entendiendo Scrolling Desbordante

Cuando un elemento desborda su caja, y su desbordamiento está configurado para desplazarse, todo el contenido permanece dentro de la caja; ninguno de los desbordamientos aparece fuera de la caja. Esto es un **scrolling overflow**.

El contenido tiene que permanecer dentro de la caja, pero no encaja; ¿qué puede resolver un conflicto de este tipo? Se supone que la caja se mira hacia abajo en un área más grande, y que el espectador se puede mover con el scroll alrededor de esta área. Esa es una manera para que el espectador alcance todo el contenido. Se puede lograr esto usando el valor de desplazamiento con la propiedad **overflow**.

### 4.10.2. Entendiendo el Desbordamiento Visible y Oculto

La sección anterior demostraba cómo la barra de scroll daba al usuario la oportunidad de ver todo el contenido sin la necesidad de añadir más espacio a la vista. Otras dos tácticas son el uso de las propiedades de **overflow**, **visible** y **hidden**: **visible overflow** escribe encima del contenido que sigue y **hidden overflow** hace el desbordamiento invisible.

El desbordamiento oculto mantiene un diseño bajo control: garantiza que el desbordamiento no "contamine" un diseño agradable con piezas fuera de lugar. Por otro lado, lo oculto puede dar lugar a sorpresas. Si un usuario final con problemas de visión, por ejemplo, especifica una fuente más grande de lo que esperaba, el uso de desbordamiento oculto puede hacer que elementos cruciales de su diseño sean completamente invisibles e inaccesibles; en el peor de los casos, el usuario final podría enfrentarse a una pantalla sin controles visibles o formas de navegar de vuelta a página de inicio.

Para más detalles sobre desbordamiento en *CSS*, visitar: [http://www.w3schools.com/cssref/pr\\_pos\\_overflow.asp](http://www.w3schools.com/cssref/pr_pos_overflow.asp)



## 5. Entendiendo CSS Básico: Layouts

---

### 5.1. Organizando el Contenido de la Interfaz de Usuario (UI) usando CSS

Una **interfaz de usuario (UI)** es la parte de un sitio Web o aplicación con la que interactúa un usuario. La interfaz de usuario tiene un layout que puede ir desde lo más simple con sólo un botón o dos hasta lo más complejo con muchas partes, y cada parte puede contener uno o más botones, menús y barras de herramientas, formularios, etcétera.

Con una gama tan amplia de layouts, crear interfaces de usuario que funcionen bien para sitios web y aplicaciones visualizadas en dispositivos móviles es todo un reto. El posicionamiento y el tamaño automático de los elementos de la interfaz de usuario se ha convertido en un elemento central del buen diseño. Por ejemplo, el posicionamiento relativo de los elementos de la interfaz de usuario es apropiado para muchas páginas web, y los desarrolladores web han utilizado la propiedad flotante durante años para lograr flexibilidad en sus diseños. Pero el posicionamiento relativo no funciona para la mayoría de las aplicaciones Web móviles porque da lugar a una inapropiada superposición de elementos o da lugar a que aparezcan en lugares equivocados.

Una combinación de posicionamiento absoluto y cajas flexibles (containers) funciona mucho mejor para aplicaciones Web para móviles y partes de páginas Web en general. Por ejemplo, un elemento de interfaz de usuario que debería aparecer siempre en el mismo lugar de la pantalla, como un encabezado o pie de página, debería utilizar el posicionamiento absoluto. Para mayor flexibilidad, el modelo CSS3 Flexbox Box es ideal para artículos que deben redimensionarse o reposicionarse (horizontal o verticalmente) dependiendo del tamaño de la pantalla. Otro modelo relacionado con la interfaz de usuario es el modelo CSS3 Grid Layout, que le proporciona un mayor control sobre layouts complejos que el modelo flexbox. Esta lección se centra en la caja Flexbox y modelos Grid Layout.

Se necesitan entender algunas cosas sobre CSS3 y compatibilidad con navegadores. Al igual que *HTML5*, la CSS3 está todavía en formato borrador y en proceso de modificación. Los nombres de algunas propiedades pueden cambiar de una versión del borrador de CSS3 a la siguiente, y se pueden introducir nuevos valores de propiedad mientras se eliminan otros.

Para ayudar a garantizar que los estilos CSS3 funcionen durante esta fase de transición, muchos de los principales navegadores Web ofrecen nombres de propiedad alternativos. Estas soluciones simplemente añaden un prefijo de proveedor, que es una palabra clave rodeada de guiones, delante del nombre de una propiedad CSS3. Se debe tener en cuenta lo siguiente al trabajar con propiedades CSS3:

- Internet Explorer usa el prefijo **ms**.
- Firefox soporta el prefijo **moz**.
- Opera soporta el prefijo **o**.
- Chrome y Safari soporta el prefijo **webkit**.

Por ejemplo, una propiedad CSS3 podría ser **flexbox**. Para aplicar el estilo **flexbox** a los elementos y visualizarlos en *Firefox*, se necesita utilizar la propiedad **-moz-flexbox**.





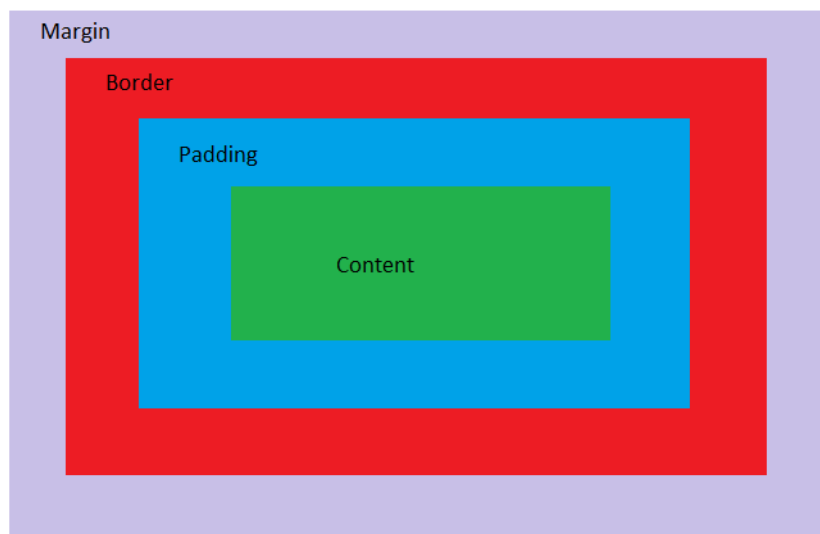
Una buena práctica es incluir los cuatro prefijos de los proveedores para las características de CSS3 que todavía están surgiendo. De esta manera, la página Web tiene más oportunidades de ser vista correctamente sin importar que navegador se utiliza. Sin embargo, incluir los cuatro prefijos de proveedor en el código hace el código más largo y no garantiza que la función CSS3 funcione en todos los navegadores. Si un navegador simplemente no soporta la característica o la propiedad del prefijo del vendedor, la característica no se mostrará correctamente. El sitio web "*When Can I Use*" en *caniuse.com* es de ayuda a determinar cuáles soportan características específicas de CSS3 y HTML5.

**Nota:** Otra práctica recomendada, especialmente durante la transición a CSS3, es utilizar familias de fuentes genéricas para evitar posibles problemas de visualización en diferentes navegadores. Una familia de fuentes monospace, ya sea serif o sans serif, es la mejor manera de evitar resultados inesperados. Sin embargo, CSS3 también ofrece la posibilidad de utilizar cualquier fuente que desee. La clave es aprender cuándo usar monospace y cuándo está bien usar una fuente más elegante.

### 5.1.1. Usando Flexbox para Layouts simples y usando Grid para Layouts complejos

Hace años, W3C creó especificaciones para un modelo de caja simple, llamado el modelo CSS Box. Este modelo describe las cajas que rodean el contenido de un documento HTML, ya sea que el documento se convierta en una página Web o en una aplicación Web. Piense en cada parte de un documento HTML como si estuviera en una caja. Cada caja debe ajustarse a las reglas definidas por el modelo de caja.

El modelo CSS Box se muestra en la siguiente imagen. Las partes del modelo CSS Box son margen, borde, relleno y contenido.



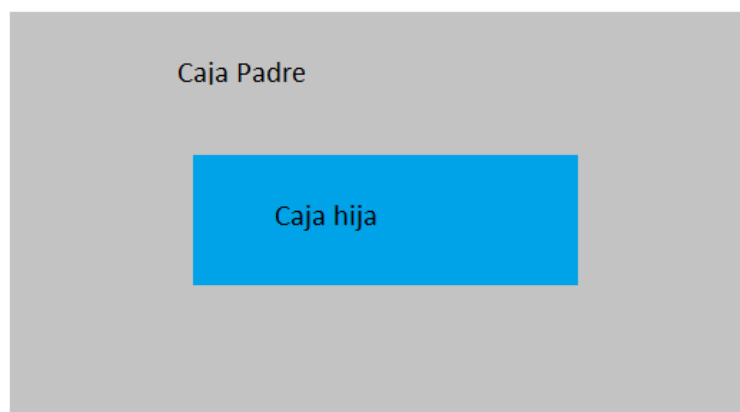
El **margin** es transparente y se sitúa en el borde más externo de la caja, dejando espacio entre la caja y otras cajas del documento. El **border** rodea la caja misma. Un borde puede ser transparente, o puede ser de color y tener un patrón como una línea discontinua. El **padding** es el espacio entre el borde de la caja y su contenido. El relleno generalmente toma la misma forma como el color de fondo de la caja. El **content** es lo que se muestra en la caja, como por ejemplo texto e imágenes. Se utilizan las propiedades de borde, margen, relleno, altura y anchura de CSS para modificar las diferentes partes del modelo de la caja.

Un problema importante con el modelo CSS Box es que los diferentes navegadores Web aplican las propiedades CSS de forma diferente. Por ejemplo, aunque el W3C establece que las propiedades de altura y anchura definen la altura y anchura del contenido de una caja, las versiones anteriores de *Internet Explorer* aplican las mismas propiedades a la altura y a la anchura del borde, lo que incluye el relleno y el contenido.

**Nota:** los diseñadores de páginas web y aplicaciones web a menudo recurren al uso de hacks (valores de propiedad CSS personalizados) para forzar a Internet Explorer a usar etiquetas CSS que otros navegadores ignorarán. El concepto es similar a los prefijos de proveedor para propiedades CSS mencionados en la sección anterior.

Otros dos conceptos a entender sobre el modelo CSS Box son los elementos tipo *block-level* e inline. Un **elemento tipo block-level** crea cajas que contribuyen al layout del documento. Secciones, artículos, párrafos, listas e imágenes son ejemplos de elementos de nivel de bloque. Los **elementos tipo inline** están diseñados para presentar el texto y no interrumpen el flujo del documento. Aplicando negrita y la nueva marca *HTML5* son ejemplos de elementos de tipo inline.

Por último, es importante entender la relación padre/hijo cuando se trabaja con el CSS. Esencialmente, una caja de padre puede contener una o más cajas. Las cajas contenidas dentro una caja de padre se conocen como cajas hijas. La siguiente imagen muestra un ejemplo simple de una caja de padre con una caja hija anidada. Una caja hija puede heredar estilos CSS de una caja padre, que significa que los estilos aplicados a una caja padre también se aplican a una caja hija. No todas las propiedades CSS son heredables; al aplicar estilos, es necesario comprobar la especificación CSS para determinar herencia de la propiedad.



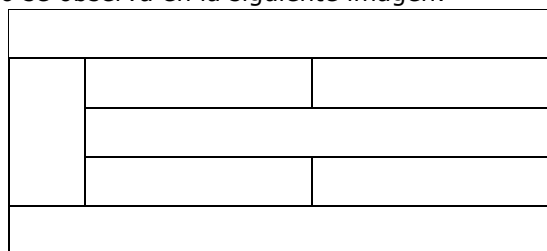
Lo que a CSS le ha faltado desde su inicio es una manera fácil de organizar los elementos horizontal y verticalmente en un documento *HTML*, para usar CSS y controlar el layout que se muestra bien en varios navegadores y cuando se ve desde diferentes tamaños de pantalla. Los modelos CSS3 Flexbox Box y Grid Layout abordan esta preocupación.

