Identificación: Anexo02-Trabajndo-HTML-CSS-JS
Versión: 1.0

Tabla de contenido

HTML5 – CSS3 - JAVASCRIPT

HTML5 provee básicamente tres características: estructura, estilo y funcionalidad. Nunca fue declarado oficialmente pero, incluso cuando algunas APIs (Interface de Programación de Aplicaciones) y la especificación de CSS3 por completo no son parte del mismo, HTML5 es considerado el producto de la combinación de HTML, CSS y Javascript. Estas tecnologías son altamente dependientes y actúan como una sola unidad organizada bajo la especificación de HTML5. HTML está a cargo de la estructura, CSS presenta esa estructura y su contenido en la pantalla y Javascript hace el resto.

En el presente documento hay presente varios ejemplos los cuales se recomienda que cada uno de ellos los realicen en su editor de código preferido.

HTML -- Estructura Global

<!DOCTYPE html>

Es la primer etiqueta y antes de ella no debe haber texto ni lineas en blanco.

<html>

```
<!DOCTYPE html>
<html lang=”es”>
</html>
```

Lang = Este atributo se ubuica en la etiqueta de apertura <html> y atributo define el idioma humano del contenido del documento que estamos creando, en este caso es por español.

<head> Continuemos construyendo nuestra plantilla. El código HTML insertado entre las etiquetas <html> tiene que ser dividido entre dos secciones principales. Al igual que en versiones previas de HTML, la primera sección es la cabecera y la segunda el cuerpo. El siguiente paso, por lo tanto, será crear estas dos secciones en el código usando los elementos <head> y <body> .

```
<!DOCTYPE html>
<html lang="es">
        <head>
        </head>
</html>
```

<**meta**> construye la cabecera del documento. Algunos cambios e innovaciones fueron incorporados dentro de la cabecera, y uno de ellos es la etiqueta que define el juego de caracteres a utilizar para mostrar el documento. Ésta es una etiqueta <meta>que especifica cómo el texto será presentado en pantalla:

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”identificador-juego-de-caracteres”>
        </head>
</html>
```

Ejemplo

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”iso-8859-1”>
        </head>
</html>
```

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
        </head>
</html>
```

https://www.w3.org/TR/WD-html40-970708/charset.html

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
        </head>
</html>
```

<**link**> Otro atributo en la cabecera del documento es <link>. Este elemento es usado para incorporar estilos, códigos Javascript, imágenes o iconos desde archivos externos. Uno de los usos más comunes para <link> es la incorporación de archivos con estilos CSS:

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <link rel=”stylesheet” href=”direccion.css”>
        </head>
</html>
```

Otra forma de incluir CSS dentro del mismo documento con la etiqueta <style> → **No recomendado**

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <style>
            </style>
        </head>
</html>
```

<**body>** La siguiente gran sección que es parte principal de la organización de un documento HTML es el cuerpo. El cuerpo representa la parte visible de todo documento y es especificado entre etiquetas <body>.