El modelo CSS **Flexbox Box** es un modo de layout para utilizar cajas flexibles en interfaces de usuario. El modelo forma parte del borrador de la especificación CSS3. Un **flexbox** ofrece layouts flexibles para el diseño de la interfaz de usuario. Se pueden crear páginas Web y aplicaciones para móviles con elementos, controles, barras de herramientas, menús y formularios que se redimensionan y reposicionan automáticamente cuando el usuario cambia el tamaño de la ventana del navegador. El navegador tiene en cuenta el espacio disponible y calcula las dimensiones para el usuario, lo que permite obtener tamaños y posiciones relativos.

Se puede colocar el contenido de una flexbox dinámicamente en cualquier dirección, ya sea a la izquierda, a la derecha, arriba o abajo. También se puede cambiar el orden de las cajas y flexionar sus tamaños y posiciones para llenar el espacio disponible. Una flexbox multilínea envuelve el contenido en varias líneas, de forma muy parecida a como un procesador de texto maneja el texto de un párrafo.

**Nota:** el diseño de Flexbox es similar al de los layout de bloques, pero Flexbox no utiliza columnas ni flotadores. Además, mientras que el layout de bloques suele distribuir el contenido verticalmente (y el layout en línea tiende a distribuir el contenido horizontalmente), una caja flexible cambia de tamaño en ambas direcciones.

Donde el modelo Flexbox Box es adecuado para cosas simples como botones, barras de herramientas y muchos formularios, se puede utilizar el modelo CSS **Grid Layout** para layouts más complejos. El modelo de diseño de cuadrícula permite controlar el diseño de secciones o documentos completos basados en *HTML* utilizando *CSS3*. Como su nombre indica, un layout de cuadrícula utiliza filas y columnas para hacer que el diseño parezca más limpio y estructurado, como se observa en la siguiente imagen:



Grid layout también ofrece modularidad, de modo que se pueden colocar fácilmente elementos en una cuadrícula o mover partes de una cuadrícula a un área diferente de un documento. Las cuadrículas son mucho más flexibles y fáciles de trabajar que con tablas de *HTML* o incluso columnas o flotadores para estructurar el layout.

## 5.2. Usando un Flexible Box para Establecer Contenido de Alineación, Dirección y Orientación

Se define un elemento como flexbox utilizando las propiedades de CSS **display: flexbox** o **display: inline-flexbox**, que se describen a continuación:

- **Flexbox:** un conjunto de flexbox como un elemento block-level.
- **Inline-flexbox:** un conjunto de **flexbox** como un elemento inline-level.

Una caja dentro de una caja es una caja hija, que puede ser flexible o no. Una caja hija se conoce como un **flexbox item**.

Flexbox también presenta otras nueve propiedades, como se indica en la siguiente tabla.

Propiedad	Valor	Descripción
<b>flex</b>	<b>pos-flex</b> <b>neg-flex</b> <b>preferred-size</b> <b>none</b>	Hace flexible las cajas hijas en altura y anchura

<b>flex-align</b>	<b>start</b> <b>end</b> <b>center</b> <b>baseline</b> <b>stretch</b>	Establece la alineación predeterminada para las cajas hijas; si la orientación de la caja padre es horizontal, flex-align determina la alineación vertical de las cajas hijas y viceversa
<b>flex-direction</b>	<b>row</b> <b>row-reverse</b> <b>column</b> <b>column-reverse</b>	Controla la dirección de las cajas hijas en la caja padre; también afecta a la propiedad flex-pack
<b>flex-flow</b>	<b>flex-direction</b> <b>flex-wrap</b>	Establece las propiedades flex-direction y flex-wrap simultáneamente
<b>flex-item-align</b>	<b>auto</b> <b>start</b> <b>end</b> <b>center</b> <b>baseline</b> <b>stretch</b>	Sobrescribe el alineamiento por defecto del estilo de las cajas hijas con la propiedad flex-align
<b>flex-line-pack</b>	<b>start</b> <b>end</b> <b>center</b> <b>justify</b> <b>distribute</b> <b>stretch</b>	Establece el alineamiento de la caja hija dentro de la caja padre cuando existe espacio extra
<b>flex-order</b>	<b>number</b>	Asigna cajas hijas a grupos y controla el orden en el que aparecen en el layout, comenzando con el grupo con la numeración más baja
<b>flex-pack</b>	<b>start</b> <b>end</b> <b>center</b>	Justifica el alineamiento de cajas hija dentro de un flexbox para asegurar que todo el espacio blanco en la caja padre está relleno
<b>flex-wrap</b>	<b>wrap</b> <b>nowrap</b> <b>wrap-reverse</b>	Determina si las cajas hija crean automáticamente una línea y se envuelve en ella o sobresale del flexbox

### 5.2.1. Trabajando con Flexboxes y Flexbox Items

Se supone que una empresa proporciona tres tipos principales de servicios, que se muestran y se describen brevemente en tres párrafos de una página web. Los tres párrafos forman tres contenedores de información, como se muestra en la siguiente imagen.





Existe un espacio extra a la derecha de la última caja de hijos, etiquetada como *Child 3*. Se puede modificar el CSS que controla las cajas para que las tres cajas hijas se expandan automáticamente en tamaño de manera uniforme para llenar el espacio disponible en la caja flexible. También se puede modificar una caja secundaria, como *Child 3*, para que sea flexible y llene el espacio, como se muestra en la siguiente imagen.



#### a. Aplicación de Escala Proporcional Dentro de una Caja Flexible

El W3C especifica la propiedad **flex**, que controla la altura y el ancho de los elementos de la flexbox. Mientras que la propiedad **display: flexbox** crea una caja padre flexible, la propiedad **flex** es la que da la naturaleza flexible a las cajas hijas.

La propiedad **display: flexbox** se utiliza sin valores adicionales.

La propiedad **flex** puede tener un valor de flexión positivo y/o negativo, un tamaño preferido y la palabra clave **none**, como se muestra:

<b>flex: pos-flex neg-flex preferred-size none</b>
--

Los valores de flexión positivos y negativos indican flexibilidad. Contrariamente al uso de la palabra "negativo" ambos son en realidad números positivos, como 1, 2, 3, y así sucesivamente. (También se puede usar 1.0, 2.0, 3.0, etc.)

Si queda espacio en la flexbox cuando aumenta el tamaño de la pantalla, las posiciones de la flexbox se expanden hasta rellenar el espacio en función del valor positivo de flexión. Un valor de 1 significa que cada artículo de la caja flexible ocupa una parte igual del espacio disponible, un valor de 2 significa que cada artículo ocupará dos partes iguales, y así sucesivamente. Si los elementos de la flexbox se desbordan de la caja padre porque son colectivamente más anchos que el padre, el navegador utiliza el valor de flexión negativa para determinar la altura o el ancho de cada artículo.

Si no especifica un valor de flexión positivo, el valor predeterminado es 1. La omisión de un valor negativo de flexión, por defecto es 0.

El valor de tamaño preferido puede ser cualquier valor que sea válido para la propiedad CSS **height** and **width**, como 100px. Si no especifica un valor de tamaño preferido, el valor predeterminado es 0px. También puede establecer el valor de tamaño preferido en automático, que utiliza el valor de la propiedad **width** o **height** como tamaño preferido.

La palabra clave **none** es equivalente a **0 0 auto**.

**Nota:** el valor de la propiedad **flex** puede necesitar alguna explicación adicional. Digamos que se tiene un flexbox con tres cajas para hijos. El valor de flexión para child1 e child2 es 1 y el valor para el child3 es 2. Un hijo con una flexión de 2 es el doble de flexible que un hijo con una flexión de 1. Esto no significa necesariamente que child3 tendrá el doble de ancho que child1 y child2. El valor de flexión es un cálculo basado en el espacio disponible para estirar o encoger; el cambio se asigna en función de la parte de flexibilidad en comparación con la otra parte cajas para hijos.

El poder de los artículos de la flexbox es que pueden escalar libremente o ajustar dinámicamente su tamaño principal. Los artículos aumentan o disminuyen de tamaño según el espacio disponible en la flexbox en la que residen.

En el siguiente código **CSS** y marcado **HTML**, el flexbox contiene cuatro elementos de flexbox. Cada hijo tiene un valor de flexión de **1** y está configurado en **auto**. Cuando el usuario cambia el tamaño de la ventana del navegador, los buzones secundarios deben expandirse y contraerse junto con el buzón principal.

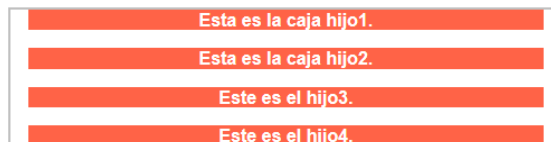
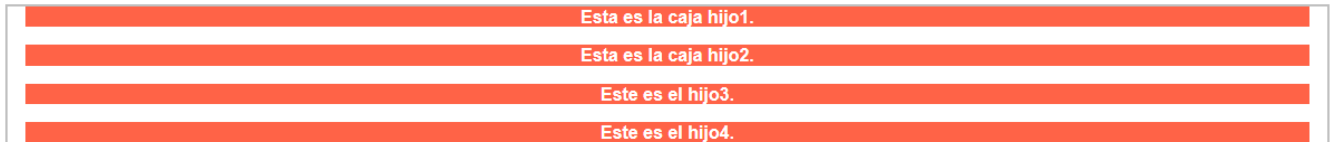
```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Flexible Child Box Example</title>
  <style>
    div { display: flexbox;
          outline: 2px solid silver
        }
    p { flex: 1 auto; margin: 1em;
        font-family: sans-serif;
        color: white;
        background: tomato;
        font-weight: bold;
        text-align: center;
      }
  </style>
</head>
<body>
  <div>
    <p>Esta es la caja hijo1.</p>
    <p> Esta es la caja hijo2.</p>
    <p> Este es el hijo3.</p>
```



```

    <p>Este es el hijo4.</p>
  </div>
</body>
</html>

```



Alternativamente, se puede utilizar la función CSS **flex** con la función CSS **height** o **width** para controlar la altura y el ancho de los artículos de la flexbox. La propiedad **flex** y la función **flex** se comporta de la misma manera, pero usa una sintaxis ligeramente diferente, una función incluye valores entre paréntesis.

La propiedad **flex-wrap** determina si las cajas secundarias crean automáticamente una nueva línea y la envuelva en ella. La propiedad **flex-wrap** utiliza la propiedad valores de **nowrap**, **wrap** y **wrap-reverse**.

Como se verá, el código CSS utiliza prefijos de proveedores (**-ms-**, **-moz-**, **-o-**, y **-webkit-**), que son necesarios para que la propiedad **flex-wrap** funcione. Los prefijos de los proveedores se utilizan frecuentemente durante la transición a CSS3 para hacer el código compatible con tantos navegadores como sea posible.

Otras propiedades que se puede usar a menudo en flexboxes son:

- **flex-pack**: Justifica la alineación de las cajas secundarias dentro de una **flexbox** y minimiza los espacios en blanco en la caja principal. Esta propiedad acepta uno de los cuatro valores: **start**, **end**, **justify** o **center**.
- **flex-align**: Establece la alineación por defecto para las cajas hijas, pero con un giro. Si la orientación de la caja principal es horizontal, la alineación flexible determina la alineación vertical de las cajas secundarias, y viceversa.

Después de que las cajas hijas de una flexbox hayan terminado de flexionarse y si todavía hay espacio disponible en la flexbox padre, las cajas hijas pueden alinearse con las propiedades **flex-pack** y **flex-align** (o **flex-item-align**). Lo más importante que hay que recordar es que se aplica la propiedad **flex-pack** a la flexbox padre en el código CSS, y se aplica **flex-align** a las cajas hijas.

#### b. Cambiando la Dirección de los Artículos Inferiores en una Flexbox

La propiedad **flex-direction** afecta a la dirección de las cajas secundarias en la caja principal. Utiliza los valores **row**, **row-reverse**, **column** y **column-reverse**.

La propiedad **flex-flow** establece las propiedades **flex-direction** y **flex-wrap** al mismo tiempo. El siguiente ejemplo utiliza la propiedad **flex-flow** con el valor de columna.

```

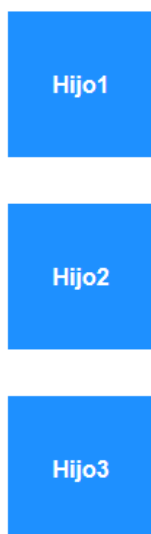
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Ejemplo Flex-flow</title>
  <style>
    div {

```



```
display: flex;
display: -ms-flex;
display: -moz-flex;
display: -o-flex;
display: -webkit-flex;
flex-flow: column;
-ms-flex-flow: column;
-moz-flex-flow: column;
-o-flex-flow: column;
-webkit-flex-flow: column;
height: 400px;
padding: 1em;
outline: 2px solid silver;
color: white;
font-family: sans-serif;
font-weight: bold;
}
p {
width: 100px;
margin: 1em;
height: 100px;
background-color: dodgerblue;
text-align: center;
line-height: 100px;
}
</style>
</head>
<body>
<div>
<p>Hijo1</p>
<p>Hijo2</p>
<p>Hijo3</p>
</div>
</body>
</html>
```

El resultado de renderizar este código y el marcado en el navegador Web se muestra en la siguiente imagen:

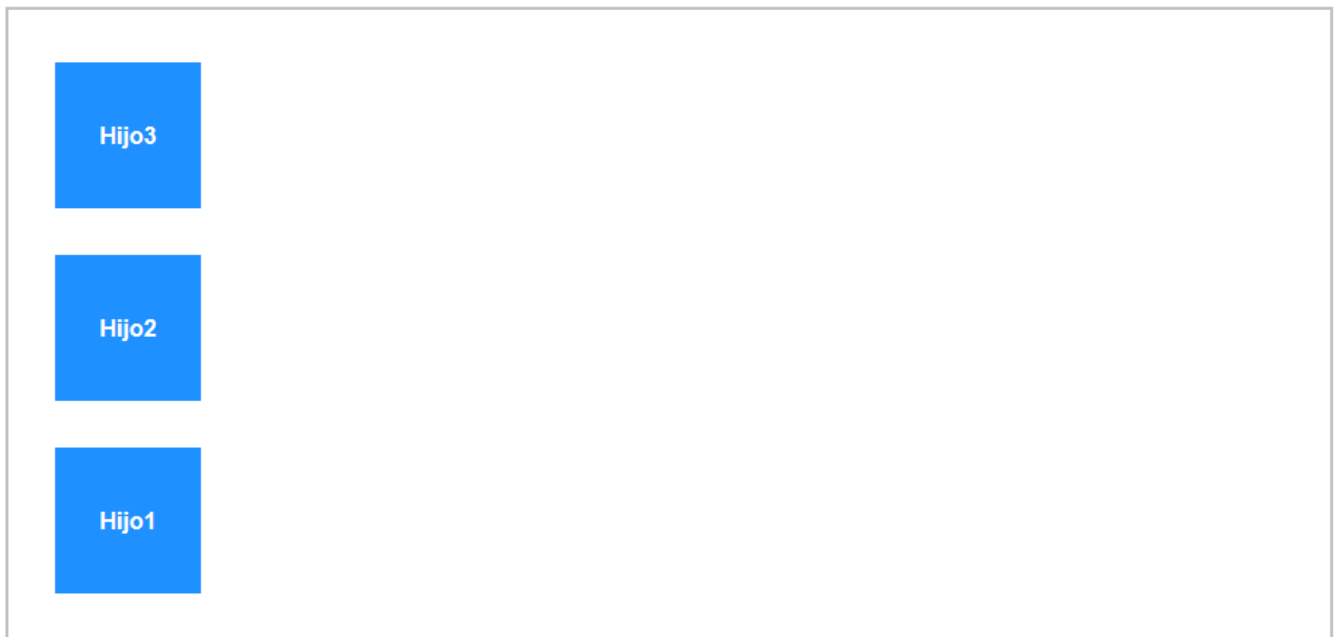




Para invertir el orden de las cajas hijas, se cambia cada uno de los valores de la columna **flex-flow** a **column-reverse**, de la siguiente manera:

```
flex-flow: column-reverse;  
-ms-flex-flow: column-reverse;  
-moz-flex-flow: column-reverse;  
-o-flex-flow: column-reverse;  
-webkit-flex-flow: column-reverse;
```

Observando los resultados en la siguiente imagen. Se observan los efectos de inversión de valores:



### c. Ordenando y Arreglando Contenido

Se puede controlar el orden y la disposición del contenido de una **flexbox** utilizando la propiedad **flex-order**. Esta propiedad reorganiza los artículos inferiores dentro de una **flexbox**. Para ello, la propiedad asigna casillas secundarias a grupos y, a continuación, controla el orden en el que aparecen en un layout, empezando por el grupo con el menor número.

A continuación, se muestra cómo funciona la propiedad **flex-order**. El siguiente código CSS y marcado crea tres cajas hijas en una caja flexible:

```
<!doctype html>  
<html>  
<head>  
  <meta charset="utf-8">  
  <title>Ejemplo de orden flexible</title>  
  <style media="screen">  
    div {  
      display: flex;  
      display: -ms- flex;  
      display: -moz- flex;
```

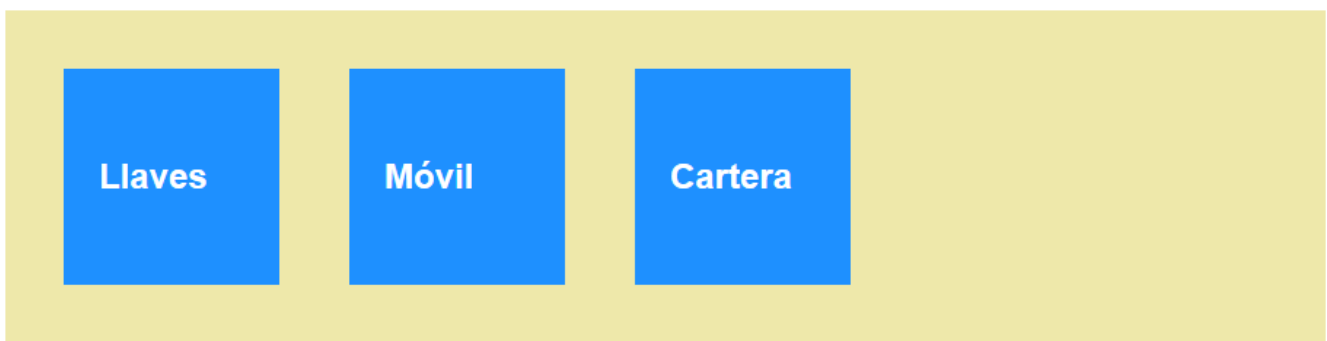
```

display: -o- flex;
display: -webkit- flex;
flex-flow: row;
-ms-flex-flow: row;
-moz-flex-flow: row;
-o-flex-flow: row;
-webkit-flex-flow: row;
height: 200px;
padding: 1em;
background-color: palegoldenrod;
font: bold 100%/1 sans-serif;

div>div {
width: 100px;
margin: 1em;
height: 100px;
background-color: dodgerblue;
text-align: center;
color: white;
font-size: x-large;
line-height: 100px;
}
</style>
</head>
<body>
<div>
<div>Llaves</div>
<div>Móvil</div>
<div>Cartera</div>
</div>
</body>
</html>

```

En el navegador web, se mostraría de la siguiente manera:



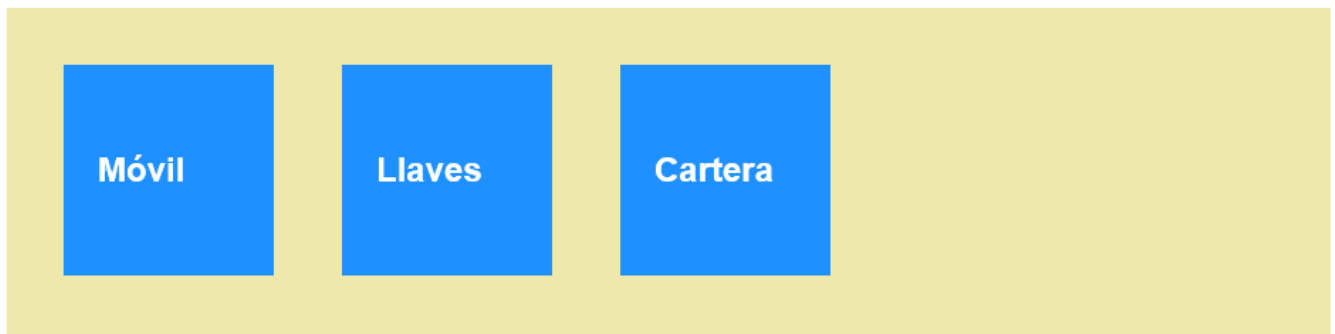
**Nota:** El elemento de estilo *HTML* anterior incluye el atributo `media=screen`, que es una consulta de medios. *Media queries* permiten adaptar un documento *HTML* a los dispositivos del usuario final. Los tipos de elementos de medios *HTML* incluyen aural, braille, portátil, impreso, proyección, pantalla, tty y tv. La misma sintaxis también se puede utilizar con las reglas `@media` y `@import` CSS. La regla `@media all` indica que el CSS debe aplicarse a todos los medios de salida.

La propiedad **flex-order** coloca las cajas hijas en grupos ordenados. El grupo por defecto es 0. Se declaran grupos y se les asigna un número en CSS utilizando la propiedad **flex-order**, y cualquier ítem hijo no asignado



explícitamente a un grupo permanece en el grupo 0, y los grupos declarados aparecen antes del grupo 0. Por lo tanto, para reordenar las cajas hijas de modo que las cajas *Llaves* y las *Cartera* aparezcan antes de la caja *Móvil*, añadir este código en la parte inferior de la sección de estilo:

```
div>div:first-child,  
div>div:last-child {  
    flex-order: 1;  
    -ms-flex-order: 1;  
    -moz-flex-order: 1;  
    -o-flex-order: 1;  
    -webkit-flex-order: 1;  
}
```



**Nota:** este ejemplo solo es aplicable con *Internet Explorer*.

### 5.3. Usando Grid Layouts para Establecer Contenido de Alineación, Dirección y Orientación

Los layouts en cuadrícula son más adecuados para layouts más complejos de los que **flexbox** suele manejar. Las interfaces de los juegos son buenos candidatos para las cuadrículas, al igual que los layouts de los periódicos. Un layout en cuadrícula permite posicionar filas y columnas para un control preciso sobre el layout, con una fracción del código CSS tal y como se requeriría utilizando técnicas más antiguas. Otra ventaja de las cuadrículas o rejillas es su modularidad. Para mover bloques de contenido de una parte de una página o de una aplicación a otra, simplemente debe mover algunas líneas de código en CSS.

Se define un elemento de cuadrícula (layout) utilizando el comando **display:grid** o **display:inline-grid** CSS de la propiedad. Esto crea el container para el layout.

Los elementos inferiores de una cuadrícula se denominan **grid items**, de acuerdo con los cuales se posicionan y dimensionan:

- **Grid tracks:** Las columnas y filas de la cuadrícula; las grid tracks se definen usando las propiedades **grid-rows** y **grid-columns**.
- **Grid lines:** Las líneas horizontales y verticales que dividen columnas o filas.
- **Grid cells:** El espacio lógico utilizado para organizar los elementos de la cuadrícula es similar a una celda de una hoja de cálculo.

Las propiedades de la cuadrícula y sus valores se enumeran en la siguiente tabla:

Propiedad	Valor	Descripción
Grid-columns o Grid-rows	length percentage fraction max-content min-content minmax (min, max)  auto	Especifica los parámetros para una o más columnas o filas de una cuadrícula.
grid-template	string + none	Proporciona una visualización de la estructura del elemento de la rejilla y define las celdas de la rejilla.
grid-cell	string none	Posiciona un elemento hijo dentro de una celda de cuadrícula con nombre
grid-column o grid-row	flex- direction  flex-wrap	Coloca los elementos inferiores en una cuadrícula
grid-column-span o grid-row-span	integer	Define las dimensiones de una celda de cuadrícula especificando la distancia (en líneas) desde la línea de inicio hasta la línea final.
grid-column-sizing o grid-row-sizing	track-minmax	Cambia el tamaño de las columnas o filas implícitas, que son de tamaño automático por defecto.
grid-flow	none rows  columns	Crea columnas o filas adicionales según sea necesario para acomodar el contenido
grid-column-align o grid-row-align	start end center  stretch	Controla la alineación de un elemento inferior dentro de una celda.

Con todas las propiedades disponibles, se puede adivinar que se puede especificar la estructura del elemento de la cuadrícula, la posición y el tamaño de los elementos **grid** de múltiples maneras.