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <link rel=”stylesheet” href=”direccion.css”>
            <body>
            </body>
        </head>
</html>
```

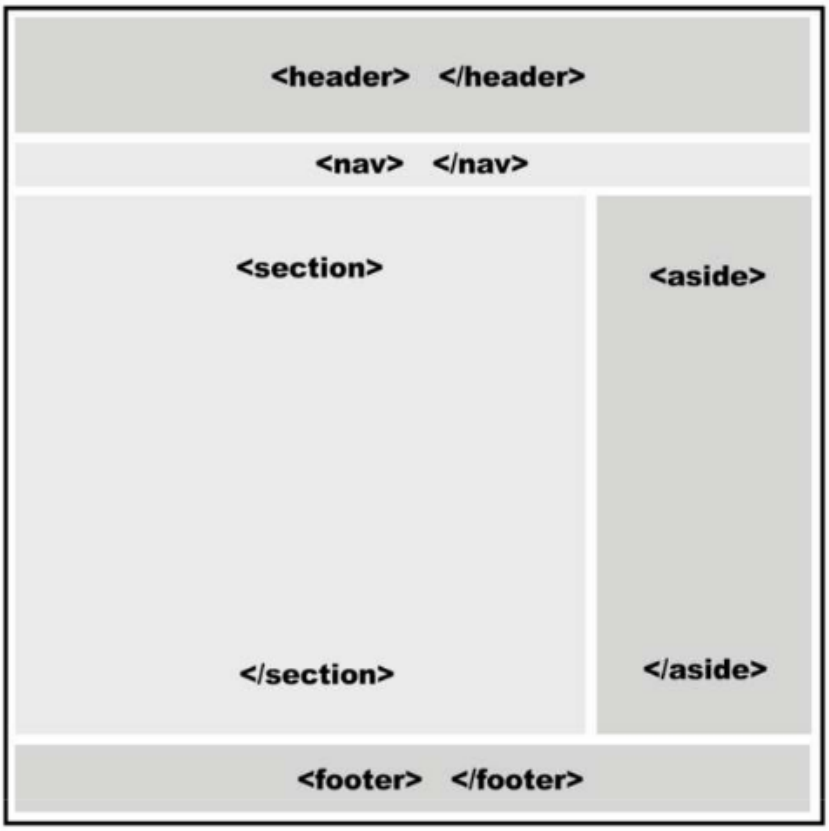
Organización

Ejemplo de la organziación del sitio.

Descripción:
1.Cabecera

2.Barra de Navegación
3.Sección de Información Principal
4.Barra Lateral
5.El pie o la barra Institucional

Como se distribuye los contenidos con una estructura básica



Vamos a realizar la codificación de acuerdo a la imagen anterior

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <link rel=”stylesheet” href=”direccion.css”>
            <body>
            <nav>
                        <ul>
                                    <li>principal</li>
                                    <li>fotos</li>
                                    <li>videos</li>
                                    <li>contacto</li>
                        </ul>
                </nav>
                <section>
                </section>
                <aside>
                    <blockquote>Mensaje número uno</blockquote>
                    <blockquote>Mensaje número uno</blockquote>
                </aside>
                <footer>
                    Derechos Reservados SENA
                </footer>
                </body>
        </head>
</html>
```

Dentro del Cuerpo <body>

<article> Cuando los elementos <article> están anidados, los internos representan artículos relacionados con el exterior. Por ejemplo, los comentarios de un blog pueden ser elementos <article> anidados al que representa la entrada del blog.

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
```

```
        <meta name=”description” content =”Descripcion del Sitio”>
        <meta name=”keywords” content =”html5, css3”>
        <title>Titulo del portal</title>
        <link rel=”stylesheet” href=”direccion.css”>
        <body>
        <nav>
                <ul>
                        <li>principal</li>
                        <li>fotos</li>
                        <li>videos</li>
                        <li>contacto</li>
                </ul>
            </nav>
            <section>
                <article>
                        Esto es un texto
                <article>
                <article>
                        Esto es otro texto
                <article>
            </section>
            <aside>
                <blockquote>Mensaje número uno</blockquote>
                <blockquote>Mensaje número uno</blockquote>
            </aside>
            <footer>
                Derechos Reservados SENA
            </footer>
            </body>
        </head>
</html>
```

<**header**> El elemento header contiene al encabezado de una sección o documento, donde se colocan habitualmente los encabezados, los vínculos de navegación, los formularios de búsqueda, los logos, las tablas de contenidos, las introducciones, etc. En muchos casos, los contenidos del encabezado de un documento son consistentes a lo largo de todo el sitio.

<h1> es el elemento HTML utilizado, de manera habitual, para identificar la cabecera más importante en una página web - **Titulos**.
<h2> es el elemento HTML utilizado, de manera habitual, para identificar la cabecera más importante en una página web - **subsecciones**.

<h3> es el elemento HTML utilizado, de manera habitual, para identificar la cabecera más importante en una página web - **subapartados de un bloque encabezado**.

Relación de H`s

h1 > h2 > h3 > h4 > h5 > h6

<**hgroup**> Nos permite agurpar cabeceras
Ejemplo:

```
<hgroup>
        <h1>Título del mensaje uno</h1>
        <h2>Subtítulo del mensaje uno</h2>
</hgroup>
```

IMPORTANTE: El elemento <hgroup> es necesario cuando tenemos un título y subtítulo o más etiquetas H juntas en la misma cabecera. Este elemento puede contener solo etiquetas H y esta fue la razón por la que en nuestro ejemplo dejamos los datos adicionales afuera.

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <link rel=”stylesheet” href=”direccion.css”>
            <body>
            <nav>
                    <ul>
                        <li>principal</li>
                        <li>fotos</li>
                        <li>videos</li>
                        <li>contacto</li>
                    </ul>
                </nav>
                <section>
                    <article>
                        Esto es un texto
                    <article>
                    <article>
                        Esto es otro texto
                    <article>

                </section>
                <aside>
                    <header>
                        <hgroup>
                            <h1>Título del mensaje uno</h1>
                            <h2>Subtítulo del mensaje uno</h2>
                        </hgroup>
                    </header>
                        <blockquote>Mensaje número uno</blockquote>
```

```
                    <blockquote>Mensaje número uno</blockquote>
                <footer>
                    Derechos Reservados SENA
                </footer>

            </aside>
            <footer>
                Derechos Reservados SENA
            </footer>
            </body>
        </head>
</html>
```