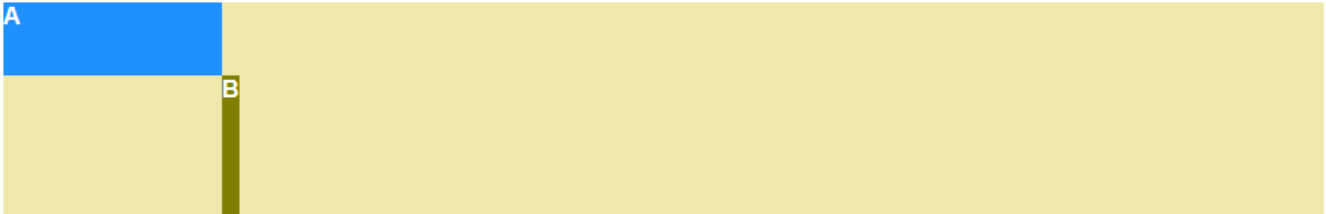
### 5.3.1. Creando una cuadrícula usando propiedades CSS para filas y columnas

Las propiedades principales que crean una cuadrícula son **display:grid** (o **display:inline-grid**), **grid-columns** y **grid-rows**.

Se pueden definir columnas y filas para que tengan un tamaño fijo, que no se redimensionen cuando la pantalla cambie de tamaño o cambie una fracción de la cuadrícula. Los tamaños de las cuadrículas se definen usando **fr**, así que una definida como **2fr** será el doble del tamaño de una fila definida como **1fr**. El valor **1fr** se define como "una fracción". También se puede utilizar el valor "**auto**" para hacer columnas o filas que se ajusten a su contenido.



El siguiente código *CSS* y el marcado *HTML* proporcionan un ejemplo de un layout de cuadrícula. El prefijo *ms* de proveedor se incluye al principio de todas las construcciones relacionadas con la red porque sólo *Microsoft Internet Explorer 10* soporta diseños de cuadrícula. Esto se muestra en la siguiente imagen.



```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Ejemplo de cuadrícula</title>
  <style type="text/css">
    #grid {
      background: palegoldenrod;
      border: silver;
      display: -ms-grid;
      color: white;
      font-family: sans-serif;
      font-weight: bold;
      -ms-grid-columns: 150px auto 2fr;
      -ms-grid-rows: 50px 6em auto;
    }
    #A {
      background: dodgerblue;
      -ms-grid-row: 1;
      -ms-grid-column: 1;
    }
    #B {
      background: olive;
      -ms-grid-row: 2;
      -ms-grid-column: 2;
    }
  </style>
</head>
<body>
  <div>
    <div id="grid">
      <div id="A">A</div>
      <div id="B">B</div>
    </div>
  </div>
</body>
</html>
```

En este ejemplo, la estructura de cuadrícula tiene tres columnas y tres filas. La primera columna tiene una anchura fija de 150 píxeles. La segunda columna ajusta su ancho al contenido de la columna, como lo indica la palabra clave *auto*. La tercera columna es dos unidades de fracción del espacio restante en la cuadrícula. La primera fila tiene 50 píxeles de alto, la segunda fila tiene 6 ems de alto, y la tercera fila se ajusta para ajustarse al contenido de la fila.

Para añadir otro elemento a la cuadrícula de forma que aparezca una tercera celda y columna, se inserta el siguiente código al final de la sección style:

```
#C {
  background: orange;
  -ms-grid-row: 3;
  -ms-grid-column: 3;
}
```

Y se añade el siguiente código después del ítem div en la sección body:

```
<div id="C">
  C
</div>
```

El resultado final es el que se muestra en la siguiente figura:



### 5.3.2. Entendiendo Grid Templates

El Grid Template Layout Module de CSS del W3C presenta otro enfoque a los layouts de cuadrícula mediante la creación de una **grid template**, que es como una tabla vacía en la que pueden fluir los datos. Una cuadrícula utiliza caracteres alfabéticos para representar la posición de los elementos de una cuadrícula.

Se utiliza la propiedad de **grid-position** y se asigna un carácter alfabético como valor de posición. Los siguientes ejemplos muestran la propiedad de **grid-position** definida para cuatro elementos:

```
news { grid-position: a; }
weather { grid-position: b; }
sports { grid-position: c; }
events { grid-position: d; }
```

Después de asignar posiciones, se crea un layout utilizando strings de caracteres. Una cadena es igual a una fila, y cada carácter en la cadena es una columna. Por ejemplo, para crear una cuadrícula con una fila con cuatro columnas que se ajusten al contenido, se usará la siguiente sintaxis:

```
div { grid-template: "abcd"; grid-rows: auto;  
grid-columns: auto;}
```

Aunque este ejemplo usó la palabra clave **auto**, se puede usar cualquiera de los valores para las filas de cuadrícula y las columnas de cuadrícula que se muestran en la tabla anterior.

La especificación para los layouts de plantillas de cuadrícula está en formato de borrador y no está soportada por ningún navegador web en el momento de escribir este documento. Sin embargo, es posible encontrar plantillas de cuadrícula en el examen del *MTA 98-375*. Por lo tanto, se debe comprobar la última especificación del módulo de diseño de plantillas de cuadrícula *CSS* del *W3C* cuando se prepare para realizar el examen.



## 6. Manejando Texto Flotante usando CSS

---

### 6.1. Manejando el Flotamiento del Contenido de Texto usando CSS

El complejo layouts de páginas para publicaciones impresas como revistas, periódicos y libros ha sido afinado por fabricantes de software como *Adobe* y *Microsoft*. *Adobe InDesign*, *Microsoft Publisher*, e incluso *Microsoft Word 2010* manejan el flujo de contenido entre las columnas de manera eficiente. *InDesign* y *Publisher* en particular son expertos en fluir el contenido entre las áreas de un documento que no son contiguos (que se tocan), como los separados por imágenes, cajas de contenido o páginas. El software de autoedición facilita la conexión de contenido en diferentes áreas, de modo que los cambios hecho a un área permite que el contenido fluya correctamente a otras áreas conectadas.

En los documentos *HTML*, el flujo de contenido ha sido un reto para los diseñadores Web y de aplicaciones durante años. La visualización de un layout complejo en *HTML* requiere el mismo posicionamiento flexible de cajas como programas de edición electrónica, pero las herramientas para lograr este tipo de flujo de contenido sólo han estado disponibles recientemente.

*Microsoft* y *Adobe* colaboraron con el W3C para crear el concepto de 'regions CSS' para flujo de contenido basado en la web. CSS Regions permite a los desarrolladores transmitir el contenido de forma dinámica a través de múltiples cajas, o contenedores, en documentos *HTML* con diseños fluidos. El contenido se ajusta y muestra correctamente si un usuario visualiza el documento en un monitor de ordenador grande o en una pequeña pantalla de tabletas.

Aunque CSS3 permite utilizar la disposición de varias columnas para separar el contenido en columnas, ofrecen un mejor control del flujo de contenido en diseños más complejos. Combina CSS Regions con técnicas de layouts CSS, como columnas, flexboxes y layouts de cuadrícula.

La separación por guiones, que rompe las palabras entre sílabas al final de las líneas, también es importante para los diseños fluidos, ya que permite una justificación completa de los párrafos independientes y de los que se encuentran en columnas. Los flotadores posicionados, ahora llamados Exclusiones CSS, le permiten envolver completamente el texto alrededor de imágenes, formas y contenedores de texto.

Este capítulo cubre las regions de CSS, las múltiples columnas de CSS3, la separación por guiones de CSS3 y los flotadores posicionados para ayudarle a aprender a fluir y presentar contenido en los diseños *HTML* que se ajustan dinámicamente.

### 6.2. Entendiendo y usando Regions para Contenido de Texto entre Múltiples Secciones

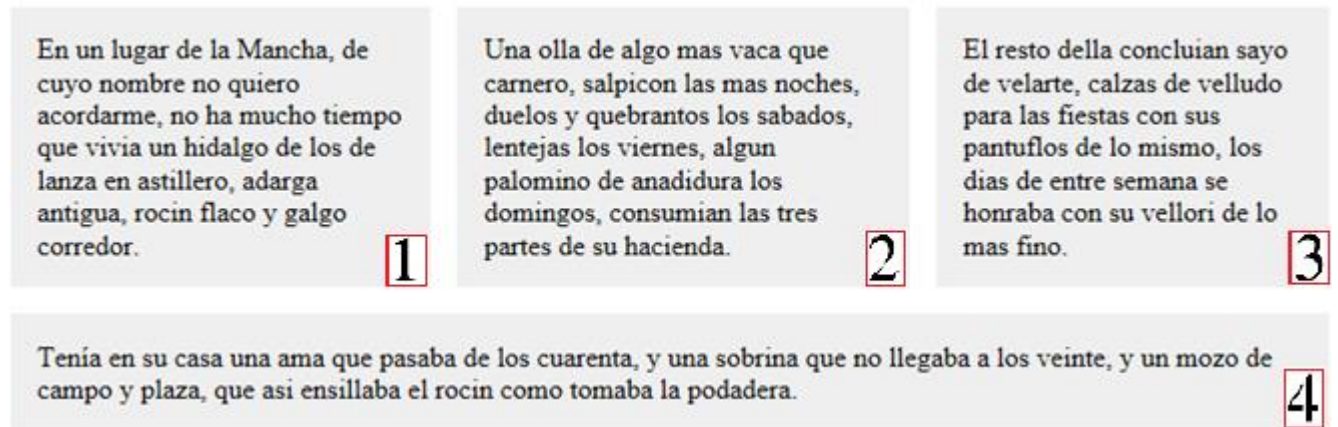
En un documento *HTML* típico, se puede mostrar el contenido en diferentes secciones o áreas, pero cada área es independiente. Si se desea que el texto desbordante se desplace de un área a otra, generalmente se tiene que hacer manualmente. Este enfoque no funciona bien cuando un usuario cambia el tamaño de la pantalla o utiliza las herramientas de accesibilidad, como una lupa. Este método tampoco se presta para el cambio automático de la orientación horizontal en tabletas y teléfonos inteligentes. Una solución es CSS Regions.

**CSS Regions** son áreas definidas (regiones) de un documento *HTML* donde el contenido puede fluir. Similar a un programa de diseño de página, cuando hay demasiado contenido para caber en una región, el programa el contenido restante fluye automáticamente a la siguiente región. (Véase la siguiente imagen) Si un usuario

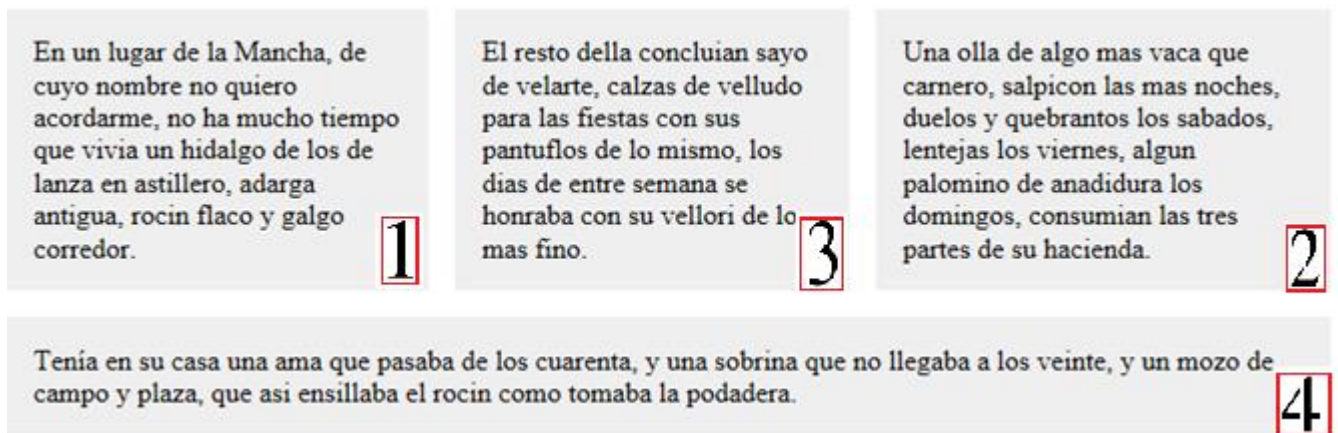




cambia de tamaño la pantalla en la que se visualiza el documento o el documento en una pantalla más pequeña o más grande. el contenido cambia de tamaño y vuelve a fluir automáticamente a través de las regions.



Las **regions** no tienen que estar una al lado de la otra dentro del documento, y se puede controlar el orden en el que aparece el contenido fluido. La siguiente figura muestra el flujo de contenido no contiguo entre regions. Este tipo de flujo se denomina a veces "story threading" y permite añadir comillas y barras laterales a un documento sin interrumpir el flujo del contenido.



### 6.2.1. Contenido Flotante a través de Contenedores Dinámicos

Una **content source** puede ser uno o más bloques de texto en el mismo documento *HTML* o en otro separado que contiene el contenido que se desea que fluya a través de un diseño. El contenido se denomina un "**content stream**".

También se necesitan **content container**, que son las áreas en las que fluye el contenido. El documento *HTML* con **content container** actúa como una página maestra, como una plantilla, en la que cada contenedor tiene el tamaño y la posición donde desea que aparezca el contenido, pero cada contenedor está inicialmente vacío.

Dentro de la fuente de contenido, el elemento que contiene el contenido a trasladar es asigna a la propiedad CSS `flow-into`. El valor de esta propiedad se llama flujo denominado.

El siguiente ejemplo comienza con algo de código CSS para una fuente de contenido. El valor de `flow-into` es "main". Es así porque el contenido realmente aparecerá en otro (en los contenedores de contenido), este elemento fuente en sí mismo no se mostrará en la ventana de diálogo página *HTML*.

```
<style>
  #source {
    flow-into: main;
  }
  .region {
    flow-from: main;
    background: #9ACD32;
  }
</style>
<body>
  <div id="source">
    <p>Lorem ipsum dolor ...</p>
  </div>
  <div id="region1" class="region"></div>
  <div id="region2" class="region"></div>
  <div id="region3" class="region"></div>
</body>
```

**Nota:** El W3C define las Regions CSS en la especificación de Regions CSS. Por lo tanto, al igual que con las flexbox y las rejillas como se describe en el capítulo 5, se debe utilizar prefijos de proveedores con nombres de propiedades. Se verá el uso del prefijo de proveedor en el siguiente ejercicio paso a paso.

Continuando en la misma muestra, la propiedad `flow-from` CSS crea el contenedor de contenido, que es una Region CSS. El valor de la propiedad `flow-from` debe coincidir con el valor de flujo de la propiedad `flow-into`, así es como se asocian la fuente de contenido y el contenedor de contenido. (El ejemplo de esta sección incluye el código fuente del contenido en el mismo *HTML* como los contenedores de contenido para mayor comodidad. En una aplicación del mundo real, se usaría un documento *HTML* para fuentes y contenidos.)

Cuando un navegador muestra la página con los contenedores de contenido, el contenido fluye a los contenedores y se muestra en la pantalla como se muestra en la primera imagen. Si se cambia el tamaño de la pantalla.

También se puede tener varias fuentes y asignar la propiedad de entrada de flujo a varios elementos. El contenido se extraerá de la fuente en el orden en que aparezca en el *Document Object Model (DOM)*. Ésto se conoce como el orden de los documentos.

**Nota:** el *DOM* fue introducido en el capítulo 1, pero debe ser revisado aquí. El *DOM* es una especificación W3C que describe la estructura del *HTML* dinámico y un documento Extensible Markup Language (*XML*) de una manera que permite que un navegador Web manipule. El *DOM* permite que los programas y scripts actualicen el contenido, la estructura y los estilos sobre la marcha, cualquier cosa en un archivo *HTML* o *XML* puede ser modificado.



### a. Texto Desbordante

Para que las Regions CSS funcionen, el flujo de contenido no puede afectar a la altura de una región, se necesitan definir alturas de región en CSS para que no sean flexibles. Una región recibe tanto contenido como pueda contener y luego fluye el contenido restante a la siguiente región.

Si todavía queda contenido después de llenar todas las regions, puede ocurrir una de estas tres situaciones. El contenido desbordado en la última región estará:

- Truncado
- Continúa desbordado y visible
- Continúa desbordando, pero oculto

Se puede controlar cómo la última región maneja el contenido desbordado utilizando la función **región-overflow** y las propiedades **overflow**.

**región-overflow** está configurado en *auto* o en *break*. Utilizando el valor automático, se puede especificar la propiedad **overflow** como visible u oculta. Por ejemplo, si se desea que el contenido de desbordamiento continúe para que fluya y sea visible, se usaría la siguiente sintaxis:

```
.region {  
    region-overflow:auto;  
    overflow: visible;  
}
```

El uso del valor de ruptura para **región-overflow** evitará que el contenido desborde la última región, truncando el contenido en ese punto. La sintaxis es:

```
.region {  
    region-overflow:break;  
}
```

### b. Implementación de Microsoft de Regiones CSS

El método de *Microsoft* para implementar CSS Regions varía un poco de la versión W3C descrita anteriormente. *Microsoft* utiliza **iframes**, que son como minicajas en una página web que contienen contenido externo incrustado en un documento *HTML*, como fuente de contenido. También se debe utilizar el prefijo de proveedor **-ms-** con las propiedades **flow-into** y **flow-from**.

Por ejemplo, a continuación, se muestra un elemento *iframe* con un ID único, que se debería añadir a la pagina máster:

```
<iframe id="main-data-source" src="source.html" />
```

Despues se usaria el selector CSS que especifica la fuente de datos:



```
#main-data-source { -ms-flow-into: main; }
```

Para crear almacenadores de contenido, se asigna un nombre de clase a los elementos que se desee usar como contenedores:

```
<div class="region"></div>  
<div class="region"></div>
```

**Nota:** El uso de *region* en ambos casos no es un error. Al igual que con cualquier regla CSS, se puede identificar regiones usando un nombre de clase compartido (como en este ejemplo) o listarlas usando IDs individuales (como en el primer ejemplo).

A continuación, se crea un selector de CSS que especifique la fuente de datos desde la que aceptar el flujo de contenido:

```
.region { -ms-flow-from: main; }
```

Si se compara este código específico de *Microsoft* y el marcado con el ejemplo general mostrado anteriormente, se debería ser capaz de ver las similitudes con bastante facilidad.

Hay algunas cosas más que debe tener en cuenta con respecto a la versión de *Microsoft* de CSS Regions:

- **msRegionUpdate:** Permite manipular regions dinámicamente.
- **msRegionOverflow:** Maneja el desbordamiento de contenido, similar a la propiedad **region-overflow**.
- **msGetRegionContent:** Un método de script definido por *Microsoft* devuelto como "una matriz de las instancias Range correspondientes al contenido del flujo de región que se posiciona en la región".

### 6.2.2. Usando columnas y unión con guión para optimizar la lectura del texto

Otras características que son nuevas en CSS3 son los layout multicolumna y la unión con guión (hyphenation). Ahora se pueden crear varias columnas (estilo de periódico) en documentos *HTML* que se escalan según el tamaño de la pantalla del usuario. La unión con guión (hyphenation) rompe las palabras entre sílabas al final de las líneas para crear un margen derecho más uniforme y eliminar los espacios en blanco dentro de los párrafos.

#### a. Creando columnas

Las propiedades CSS3 para **multi-column layout** permiten crear columnas dividiendo el texto a través de múltiples columnas, se debe especificar la cantidad de espacio que aparece entre las columnas (el hueco), se debe hacer que aparezcan líneas verticales (reglas) entre las columnas, y definir donde se rompen las columnas.

Las propiedades principales CSS que se utilizan para crear y manipular múltiples columnas en un documento *HTML* son:



- **Column-count:** Establece el número de columnas en las que se debe dividir un elemento; se puede usar también la propiedad **columns** con valores para establecer las propiedades **Column-count** y el **Column-width** simultáneamente
- **Column-gap:** Especifica el espacio entre las columnas, que también se conoce como el canalón o separación.
- **Column-rule:** Crea una línea vertical en el espacio entre columnas y establece el ancho, estilo (línea simple o doble, sólido, punteado, 3D, etc.) y color de la regla

El layout **multi-column** CSS3 utiliza el concepto de "cuadro de columnas" para referirse al contenedor, que contiene el contenido y lo muestra en columnas. El cuadro de columna se encuentra entre el cuadro de contenido y el contenido en el *CSS Box Model* original. (Se puede consultar el capítulo 5 si se necesita una actualización de el modelo de la caja.)

La siguiente tabla muestra todas las propiedades de las columnas que funcionan en CSS3.

Propiedad	Valor	Descripción
<b>break-after</b>	auto always avoid left right page column region avoid-page avoid-column avoid-region	Inserta una separación después de la caja de columna generada
<b>break-before</b>	(same as break-after)	Inserta una separación antes de la caja de columna generada
<b>break-inside</b>	auto avoid avoid-page avoid-column avoid-region	Inserta una separación dentro de la caja de columna generada
<b>column-count</b>	integer auto	Establece el número de columnas que usará un elemento
<b>column-fill</b>	auto balance	Especifica cómo rellenar columnas; si es posible, equilibra el contenido de forma equitativa entre columnas o rellena las columnas de forma secuencial.
<b>column-gap</b>	length normal	Especifica el espacio entre columnas
<b>column-rule</b>	column-rule-width column-rule-style column-rule-color transparent	Es una propiedad abreviada que establece la propiedad ancho de la regla de la columna, estilo de la regla de la columna, y las propiedades del color de la regla de la columna en el mismo lugar en una hoja de estilo
<b>column-rule-color</b>	color	Especifica el color de la regla entre pilares



<b>column-rule- style</b>	<b>border-style</b>	Especifica el estilo de la regla entre columnas, como sólidas o dobles, punteadas, etcétera
<b>column-rule- width</b>	<b>border-width</b>	Especifica el ancho de la regla entre pilares
<b>column-span</b>	<b>none all</b>	Especifica si un elemento debe abarcar todas las columnas o ninguna
<b>column-width</b>	<b>length auto</b>	Especifica el ancho de una columna o columnas
<b>columns</b>	<b>column-width column-count</b>	Fija el column-width y el column-count propiedades simultáneamente

Ahora se verá cómo funciona el código CSS. Este código utiliza la propiedad **column-count** para crear tres columnas utilizando el texto en la marca *HTML* que siguiente:

```
<head>
  <style>
    .tricolumn { column-count: 3; }
  </style>
</head>
<body>
  <h2>Tres Columnas</h2>
  <div class="tricolumn">
    En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que viví un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.
  </div>
</body>
```

Debido a que las columnas CSS3 están todavía en fase de desarrollo, es posible que se necesite agregar prefijos de proveedores a los nombres de propiedades relacionadas con las columnas. En este caso, se modifica el código de la siguiente manera para los cuatro navegadores principales:

```
<style>
  .tricolumn {
    -ms-column-count: 3;
    -moz-column-count: 3;
    -o-column-count: 3;
    -webkit-column-count: 3;
  }
</style>
```

Tal como se ve en la siguiente imagen, el navegador mostraría lo siguiente:



## Tres Columnas

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de

algo más vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda.

El resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.

Se puede realizar múltiples columnas utilizando la propiedad `columns`, que es una forma abreviada de fijar el número de columnas y el ancho de columna en una declaración. El siguiente código utiliza el valor `auto` para el ancho de columna:

```
.tricolumn { columns: 3 auto; }
```

Otra forma de usar la propiedad de las columnas es asignar un valor numérico a `column-width` y dejar el `column-count` en `auto`. Este ejemplo establece `column-width` en `15em`, lo que significa que el elemento multicolumna tendrá un ancho de columna de `15 ems` (o 15 veces el ancho del tamaño de fuente del contenido de la columna):

```
columns: auto 15em;
```

Ahora se va a añadir una regla de columna. Esta propiedad establece el ancho, estilo y color de la regla entre todas las columnas. La sintaxis de una línea azul discontinua de 3 píxeles de ancho es:

```
column-rule: 3px dashed blue;
```

Para controlar el tamaño del espaciado entre columnas, se utiliza la propiedad `column-gap`. Esta propiedad utiliza un valor entero, como se muestra a continuación, o la palabra clave `normal`.

```
column-gap: 3em;
```

El efecto combinado de tres columnas con una regla de columna y una separación `3em` se muestra en la primera imagen a continuación, (con prefijos de proveedor aplicados). Redimensionar la ventana del navegador también redimensiona las columnas y redistribuye el contenido entre ellos, como se muestra en la segunda imagen que se muestra a continuación.





## Tres Columnas

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más

vaca que carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El resto della

concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.

## Tres Columnas

En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocín flaco y galgo corredor. Una olla de algo más

carnero, salpicón las más noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino de añadidura los domingos, consumían las tres partes de su hacienda. El

resto della concluían sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo mismo, los días de entre semana se honraba con su vellori de lo más fino.