<figure> y <figcaption> La etiqueta <figure> fue creada para ayudarnos a ser aún más específicos a la hora de declarar el contenido del documento. Antes de que este elemento sea introducido, no podíamos identificar el contenido que era parte de la información pero a la vez independiente, como ilustraciones, fotos, videos, etc... Normalmente estos elementos son parte del contenido relevante pero pueden ser extraídos o movidos a otra parte sin afectar o interrumpir el flujo del documento. Cuando nos encontramos con esta clase de información, las etiquetas <figure> pueden ser usadas para identificarla:

```
<!DOCTYPE html>
<html lang="es">
        <head>
            <meta charset=”UTF-8”>
            <meta name=”description” content =”Descripcion del Sitio”>
            <meta name=”keywords” content =”html5, css3”>
            <title>Titulo del portal</title>
            <link rel=”stylesheet” href=”direccion.css”>
            <body>
            <nav>
                    <ul>
                        <li>principal</li>
                        <li>fotos</li>
                        <li>videos</li>
                        <li>contacto</li>
                    </ul>
                </nav>
                <section>
                    <article>
                        Esto es un texto
                    <article>
                    <article>
                        Esto es otro texto
                    <article>

                </section>
```

```
            <aside>
                <header>
                    <hgroup>
                        <h1>Título del mensaje uno</h1>
                        <h2>Subtítulo del mensaje uno</h2>
                    </hgroup>
                </header>
                <blockquote>Mensaje número uno</blockquote>
                <blockquote>Mensaje número uno</blockquote>
                <figure>
                    <img src="http://sena/content/myimage.jpg">    <figcaption>
                    Esta es la imagen del primer mensaje
                </figcaption>
                </figure>
                <footer>
                    Derechos Reservados SENA
                </footer>

            </aside>
            <footer>
                Derechos Reservados SENA
            </footer>
            </body>
        </head>
</html>
```

Nuevos y viejos elementos

<**mark**> La etiqueta <mark> fue agregada para resaltar parte de un texto que originalmente no era considerado importante pero ahora es relevante acorde con las acciones del usuario. El ejemplo que más se ajusta a este caso es un resultado de búsqueda. El elemento <mark>resaltará la parte del texto que concuerda con el texto buscado:

```
<span>Mi <mark>carro</mark> es rojo</span>
```

Si un usuario realiza una búsqueda de la palabra "carro", por ejemplo, los resultados podrían ser mostrados con el código anterior. La frase del ejemplo representa los resultados de la búsqueda y las etiquetas <mark> en el medio encierran lo que era el texto buscado (la palabra "carro"). En algunos navegadores, esta palabra será resaltada con un fondo amarillo por defecto, pero siempre podemos sobrescribir estos estilos con los nuestros utilizando CSS. En el pasado, normalmente obteníamos similares resultados usando el elemento <b>. El agregado de <mark> tiene el objetivo de cambiar el significado y otorgar un nuevo propósito para éstos y otros elementos relacionados:

- es para indicar énfasis (reemplazando la etiqueta <i> que utilizábamos anteriormente).

- es para indicar importancia.
- <mark> es para resaltar texto que es relevante de acuerdo con las circunstancias.
- <b> debería ser usado solo cuando no hay otro elemento más apropiado para la situación

<small> La nueva especificidad de HTML es también evidente en elementos como *<small>.* Previamente este elemento era utilizado con la intención de presentar cualquier texto con letra pequeña. La palabra clave referenciaba el tamaño del texto, independientemente de su significado. En HTML5, el nuevo propósito de <small> es presentar la llamada letra pequeña, como impresiones legales, descargos, etc...

```
<small>
        Derechos Reservados &copy; 2011 MinkBooks
</small>
```