Para especificar si un elemento como un encabezado debe abarcar las columnas, utilice la propiedad **column-span**. Esta propiedad se establece en un número de columnas para abarcar o utiliza las palabras clave "todo o nada", lo que significa que un elemento abarca todas las columnas o no abarca ninguna columna, respectivamente.

```
column-span: all;
```

Como se puede ver, las propiedades de columnas relacionadas proporcionan una visualización flexible del contenido con un mínimo de declaraciones CSS.

### b. Usando Separación por Guiones (Hyphenation)

La separación por guiones (**hyphenation**) es el proceso de conectar dos palabras con un guión (-) o de romperlas entre sílabas al final de una línea. La separación por guiones es muy útil en la disposición multi-columna para eliminar los espacios en blanco dentro de las columnas dejados por palabras largas que se ajustan automáticamente a la siguiente línea, haciendo que el texto parezca más profesional. Generalmente, la separación por guiones automática intenta justificar el texto con el margen derecho.

CSS3 introduce la propiedad **hyphens**, que controla la separación por guiones. La propiedad utiliza el **none**, **manual** y **auto**:

- **auto**: Permite la separación de palabras por guiones automática basándose en las oportunidades para saltar de línea dentro de las palabras o mediante un "recurso de separación por guiones apropiado para el idioma".
- **manual**: Permite la separación por guiones de palabras basándose en las oportunidades para saltar de línea dentro de las palabras
- **none**: Previene la separación por guiones

El W3C señala que se debe declarar un idioma utilizando los atributos **HTML lang** o **XML xml:lang** para que se produzca una correcta separación por guiones automática. Eso significa que si todo el documento **HTML** está





en el mismo idioma (inglés, por ejemplo) y se desea habilitar separación por guiones automática, se debe añadir el atributo al elemento *HTML* o a la declaración *doctype*, como:

```
<!doctype html>
<HTML lang="en-us">
```

O

```
<HTML xmlns="http://www.w3.org/1999/xhtml"
xml:lang="en" lang="en">
```

Se necesita incluir prefijos de proveedor con la propiedad de separación por guiones, como **-ms-hyphens** para *Internet Explorer 10*, **-moz-hyphens** para Firefox, etc.

Microsoft proporciona propiedades de separación por guiones adicionales que son específicas de los entornos de Microsoft, como las siguientes:

- **-ms-hyphenate-limit-zone**: Especifica el ancho del espacio en blanco al final (llamado zona de separación por guiones o hyphenation, ilustrada en la siguiente figura) que puede dejarse en una línea antes de que ocurra la separación por guiones; el valor de la propiedad es una longitud en píxeles o un porcentaje.
- **-ms-hyphenate-limit-chars**: Especifica el número mínimo de caracteres en una palabra que puede tener guión; si el número de caracteres es inferior al mínimo, la palabra no tiene guión.
- **-ms-hyphenate-limit-lines**: Especifica el número máximo de líneas con guión que pueden contener palabras con guión

La siguiente marca de revisión con CSS en línea utiliza la propiedad **-ms-hyphens**, que está configurada como **auto**:

```
<!doctype html>
<HTML lang="en-us">
<body>
  <div style="width: 200px;
  border: 2px solid orange;">
    <p style="-moz-hyphens: auto;
    text-align: justify;
    font-size: 14pt;">
      En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que
      viv&iacutetea un hidalgo de los de lanza en astillero, adarga antigua, roc&iacuten flaco y
      galgo corredor. Una olla de algo m&aacutes vaca que carnero, salpic&oacuten las m&aacutes
      noches, duelos y quebrantos los s&aacutebados, lentejas los viernes, alg&uacuten palomino de
      a&ntildeadadura los domingos, consum&iacutean las tres partes de su hacienda. El resto della
      conclu&iacutean sayo de velarte, calzas de velludo para las fiestas con sus pantuflos de lo
      mismo, los d&iacutetas de entre semana se honraba con su vellori de lo m&aacutes fino. </p>
    </div>
  </body>
</html>
```

Esta marca aparece en Mozilla Firefox como se muestra en la siguiente imagen.

En un lugar de la Mancha,  
de cuyo nombre no quiero  
acordarme, no ha mucho  
tiempo que vivía un hi-  
dalgo de los de lanza en  
astillero, adarga antigua,  
rocín flaco y galgo corre-  
dor. Una olla de algo más  
vaca que carnero, salpicón  
las más noches, duelos y  
quebrantos los sábados,  
lentejas los viernes, algún  
palomino de añadidura los  
domingos, consumían las  
tres partes de su hacienda.  
El resto della concluían  
sayo de velarte, calzas de  
velludo para las fiestas  
con sus pantuflos de lo  
mismo, los días de entre  
semana se honraba con su  
vellori de lo más fino.

**Nota:** Aunque *Microsoft* ha habilitado varias propiedades de separación por guiones en aplicaciones de *Internet Explorer 10* y *Windows 8*, la especificación del *W3C* para la separación por guiones sigue evolucionando. El *W3C* está trabajando en las propiedades `hyphenate-character`, `hyphenate-limit-zone`, `hyphenate-limit-word`, `hyphenate-limit-lines`, y `hyphenate-limit-last`.

### 6.2.3. Usando CSS Exclusions para crear texto flotante alrededor de un objeto flotante

Los flotamientos se usaron en el ejercicio del principio de este capítulo. A ambos flotamientos se les asignó el valor `left`, lo que hizo que aparecieran uno al lado del otro cuando se renderizó. La colocación fue relativa al resto del documento. Además, si no se especifica una altura o un ancho para un flotamiento, el elemento cambia automáticamente de tamaño para ajustarse a su contenido.

Con *CSS Exclusions*, se puede controlar la posición de un flotamiento con precisión, a una distancia especificada, desde la parte superior, inferior, izquierda o derecha de un contenedor. También se puede crear un flotamiento en cualquier forma: rectangular, circular, triangular, y casi cualquier cosa en el medio. Las otras partes de el documento simplemente fluye alrededor de la exclusión.

El ejemplo de CSS Exclusions mostrado en la siguiente imagen, utiliza un diseño de varias columnas con una exclusión (referido como flotador posicionado en el sitio Web) en el centro. Se puede mover la caja a cualquier lugar en el diseño del ejemplo y ver que el texto circundante fluye automáticamente alrededor del cuadro azul. La siguiente tabla lista las propiedades de W3C relacionadas con CSS Exclusions:

## Overlapping exclusions

Lorem ipsum dolor sit amet, consectetur adip-  
 isicing elit, sed do eiusmod tempor incididunt  
 ut labore et dolore magna aliqua. Ut enim ad  
 minim veniam, quis nostrud exercitation ul-  
 lamco laboris nisi ut aliquip ex ea commodo  
 consequat. Duis aute irure dolor in repre-  
 henderit in voluptate velit esse cillum  
 dolore eu fugiat nulla pariatur. Ex-  
 cepteur sint occaecat cupidatat  
 non proident, sunt in culpa qui  
 officia deserunt mollit anim  
 id est laborum. Lorem ipsum  
 dolor sit amet, bore et do-  
 consectetur lore magna  
 adipisicing elit, aliqua. Ut enim  
 sed do eius- ad minim veniam,  
 mod tempor quis nostrud exercita-  
 incididunt ut la- tion ullamco laboris nisi ut aliquip ex ea com-  
 bore et dolore modo consequat. Duis aute irure dolor in rep-  
 magna aliqua. rehenderit in voluptate velit esse cillum dolore  
 Ut enim ad eu fugiat nulla pariatur. Excepteur sint occaecat  
 minim veniam, cupidatat non proident, sunt in culpa qui officia  
 quis nostrud deserunt mollit anim id est laborum. Lorem ip-  
 exercitation ul- sum dolor sit amet, consectetur adipisicing elit, laboris nisi ut aliquip ex ea  
 lamco laboris sed do eiusmod tempor incididunt ut labore et commodo consequat. Duis  
 nisi ut aliquip ex ea commodo consequat. aute irure dolor in reprehenderit in voluptate  
 Duis aute irure dolor in reprehenderit in vo- velit esse cillum dolore eu fugiat nulla paria-  
 luptate velit esse cillum dolore eu fugiat nulla tur. Excepteur sint occaecat cupidatat non  
 pariatur. Excepteur sint occaecat cupidatat proident, sunt in culpa qui officia deserunt

Propiedad	Valor	Descripción
shape- outside	auto shape url	Crea la forma general de una exclusión
shape-inside	outside-shape auto shape uri	Modifica la forma del contenido
wrap	wrap-flow wrap-margin wrap-padding	Es un método abreviado para establecer las propiedades de flujo de envoltura, margen de envoltura y relleno de envoltura en una sola declaración.



<b>wrap-flow</b>	<b>auto</b> <b>both</b> <b>start</b> <b>end</b> <b>maximum</b> <b>clear</b>	Especifica cómo las exclusiones afectan al contenido en línea dentro de los elementos de nivel de bloque.
<b>wrap-margin</b>	<b>length</b>	Proporciona una compensación para el contenido fuera del elemento
<b>wrap-padding</b>	<b>length</b>	Proporciona un pad (un offset) para el contenido dentro de un elemento
<b>wrap-through</b>	<b>wrap</b> <b>none</b>	Especifica cómo debe envolverse el contenido alrededor de un elemento de exclusión.

Para crear una exclusión simple de CSS, se usa la propiedad **wrap-flow: both** para mostrar contenido en todos los lados de la exclusión. Otra opción es usar **wrap-flow: clear**, que muestra el contenido por encima y por debajo de la exclusión, pero deja los lados en blanco.

Se declara una forma de exclusión utilizando las propiedades **shape-inside** y **shape-outside**, que definen el contenido y la forma general de una exclusión, respectivamente.

Actualmente, las CSS Exclusions no funcionan de forma fiable en ninguno de los principales navegadores. Si se quiere experimentar con CSS Exclusions, se debe probar *Internet Explorer 10*. Para utilizar cualquier propiedad de CSS Exclusions para renderizar en *Internet Explorer 10*, se debe añadir el prefijo **-ms-**. *Microsoft* también utiliza las propiedades **-ms-wrap-side** y **-ms-flow-wrap**, que no forman parte de la especificación W3C.



## 7. Entendiendo JavaScript y Fundamentos de Programación

---

### 7.1. Breve Historia de JavaScript

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y, por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en *Netscape*, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como *ScriptEase*) al navegador *Netscape Navigator 2.0*, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje *LiveScript*.

Posteriormente, *Netscape* firmó una alianza con *Sun Microsystems* para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento *Netscape* decidió cambiar el nombre por el de *JavaScript*. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de *JavaScript* fue un completo éxito y *Netscape Navigator 3.0* ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, *Microsoft* lanzó *JScript* con su navegador *Internet Explorer 3*. *JScript* era una copia de *JavaScript* al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, *Netscape* decidió que lo mejor sería estandarizar el lenguaje *JavaScript*. De esta forma, en 1997 se envió la especificación *JavaScript 1.1* al organismo *ECMA* (European Computer Manufacturers Association).

*ECMA* creó el comité *TC39* con el objetivo de "estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa". El primer estándar que creó el comité *TC39* se denominó *ECMA-262*, en el que se definió por primera vez el lenguaje *ECMAScript*.

Por este motivo, algunos programadores prefieren la denominación *ECMAScript* para referirse al lenguaje *JavaScript*. De hecho, *JavaScript* no es más que la implementación que realizó la empresa *Netscape* del estándar *ECMAScript*.

La organización internacional para la estandarización (*ISO*) adoptó el estándar *ECMA-262* a través de su comisión *IEC*, dando lugar al estándar *ISO/IEC-16262*.

### 7.2. Gestión y Mantenimiento JavaScript

En los programas de *JavaScript*, muchos elementos tienen que cooperar para obtener un buen resultado. *JavaScript* se diferencia de *CSS* y de *HTML* en algunos aspectos; en esta lección se aprenderán nuevos conceptos para hacer el mejor uso de *JavaScript*. *HTML* y *CSS*, por ejemplo, están muy enfocados en hacer que las cosas se vean de una forma particular. *JavaScript*, en cambio, le da más atención a cómo las cosas actúan. Para comprobar el funcionamiento de una aplicación de *JavaScript*, podría ser necesario mirar el display repetidamente en el tiempo y, quizás, interactuar con él.