<cite>Otro elemento que ha cambiado su naturaleza para volverse más específico es <cite>. Ahora las etiquetas <cite> encierran el título de un trabajo, como un libro, una película, una canción, etc... Amo la película <cite>Tentaciones</cite>
<address>El elemento <address> es un viejo elemento convertido en un elemento estructural. No necesitamos usarlo previamente para construir nuestra plantilla, sin embargo podría ubicarse perfectamente en algunas situaciones en las que debemos presentar información de contacto relacionada con el contenido del elemento <article> o el cuerpo completo. Este elemento debería ser incluido dentro de <footer>, como en el siguiente ejemplo:

```
<article>
        <header>
                <h1>
                Título del mensaje
                </h1>
        </header>
        Este es el texto del mensaje
        <footer>
                <address>
                        <a href="http://www.jdgauchat.com">JD Gauchat</a>
                </address>
        </footer>
</article>
```

<time>En cada <article> de nuestra última plantilla (Listado 1-18), incluimos la fecha indicando cuándo el mensaje fue publicado. Para esto usamos un simple elemento <p>dentro de la cabecera (<header>) del mensaje, pero existe un elemento en HTML5 específico para este propósito. El elemento <time> nos permite declarar un texto comprensible para humanos y navegadores que representa fecha y hora:

```
<article>
        <header>
                <h1>
                        Título del mensaje dos
                </h1>
                <time datetime="2011-10-12" pubdate>
                        publicado 12-10-2011
                </time>
        </header>
        Este es el texto del mensaje
</article>
```

Video en HTML5

Intentemos ignorar por un momento estas complicaciones y disfrutar de la simplicidad del elemento <video>. Este elemento ofrece varios atributos para establecer su comportamiento y configuración. Los atributos width y height, al igual que en otros elementos HTML ya conocidos, declaran las dimensiones para el elemento o ventana del reproductor. El tamaño del video será automáticamente ajustado para entrar dentro de estos valores, pero no fueron considerados para redimensionar el video sino limitar el área ocupada por el mismo para mantener consistencia en el diseño. El atributo srcespecifica la fuente del video. Este atributo puede ser reemplazado por el elemento <source> y su propio atributo src para declarar varias fuentes con diferentes formatos, como en el siguiente ejemplo:

```
<video src=”http://sena/content/trailer.mp4” controls> </video>
```

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Video</title>
 </head>
 <body> <section id="reproductor">
<video id="medio" width="720" height="400" controls>
<source src="http://sena/content/trailer.mp4">
<source src="http://sena/content/trailer.ogg">
 </video>
</section>
</body>
</html>
```

Nota: formatos soportados (MP4 u OGG).

Reproducción Con Atributos

```
<!DOCTYPE html>
<html lang="es">
```

```
<head>
<title>Reproductor de Video</title>
 </head>
 <body> <section id="reproductor">
<video id="medio" width="720" height="400" controls preload loop poster=”http://asdfg.com”>
<source src="http://sena/content/trailer.mp4">
<source src="http://sena/content/trailer.ogg">
 </video>
</section>
</body>
</html>
```

Reproducción con Eventos

progress: Este evento es disparado periódicamente para informar acerca del progreso de la descarga del medio. La información estará disponible a través del atributo buffered, como veremos más adelante.
canplaythrough Este evento es disparado cuando el medio completo puede ser reproducido sin interrupción. El estado es establecido considerando la actual tasa de descarga y asumiendo que seguirá siendo la misma durante el resto del proceso. Existe otro evento más para este propósito, canplay, pero no considera toda la situación y es disparado tan pronto como algunas partes del medio se encuentran disponibles (luego de descargar los primeros cuadros de un video, por ejemplo).
ended Es disparado cuando el reproductor llega al final del medio.
pause Es disparado cuando el reproductor es pausado.
play Es disparado cuando el medio comienza a ser reproducido.
error Este evento es disparado cuando ocurre un error. Es relacionado con el elemento <source> correspondiente a la fuente del medio que produjo el error.

Para nuestro reproductor de ejemplo solo vamos a escuchar a los habituales eventos click y load.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Video</title>
 </head>
 <body> <section id="reproductor">
<video id="medio" width="720" height="400" controls preload loop poster=”http://asdfg.com”>
<source src="http://sena/content/trailer.mp4">
<source src="http://sena/content/trailer.ogg">
 </video>
</section>
<sript>
function estado(){ if(!medio.ended){ var total=parseInt(medio.currentTime*maximo/medio.duration);
progreso.style.width=total+'px'; }else{ progreso.style.width='0px'; reproducir.innerHTML='Reproducir';
window.clearInterval(bucle); } }
```