Esta lección se enfoca en crear programas *JavaScript* y usar funciones. Con esta lección se aprenderá a crear programas simples y usar librerías de *JavaScript*, como *jQuery*, u otras. También se aprenderá a localizar y acceder a elementos, escuchar y responder a eventos, mostrar y ocultar elementos, actualizar el contenido de elementos, y añadir elementos durante el flujo de la aplicación.

### 7.2.1. Creación y uso de Funciones

Una función es una parte del programa definida y ejecutada de forma aislada. La acción de una función es la secuencia de acciones que hay dentro de ella.

En principio, todo programa de *JavaScript* puede ser escrito como una declaración tras otra en la secuencia exacta en la que se deberían de ejecutar. Los programadores han encontrado, sin embargo, que es útil para introducir símbolos, o nombres para partes especiales de un programa. La primera de estas partes es una función. En la programación, una función es un fragmento de programa definido y ejecutado aisladamente de otras partes. Es común en la programación utilizar funciones escritas por otras personas, a veces sin unas instrucciones detalladas de cómo funcionan exactamente.

Una función de programación tiene dos propósitos principales:

- Una tarea realizada en múltiples situaciones puede ser definida una sola vez y utilizada en múltiples casos con confianza de que su comportamiento será el mismo. Esta forma de trabajar es menos propensa a errores que la repetición de código en cada contexto donde sea necesario.
- Para la persona que escribe, lee o mantiene el programa es útil e informativo, ya que se pueden identificar fragmentos del programa con los nombres de la función. Así como los libros tienen capítulos, con nombres que explican sus acciones o temas a tratar, los programas de ordenador tienen funciones. Es importante entender que la acción de una función es contener una secuencia de acciones declaradas dentro de ella. Cuando se ejecuta un programa que contiene una función, el programa simplemente ejecuta sentencias dentro de la función. También es importante distinguir entre la definición y la ejecución de una función. La expresión "**function ejemplo1**() {...}" no ejecuta el código que contiene en su interior. Lo que se ve en el código es sólo la definición de una función. Sólo cuando la función es invocada para nuestro propósito se ejecutará su código.

Mientras el nombre de la función está bajo nuestro control, hay algunas restricciones en nuestra elección: el nombre debe estar compuesto de letras, dígitos, guiones bajos y signos de dólar, y el primer carácter del nombre debe ser una letra, guión bajo o signo de dólar. "*ejemplo\$1*" es un posible nombre de función *JavaScript*, pero "*no.con.puntos*", "*1mal*", y "*nombre/apellidos*" no lo son.

Los programas *JavaScript* tienen otro tipo de abreviatura simbólica común, a la que se llama **variable**. Mientras que una función contiene un conjunto de secuencias de acciones, una variable contiene datos. Se utiliza la sintaxis **var** para definir una nueva variable en *JavaScript*.

Las reglas para los identificadores *JavaScript* -básicamente, los nombres de las variables y funciones- son en realidad algo más complicado que lo que se ha escrito arriba. Los identificadores no pueden ser iguales que palabras clave ya utilizadas en el lenguaje; "*if*", por ejemplo, tiene un significado especial en *JavaScript* y no está disponible como nombre de variable. Sin embargo, los caracteres pueden ser de otros alfabetos que no sean el inglés, bajo ciertas circunstancias.



### 7.2.2. Usando jQuery y otras Librerías

Los programadores más efectivos saben cómo hacer buen uso de lo que otros escriben. Una librería de programación contiene fragmentos de código, subrutinas, clases y otros recursos que se pueden reutilizar para crear software. Hay muchas librerías *JavaScript* disponibles, y *jQuery* es una de los más populares.

Otros programadores ya han pensado en muchas de las tareas a las que un programador se puede enfrentar, ya sea la confirmación de que los números de la tarjeta de crédito son inválidos, la visualización de los datos médicos o la construcción de un servicio de chat para un equipo repartido en cuatro continentes. Se puede aprovechar lo que otros programadores ya han escrito, utilizando las librerías de *JavaScript*. Una librería es una colección de recursos, como códigos de función y subrutinas, que los desarrolladores usan para crear programas. Los programadores de *JavaScript* a veces identifican funciones que no devuelven ningún valor como subrutinas.

La mayoría de los lugares de trabajo ya tienen establecida la política sobre qué librerías se deben invocar y cómo hacerlo. La Web está llena de consejos sobre bibliotecas y su aplicabilidad. Muchas librerías, pero no todas, están disponibles sin coste alguno.

Algunas librerías tienen políticas formales de "apoyo", es decir, un compromiso de responder cuando se notifican los fallos. Otras están destinadas a un número reducido de programadores: por ejemplo, una librería que facilita el desarrollo de aplicaciones que controlan panaderías de escala industrial, mientras que otras están dirigidas a todos los que programan en *JavaScript*.

*jQuery* es la biblioteca líder de *JavaScript*. Más de la mitad de los 10.000 sitios web más visitados en el mundo usan *jQuery*. *jQuery* está disponible para su uso sin coste y con restricciones mínimas. También tenga en cuenta que *Microsoft* y otros líderes de la industria hacen copias de *jQuery* disponibles para para descargar y usar.

Es posible utilizar librerías de terceros, además de *jQuery*. La mayoría de las veces, se darán instrucciones explícitas sobre qué librería usar. Cuando una opción está disponible, se puede encontrar mucha información en la Web con una búsqueda del nombre de la librería. Además de *jQuery*, hay otras librerías populares son *Dojo*, *MooTools* y *YUI*. Cuando se utiliza una librería de terceros, normalmente se necesita incluir un elemento como el siguiente:

```
<script type = "text/JavaScript" src = "web o dirección local de La Librería JavaScript">  
</script>
```

Cuando se quiere usar `wonderful_function()` de una librería de terceros en una de las páginas que se están programando, la única manera de que el navegador sepa lo que se quiere decir con `wonderful_function()` es a través de la referencia a su aparición en la librería. El `<script ....>` proporciona esa referencia. También es necesario que se lea la documentación de la biblioteca que se desea utilizar.

### 7.3. Actualizando la UI usando JavaScript

En este punto ya se conocen los conceptos básicos de *HTML*: navegación por medio de hipervínculos, recuperar datos mediante envío de formularios y todo el estilo visual del *HTML* tradicional.





Si se necesita que una aplicación haga funciones como mostrar en vivo actualizaciones sobre los horarios u otras cosas, responder rápidamente y en detalle sobre la información de una cuenta, asesorar de manera personalizada y mucho más. Sólo *JavaScript* hace posible todo este dinamismo e interactividad.

Las capacidades de *JavaScript* incluyen la entrada de datos, la respuesta a las pulsaciones del teclado y el ratón, movimientos, visualización de resultados, cálculos complejos y más. Las aplicaciones móviles construidas sobre la base de *HTML5*, por supuesto, tienen las mismas capacidades.

### 7.3.1. Localizando y Accediendo a Elementos

Si se ve algo en el navegador, *JavaScript* puede "llegar a él" y ponerlo bajo control del programador. Se puede utilizar el método `getElementById()` para acceder a los elementos de visualización por su ID. Éste devuelve una referencia al primer objeto con el id o atributo NAME especificado.

La validación es una responsabilidad importante para *JavaScript*: ¿Se ha introducido el usuario una cifra que se ajuste a sus necesidades? ¿Una restricción presupuestaria? ¿Es un número de tarjeta de crédito legal? ¿Es una dirección de correo electrónico permitida y no entra en conflicto con una ya existente? A menudo se tiene que confirmar que la entrada de un usuario es numérica, en el sentido de que 0 y 3.141 son números correctos, pero *abc* y *3.141.* no lo son. Aunque tendría sentido que *JavaScript* tuviese una función como `is_a_number()`, por razones históricas sólo incluye `isNaN`.

¿Por qué se llama a `getElementById` un "método"? ¿No incluye *JavaScript* una librería de funciones para realizar tareas útiles?

Sí y no. Aunque *JavaScript* incluye una librería de funciones útiles, las más comunes están fuertemente ligadas a objetos específicos definidos por *HTML*. Estas capacidades se denominan métodos. Se diferencian de las funciones sólo en que están siempre asociados y utilizados con un objeto en particular. `isNaN()` es un ejemplo de una función *JavaScript*, que prueba para "no es número". Si `isNaN()` devuelve un valor de 0 (`false`), el valor es un número. `document.getElementById()` es un ejemplo de un método *JavaScript*; se puede utilizar únicamente `getElementById` con el objeto de documento especial.

### 7.3.2. Escuchando y Respondiendo a Eventos

Un "evento" es un concepto crucial en la programación interactiva. Mucha programación *JavaScript* tiene que ver con las respuestas cuando algo sucede. El evento `Load` se utiliza comúnmente y se activa cuando su propietario está listo.

Muchos requisitos de la aplicación implican eventos, que son acciones que desencadenan otras acciones. Las descripciones en términos de "when" o "if" se codifican típicamente en *JavaScript* en términos de eventos. Esto puede sorprender a los programadores provenientes de otros lenguajes, donde el énfasis está en la secuencia:

- 1- Se debe hacer lo primero.
- 2- Entonces Se debe hacer lo segundo.
- 3- Entonces Se debe hacer lo tercero.
- 4- Se debe completar la secuencia.

La programación basada en eventos, en cambio, se parece más a un diálogo: un usuario toma alguna acción, entonces el programa *JavaScript* responde, y así sucesivamente.





Todos los programas de ejemplo de *JavaScript* presentados hasta ahora han implicado eventos. Para asignar un valor al controlador de eventos **onclick** para el evento Click tiene el siguiente efecto: "Cuando el usuario final hace click en el elemento en cuestión, luego ejecuta el script dado por el comando **onclick**". En este caso, la acción de click es el evento, y el script es la respuesta, o devolución de llamada.

Las referencias *JavaScript* tabulan todos los eventos reconocidos. Entre los que a menudo se programan, más allá de haciendo click en un elemento, están las siguientes:

- Presentación de un formulario.
- Pulsaciones.
- Otras maniobras del ratón, como hacer doble clic y mover el ratón.
- Selección de un elemento de un listbox.
- El momento en que una imagen ha terminado de cargarse.

El controlador de eventos **onload** es más importante de lo que muchos principiantes creen. **onload** "pertenece" a los elementos *HTML*; se activa cuando quien lo contiene está listo. El **onload** para una **<img>** (imagen) se produce cuando la imagen está totalmente renderizada y visible; el **onload** de una tabla se ejecuta una vez todas las celdas de esta tabla han sido generadas.

### 7.3.3. Mostrando elementos ocultos

Se pueden mostrar y ocultar elementos utilizando el atributo de visualización *HTML* para hacer su visualización "inteligente", mostrando al usuario final la información pertinente y ocultándola cuando ya no es necesario.

Una pantalla en particular puede mostrar diferentes tipos de información dependiendo de las circunstancias: un número de teléfono fuera del horario laboral, una advertencia sobre el uso sólo durante un mes con exceso de tráfico, o avisos sobre la actividad de otros usuarios sólo cuando otros usuarios han iniciado sesión. Una forma conveniente de organizar tales variaciones es diseñar todos los mensajes posibles y, a continuación, mostrar sólo los pertinentes. Los elementos *HTML* tienen un atributo de visualización útil para este enfoque.

### 7.3.4. Actualizando el Contenido de Elementos

*JavaScript* tiene la capacidad de hacer cálculos complejos, utiliza la propiedad *innerHTML* para cambiar el contenido actual de los elementos *HTML* (referido como "contenido interno") o insertar nuevo contenido.

### 7.3.5. Añadiendo Elementos

Se puede utilizar el comando **createElement** y el método **appendChild** en *JavaScript* para añadir elementos después de que el *HTML* esté completo.

*JavaScript* permite realizar modificaciones aún más radicales en una pantalla. Cuando es necesario, es posible crear nuevos elementos y encajarlos en una pantalla existente. Esto se logra con el comando **createElement** y el método **appendChild**.



## 8. Trabajando con Gráficos y Accediendo a Datos

### 8.1. Trabajando con Imágenes, Formas y otros Gráficos

*JavaScript* puede mostrar diferentes tipos de gráficos, desde archivos *JPG* y *PNG*, hasta formas como cajas y círculos. Una forma es usar el método `createElement()`. Este método crea una función reutilizable para mostrar una imagen:

```
function mostrar_imagen(src, ancho, alto, alt) {  
    var img = document.createElement("img");  
    img.src = src;  
    img.width = ancho;  
    img.height = alto;  
    img.alt = alt;  
    // Se añade esto al <body>  
    document. body.appendChild(img);  
}
```

Para mostrar la imagen, se incluye este código:

```
<button onclick="mostrar_imagen('path/filename', 276,110, 'Logo');">  
    Mostrar logo  
</button>
```

Mostrar una imagen cuando se pulsa un botón es una tarea bastante sencilla. En realidad, la creación de gráficos sobre la marcha requiere un *canvas* o un *SVG*. Esta lección cubre la creación de gráficos usando *canvas*.

#### 8.1.1. Manipulando *Canvas* con *JavaScript*

Con *HTML5* y *JavaScript* se pueden controlar fácilmente no sólo textos, formularios e imágenes estáticas, también se pueden dibujar gráficos complejos. El elemento de *canvas* de *HTML5* es un área de dibujo bajo control del programa. Como se aprendió anteriormente, el elemento de *canvas* crea un contenedor para los gráficos y utiliza *JavaScript* para dibujar los gráficos dinámicamente. Con *JavaScript*, también se pueden animar objetos haciendo que se muevan, cambien de escala, etc.

Para dibujar un objeto de *canvas*, principalmente se utiliza la función `getElementById()`, para encontrar el elemento de *canvas*, y `canvas.getContext` (a veces abreviado como `c.getContext`) para crear el objeto de *canvas*. A continuación, se pueden utilizar una variedad de métodos para dibujar formas, incluir imágenes, etc. Aplicaciones enteras, incluyendo juegos y simuladores impresionantes, han sido construidas con *Canvas* de *HTML5*.



## 8.2. Enviando y Recibiendo Datos

Si una aplicación a la que se puede acceder desde un ordenador muestra datos, hay probablemente una forma de que *JavaScript* llegue a esos datos. Uno de los papeles más importantes de *JavaScript* es comunicarse en tiempo real con fuentes de datos remotas.

Parte del poder de las aplicaciones informáticas modernas es su capacidad para coordinar la información de muchas fuentes en una pantalla útil: *JavaScript* puede reunir bases de datos del lado del servidor del historial del cliente, el contenido de la página actual, actualizaciones remotas de precios o detalles meteorológicos, entre otros.

La creación de programas *JavaScript* que envían y reciben datos es un reto, no porque sean difíciles, sino que es debido a que una aplicación *JavaScript* tiene una estructura diferente a cualquier otra. Cada página individual puede ser cargada en un navegador y usada de forma aislada.

Sin embargo, un programa *JavaScript* que envía y recibe datos debe tener más partes: debe haber un receptor o emisor en algún lugar con el que el programa *JavaScript* pueda intercambiar datos.

La mayoría de las veces, las páginas *HTML* o las aplicaciones para móviles se comunican con un servidor en red en alguna aplicación central.

Las transacciones de datos *JavaScript* a menudo se presentan como conceptualmente difíciles y se dice que sólo se pueden afrontar si se domina *AJAX*, *XML*, *JSON*, y un complicado conjunto de otros acrónimos. Esto no es verdad.

Mediante la comunicación de red *JavaScript* se pueden pasar cero o más piezas de datos, y se recibe un resultado de vuelta. La gran dificultad está en la creación del propio "laboratorio" con las partes necesarias para que este diálogo funcione. Una vez que se tenga ese primer canal funcionando sin problemas, se aprenderán rápidamente las reglas de seguridad, control de flujo y codificación propias de Comunicaciones de datos *JavaScript*.

Una de las técnicas principales para la transferencia de datos es la API *XMLHttpRequest* (a veces abreviado como *XHR*). *XMLHttpRequest* le permite utilizar *JavaScript* para pasar datos en forma de cadenas de texto entre un cliente y un servidor. La sintaxis general podría parecer como la siguiente:

```
function load(url, data, callback) {  
    var xhr = new XMLHttpRequest();  
    xhr.open("GET", url, true);  
    xhr.onload = function() {  
        callback(xhr.responseText);  
    }  
    xhr.send(data);  
}
```

El objeto *XMLHttpRequest* crea una llamada al servidor. El método abierto especifica el método *HTTP* para contactar con el servidor y proporcionar la dirección Web del servidor. La devolución de la llamada obtiene una respuesta del servidor. Finalmente, *xhr.send(data)* envía los datos.

Más adelante se va a requerir el uso de un servidor Web: la caché de aplicaciones, por ejemplo, sólo existe para sitios a los que se accede a través de una conexión de red. Casi cualquier servidor Web es un buen compañero de prácticas. Se debe configurar el acceso a un servidor Web para que se pueda probar el uso de



la caché de aplicaciones con poca dificultad. Para enviar y recibir datos explícitamente en *JavaScript* es diferente; para que funcione se necesita un Servidor Web dinámico y programación del lado del servidor.

Cuando se elija trabajar con servidores Web, se tendrá una gran selección de alternativas que son relativamente fáciles para un recién llegado al servicio Web. *Wamp* para *Windows*, *mamp* para *Mac*, y *Lamp* para *Linux* están entre los puntos de partida más sencillos.

### 8.2.1. Transmitiendo Objetos Complejos y Parsing

Como lenguaje de propósito general, *JavaScript* puede comunicar información estructurada mucho más rica que los valores simples usados en la mayor parte de esta lección.

En la programación del mundo real, *JavaScript* puede manejar operaciones muy complicadas. Uno podría, por ejemplo, recibir un informe sobre los precios de la gasolina en una larga lista de puntos de venta al por menor y necesitar la posibilidad de aislar sólo una de esas figuras. **Parsing** es un término usado para describir un análisis de información compleja en partes constituyentes.

*JavaScript* y las librerías de *JavaScript* disponibles gratuitamente proporcionan una gran cantidad de funciones de análisis.

La extracción de datos de una página web externa legible por el ser humano es sólo un ejemplo entre otras muchas de las formas en que *JavaScript* puede analizar los datos.

Por ejemplo, también se puede utilizar *JSON* (*JavaScript Object Notation*) para intercambiar objetos *JavaScript* con un servidor. Usando las APIs *JSON.parse* y *JSON.stringify*, el código podría ser como el siguiente:

```
function loadJSON(url, data, callback) {  
    var xhr = new XMLHttpRequest();  
    xhr.open("GET", url, true);  
    xhr.onload = function() {  
        callback( JSON.parse(xhr.responseText) );  
    }  
    xhr.send( JSON.stringify(data) );  
}  
loadJSON("my.json", { id : 1 }, function(response) {  
    setTitle(response.title);  
});
```

Este código es casi idéntico al ejemplo de código *XMLHttpRequest* del apartado anterior. El objeto *XMLHttpRequest* crea una llamada al servidor, y el método abierto especifica el método *HTTP* para contactar con él y proporcionar la dirección Web de éste. La devolución de llamada utiliza *JSON* para obtener una respuesta del servidor. Cuando el servidor responde, el comando llamado *JSON.parse API*, crea el objeto *JavaScript*. El objeto se devuelve al servidor. Los datos, sin embargo, se "encadenan" al principio.

## 8.3. Cargando y Guardando Archivos

*JavaScript* puede acceder a los archivos de su ordenador local y, con *HTML5*, se puede validar el tipo de archivo antes de la carga. *JavaScript* hace que la carga de archivos sea más interactiva y libre de errores.



Muchas aplicaciones Web o para móviles incluyen una función para subir un archivo. Esto ha sido durante mucho tiempo una debilidad de *HTML*, en el sentido de que no existe una forma efectiva de especificar, por ejemplo, "sólo permitir carga de imágenes, a diferencia de los documentos, y sólo si las imágenes ocupan menos de 1.1 megabytes." Sin esta capacidad, ocurre con demasiada frecuencia que los usuarios intentan accidentalmente cargar algo distinto de lo que se pretendía o de lo que la aplicación soporta, y al conectarse en red, los retrasos hacen que la corrección del error sea un proceso que requiera mucho tiempo.

La capacidad de *HTML5* para acceder a archivos locales implica que una imagen destinada a la carga puede ser anunciada como una miniatura y validado antes de la carga. Las acciones inmediatas de *JavaScript* ayudan a que programar sea un proceso más interactivo y libre de errores.

### 8.3.1. Usando AppCache

Los usuarios quieren poder navegar incluso cuando están desconectados. La caché de aplicaciones (*AppCache*) lo hace posible. *AppCache* difiere de la caché del navegador en que la caché del navegador contiene todas las páginas web visitadas, mientras que *AppCache* sólo guarda los archivos listados en el manifiesto de caché. Se puede aplicar *AppCache* a una sola página Web o a un sitio entero.

*AppCache* guarda una copia de los archivos de su sitio Web localmente, de forma estructurada. Los archivos incluyen *HTML*, *CSS* y *JavaScript*, junto con otros recursos que el sitio necesita para funcionar. Después de que alguien haya visitado el sitio una vez, las visitas siguientes cargarán los recursos rápidamente desde la copia local en lugar de esperar una conexión de red.

Para que *AppCache* funcione correctamente, el servidor Web debe estar configurado con el tipo de MIME correcto, el cual es *text/cache-manifest*. Además, la extensión de archivo preferida para los archivos de manifiesto es *.appcache*.

### 8.3.2. Entendiendo y Usando Tipos de Datos

En *JavaScript*, los datos tienen tipos diferentes. Las más comunes son los strings y numbers. Otros tipos de datos son **array**, **bool**, **null**, **object** y **undefined**.

"ABCD" y "1234" son ejemplos de string; este último sólo incluye dígitos. 3 es un número, pero "3 dólares" es una cadena. Las comillas en *JavaScript* tratan los caracteres como cadenas de texto.

La principal importancia de los tipos de datos en este nivel es que su definición conduce a unas pocas sorpresas. En *JavaScript*, el siguiente ejemplo tiene el valor 123, ya que, normalmente si se ponen comillas al primer carácter, *JavaScript* trata al resto de caracteres como cadenas de texto, sin importar si tienen comillas o no, entonces los concatena:

```
'1' + 2 + 3
```

Pero lo siguiente tiene el valor 6 o '33', dependiendo del intérprete de *JavaScript*, aunque normalmente, sumará los dos primeros caracteres al ser dígitos, pero se concatena con el último al tener comillas y considerarse como un string:

```
1 + 2 + '3'
```

Hay reglas que definen todo este comportamiento, y cada uno de los tipos de datos tienen sus usos. Para el de esta lección, se debe ser consciente de la distinción entre un número y el carácter que representa ese número.



## 8.4. Usando *JavaScript* para Validar Campos de Formularios Introducidos por Usuarios

A medida que los usuarios finales introducen los datos en un formulario, *JavaScript* puede validar instantáneamente las entradas y sugerir alternativas.

Supóngase, por ejemplo, que un usuario final necesita introducir un número de serie del formulario XXX-XXXXXXX-X, donde cada X es un dígito. En los primeros días de la aplicación Web, los usuarios finales los escribían lo mejor que podían, entonces “enviaron” un formulario completo. El servidor verificaba si había algún error, y los reportaba lo mejor que podía.

Con *JavaScript*, la validación de la entrada del lado del cliente en forma de retroalimentación y corrección es instantánea.

## 8.5. Entendiendo y usando Cookies

Las cookies han mantenido tradicionalmente información que permite la personalización y la comodidad. *JavaScript* puede crear y recuperar cookies.

Las cookies son pequeños archivos de texto que los sitios web guardan en el disco duro de un ordenador que contienen información sobre el usuario y sus preferencias de navegación. El contenido de las cookies cambia cuando un usuario vuelve a visitar un sitio y selecciona diferentes elementos o cambia las preferencias. Desde la perspectiva de *JavaScript*, una cookie es una variable, y se utiliza *JavaScript* para crear y recuperar cookies.

En un supuesto juego de ordenador codificado en *HTML* y *JavaScript* funciona bien, pero requiere que el usuario elija un "nivel" cada vez que comienza una nueva partida. Sería una buena idea que el juego asuma que el usuario quiere empezar un nivel más allá de su última partida, con una opción de ajuste. Las cookies de *JavaScript* facilitan la programación.

## 8.6. Entendiendo y usando Almacenamiento Local

Las cookies están limitadas en la información que almacenan y, por lo tanto, en las acciones que se pueden hacer con ellas. Además, las cookies suponen una amenaza para la privacidad de los datos. *HTML5* proporciona almacenamiento local para hacer la personalización más fácil de programar y más capaz.

Las cookies tienen una mala reputación en algunos círculos por contribuir a la propagación de la información personal sin permiso, y son torpes para el almacenamiento. *LocalStorage* de *HTML5* tiene mejor seguridad y hace la programación más fácil que con las cookies.