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

Formularios Web

El elemento <form>Los formularios en HTML no han cambiado mucho. La estructura sigue siendo la misma, pero HTML5 ha agregado nuevos elementos, tipos de campo y atributos para expandirlos tanto como sea necesario y proveer así las funciones actualmente implementadas en aplicaciones web.

```
<!DOCTYPE html>
<html lang="es">
 <head>
<title>Formularios</title>
</head>
<body>
<section>
 <form name="miformulario" id="miformulario" method="get">
<input type="text" name="nombre" id="nombre">
 <input type="submit" value="Enviar">
 </form>
 </section>
 </body>
 </html>
```

El Elemento <input>

- Tipo email. `<input type="email" name="miemail" id="miemail">`
- Tipo search `<input type="search" name="busqueda" id="busqueda">`
- Tipo url `<input type="url" name="miurl" id="miurl">`
- Tipo tel `<input type="tel" name="telefono" id="telefono">`
- Tipo number `<input type="number" name="numero" id="numero" min=”0” max=”10” step=”5”>`
- Tipo range `<input type="range" name="numero" id="numero" min=”0” max=”10” step=”5”>`
- Tipo date `<input type="date" name="fecha" id="fecha">`
- Tipo week `<input type="week" name="semana" id="semana">`
- Tipo month `<input type="month" name="mes" id="mes">`
- Tipo time `<input type="time" name="hora" id="hora">`
- Tipo datetime `<input type="datetime" name="fechahora" id="fechahora">`
- Tipo datetime-local `<input type="datetime-local" name="tiempolocal"id="tiempolocal">`
- Tipo color `<input type="color" name="micolor" id="micolor">`
- Tipo datalist `<datalist id="informacion"> <option value=”123123123” label=”Teléfono 1”> <option value=”456456456” label=”Teléfono 2”> </datalist> / <input type="tel" name="telefono" id="telefono" list="informacion">`

- Tipo progress
- Tipo meter
- Tipo output

Nuevos Atributos

- Atributo placeholder <input type="search" name="busqueda" id="busqueda" placeholder="escriba su búsqueda">
- Atributo required <input type="email" name="miemail" id="miemail" required>
- Atributo multiple <input type="email" name="miemail" id="miemail" multiple>
- Atributo autofocus <input type="search" name="busqueda" id="busqueda" autofocus>
- Atributo pattern <input pattern="[0-9]{5}" name="codigopostal" id="codigopostal" title="inserte los 5 números de su código postal">
- Atributo form <input type="search" name="busqueda" id="busqueda" form="formulario">

Nota: nombre1.setCustomValidity('inserte al menos un nombre') – valueMissing - typeMismatch

CSS3 -- Estructura Global

Estilos y estructura A pesar de que cada navegador garantiza estilos por defecto para cada uno de los elementos HTML, estos estilos no necesariamente satisfacen los requerimientos de cada diseñador. Normalmente se encuentran muy distanciados de lo que queremos para nuestros sitios webs. Diseñadores y desarrolladores a menudo deben aplicar sus propios estilos para obtener la organización y el efecto visual que realmente desean.

Elementos block

Con respecto a la estructura, básicamente cada navegador ordena los elementos por defecto de acuerdo a su tipo: block (bloque) o inline (en línea). Esta clasificación está asociada con la forma en que los elementos son mostrados en pantalla.

- Elementos blockson posicionados uno sobre otro hacia abajo en la página.
- Elementos inline son posicionados lado a lado, uno al lado del otro en la misma línea, sin ningún salto de línea a menos que ya no haya más espacio horizontal para ubicarlos.

Modelos de caja

Los navegadores consideran cada elemento HTML como una caja. Una página web es en realidad un grupo de cajas ordenadas siguiendo ciertas reglas. Estas reglas son establecidas por estilos provistos por los navegadores o por los diseñadores usando CSS. CSS tiene un

set predeterminado de propiedades destinados a sobrescribir los estilos provistos por navegadores y obtener la organización deseada. Estas propiedades no son específicas, tienen que ser combinadas para formar reglas que luego serán usadas para agrupar cajas y obtener la correcta disposición en pantalla. La combinación de estas reglas es normalmente llamada modelo o sistema de disposición. Todas estas reglas aplicadas juntas constituyen lo que se llama un modelo de caja. Existe solo un modelo de caja que es considerado estándar estos días, y muchos otros que aún se encuentran en estado experimental. El modelo válido y ampliamente adoptado es el llamado Modelo de Caja Tradicional, el cual ha sido usado desde la primera versión de CSS. Aunque este modelo ha probado ser efectivo, algunos modelos experimentales intentan superar sus deficiencias, pero la falta de consenso sobre el reemplazo más adecuado aún mantiene a este viejo modelo en vigencia y la mayoría de los sitios webs programados en HTML5 lo continúan utilizando.

Estilos en línea

Una de las técnicas más simples para incorporar estilos CSS a un documento HTML es la de asignar los estilos dentro de las etiquetas por medio del atributo style.

```
<!DOCTYPE html>
<html lang="es">
        <head>
                <title>Este es el título del documento</title>
        </head>
        <body>
                <p style=”font-size: 20px”>Mi texto</p>
        </body>
</html>
```

Estilos embebidos

Una mejor alternativa es insertar los estilos en la cabecera del documento y luego usar referencias para afectar los elementos HTML correspondientes:

```
<!DOCTYPE html>
        <html lang="es">
                <head>
                        <title>Este texto es el título del documento</title>
                        <style>
                                p { font-size: 20px }
                        </style>
                </head>
                <body>
                        <p>Mi texto</p>
                </body>
        </html>
```

Archivos externos

Declarar los estilos en la cabecera del documento ahorra espacio y vuelve al código más consistente y actualizable, pero nos requiere hacer una copia de cada grupo de estilos en todos los documentos de nuestro sitio web. La solución es mover todos los estilos a un archivo externo y luego utilizar el elemento <link> para insertar este archivo dentro de cada documento que los necesite. Este método nos permite cambiar los estilos por completo simplemente incluyendo un archivo diferente. También nos permite modificar o adaptar nuestros documentos a cada circunstancia o dispositivo.

```
<!DOCTYPE html>
<html lang="es">
        <head>
                <title>Este texto es el título del documento</title>
                <link rel="stylesheet" href="misestilos.css">
        </head>
        <body>
                <p>Mi texto</p>
        </body>
</html>
```

Referenciando con palabra clave Al declarar las reglas CSS utilizando la palabra clave del elemento afectamos cada elemento de la misma clase en el documento. Por ejemplo, la siguiente regla cambiará los estilos de todos los elementos <p>:

```
p { font-size: 20px }
span { font-size: 20px }
```

Referenciando con el atributo id El atributo id es como un nombre que identifica al elemento. Esto significa que el valor de este atributo no puede ser duplicado. Este nombre debe ser único en todo el documento. Para referenciar un elemento en particular usando el atributo id desde nuestro archivo CSS la regla debe ser declarada con el símbolo # al frente del valor que usamos para identificar el elemento:

```
#texto1 { font-size: 20px }
```

Referenciando con el atributo class La mayoría del tiempo, en lugar de utilizar el atributo id para propósitos de estilos es mejor utilizar class. Este atributo es más flexible y puede ser asignado a cada elemento HTML en el documento que comparte un diseño similar:

```
.texto1 { font-size: 20px }
```

Referenciando con cualquier atributo Aunque los métodos de referencia estudiados anteriormente cubren un variado espectro de situaciones, a veces no son suficientes para encontrar el elemento exacto. La última versión de CSS ha incorporado nuevas formas de referenciar elementos HTML. Uno de ellas es el Selector de Atributo. Ahora podemos

referenciar un elemento no solo por los atributos id y class sino también a través de cualquier otro atributo:

```
p[name] { font-size: 20px }
```

Para imitar lo que hicimos previamente con los atributos id y class, podemos también especificar el valor del atributo:

```
p[name="mitexto"] { font-size: 20px }
```

CSS3 permite combinar "=" con otros para hacer una selección más específica:

```
p[name^="mi"] { font-size: 20px }
p[name$="mi"] { font-size: 20px }
p[name*="mi"] { font-size: 20px }
```

Expresiones Regulares

- La regla con el selector ^= será asignada a todo elemento <p> que contiene un atributo name con un valor comenzado en "mi" (por ejemplo, "mitexto", "micasa").
- La regla con el selector $= será asignada a todo elemento <p> que contiene un atributo name con un valor finalizado en "mi" (por ejemplo "textomi", "casami").
- La regla con el selector *= será asignada a todo elemento <p> que contiene un atributo name con un valor que incluye el texto "mi" (en este caso, el texto podría también encontrarse en el medio, como en "textomicasa").

Referenciando con pseudo clases

```
p:nth-child(2){
        background: #999999;
}
```

La pseudo clase es agregada usando dos puntos luego de la referencia y antes del su nombre. Referenciamos solo elementos <p>. Esta regla puede incluir otras referencias. Por ejemplo, podríamos escribirla como .miclase:nth-child(2) para referenciar todo elemento que es hijo de otro elemento y tiene el valor de su atributo class igual a miclase. La pseudo clase puede ser aplicada a cualquier tipo de referencia estudiada previamente. La pseudo clase nth-child() nos permite encontrar un hijo específico. Hay cuatro elementos <p> que son hermanos. Esto significa que todos ellos tienen el mismo padre que es el elemento . Lo que esta pseudo clase está realmente indicando es algo como: "el hijo en la posición..." por lo que el número entre paréntesis será el número de la posición del hijo, o índice.

asignar estilos a todos los elementos creando una regla para cada uno de ellos:

```
*{ margin: 0px; }
```

** **Leer más de:** first-child, last-child, only-child y not

Nuevos selectores

Hay algunos selectores más que fueron agregados o que ahora son considerados parte de CSS3 y pueden ser útiles para nuestros diseños. Estos selectores usan los símbolos **>**, **+** y **~** para especificar la relación entre elementos.

```
div > p.mitexto2{ color: #990000; }
```

El selector > está indicando que el elemento a ser afectado por la regla es el elemento de la derecha cuando tiene al de la izquierda como su padre. El código anterior modifica los elementos <p> que son hijos de un elemento . En este caso, fuimos bien específicos y referenciamos solamente el elemento<p> con el valor mitexto2 en su atributo class.

El próximo ejemplo construye un selector utilizando el símbolo **+**. Este selector referencia al elemento de la derecha cuando es inmediatamente precedido por el de la izquierda. Ambos elementos deben compartir el mismo padre`: p.mitexto2 + p{ color: #990000; }`

El código anterior afecta al elemento <p> que se encuentra ubicado luego de otro elemento <p> identificado con el valor mitexto2 en su atributo class

El último selector que estudiaremos es el construido con el símbolo **~.** Este selector es similar al anterior pero el elemento afectado no necesita estar precediendo de inmediato al elemento de la izquierda. Además, más de un elemento puede ser afectado:

```
p.mitexto2 ~ p{ color: #990000; }
```

Alineación:

Alinear al Centro: `text-align: center;`
Alinear a la Izquierda: `text-align: left;`
Alinear a la Derecha: `text-align: right;`
Alinear Justificado: `text-align: justify;`

Trabajar tamaños de elementos:

Ancho**:** `width`
Alto: `height`

propiedades de borde, margen y relleno

Propiedad de Padding (relleno)

Las propiedades de relleno de CSS definen el espacio entre el elemento de borde y el elemento de contenido. Se puede definir el valor de relleno de la siguiente manera:

- padding-top: 10px;
- padding-right: 10px;

- padding-bottom: 10px;
- padding-left: 10px;

También puede utilizar la propiedad:

- padding:25px 50px 75px 100px;
- padding superior es 25px
- padding derecho es 50px
- padding bajo es 75px
- padding izquierdo es 100px

padding:25px 50px 75px;
padding superior es 25px
padding derecho y padding izquierdo son 50px
padding bajo es 75px

padding:25px 50px;
padding superior y padding bajo son 25px
padding derecho y padding izquierdo son 50px

padding:25px;
todos paddings son 25px
Nota: el valor de relleno es añadido a la anchura del elemento y es afectado por el fondo del elemento.
En otras palabras, tenemos el elemento div con la clase de div-1:

```
div.div-1{
width:150px;
padding: 25px;}
```

La anchura del navegador agrega los valores de relleno izquierdo y derecho al ancho del cuadro. Como resultado, Usted verá el cuadro de 200 píxeles de ancho.

Propiedad de Border (borde)
Las propiedades de borde de CSS le permiten especificar el estilo y el color del borde del elemento.
border-width (anchura de borde)
La propiedad border-width es utilizada para configurar el ancho de borde. El ancho es especificado en píxeles, o por medio de uno de los tres valores predefinidos: thin, medium, o thick.
border-color (color de borde)
La propiedad border-width es utilizada para configurar el color de borde. Se puede configurar color por:

- name – specify a color name, like “red”

- RGB – specify a RGB value, like “rgb(255,0,0)”
- Hex – specify a hex value, like “#ff0000”

border-style (estilo de borde)

- **dotted**: Define un borde punteado
- **dashed**: Define un borde de rayas
- **solid**: Define un borde sólido
- **double**: Define dos bordes. El ancho de dos bordes es el mismo que el valor border-width
- **groove**: Define un borde estriado (grooved) 3D. El efecto depende del valor de border-color
- **ridge**: Define un borde puntiagudo (ridged) 3D. El efecto depende del valor de border-color
- **inset**: Define un borde de inset 3D. El efecto depende del valor de border-color
- **outset**: Define un borde de outset 3D. El efecto depende del valor de border-color

Con la ayuda de la propiedad shortand puede definir el borde del elemento de la siguiente manera:

```
div.div-2{
        border:1px solid #ccc;
}
```

Propiedad de Margin (margen)

Las propiedades de margen de CSS define el espacio alrededor de elementos. El margen limpia un área alrededor de un elemento (fuera de borde). El margen no tiene color de fondo, y es completamente transparente.

Se puede definir el margen de valores de elementos de la siguiente manera:

- margin-top:100px;
- margin-bottom:100px;
- margin-right:50px;
- margin-left:50px;

También puede utilizar la propiedad shortand:

- **margin:25px 50px 75px 100px;**
  - margen superior es 25px
  - margen derecho es 50px
  - margen inferior (bajo) es 75px
  - margen izquierdo es 100px
- **margin:25px 50px 75px;**
  - margen superior es 25px
  - margen derecho y margen izquierdo son 50px
  - margen inferior (bajo) es 75px
- **margin:25px 50px;**
  - margen superior y margen inferior (bajo) son 25px

- margen derecho y margen izquierdo son 50px

- **margin:25px;**
  - cuatro márgenes son 25px

Usando el valor auto de la propiedad de margen se puede centrar el bloque horizontal.

```
div.div-3{
            margin: 0 auto;
  }
```

Ejercicios Prácticos

Ejercicio 1.

```
div {
width: 100px;
margin: 20px;
padding: 10px;
border: 1px solid #000000;

-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
box-sizing: border-box; }
```

Ejercicio 2.

```
#principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-border-radius: 20px; -webkit-border-radius: 20px; border-radius: 20px; } #titulo { font: bold 36px verdana, sans-serif; }
```

Ejercicio 3.

```
body { text-align: center; } #principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-border-radius: 20px 10px 30px 50px; -webkit-border-radius: 20px 10px 30px 50px; border-radius: 20px 10px 30px 50px; } #titulo { font: bold 36px verdana, sans-serif; }
```

Ejercicio 4.

```
body { text-align: center; } #principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-border-radius: 20px / 10px; -webkit-border-radius: 20px / 10px; border-radius: 20px / 10px; } #titulo { font: bold 36px verdana, sans-serif; }
```

Ejercicio 5.

```
body { text-align: center; } #principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-border-radius: 20px; -webkit-border-radius: 20px; border-radius: 20px; -moz-box-shadow: rgb(150,150,150) 5px 5px; -webkit-box-shadow: rgb(150,150,150) 5px 5px; box-shadow: rgb(150,150,150) 5px 5px; } #titulo { font: bold 36px verdana, sans-serif; }
```

Ejercicio 6.

```
body { text-align: center; } #principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-border-radius: 20px; -webkit-border-radius: 20px; border-radius: 20px; -moz-box-shadow: rgb(150,150,150) 5px 5px 10px; -webkit-box-shadow: rgb(150,150,150) 5px 5px 10px; box-shadow: rgb(150,150,150) 5px 5px 10px; } #titulo { font: bold 36px verdana, sans-serif; text-shadow: rgb(0,0,150) 3px 3px 5px; }
```

@font-face

Obtener un texto con sombra es realmente un muy buen truco de diseño, imposible de lograr con métodos previos, pero más que cambiar el texto en sí mismo solo provee un efecto tridimensional. Una sombra, en este caso, es como pintar un viejo coche, al final será el mismo coche. En este caso, será el mismo tipo de letra. El problema con las fuentes o tipos de letra es tan viejo como la web. Usuarios regulares de la web a menudo tienen un número limitado de fuentes instaladas en sus ordenadores, usualmente estas fuentes son diferentes de un usuario a otro, y la mayoría de las veces muchos usuarios tendrán fuentes que otros no. Por años, los sitios webs solo pudieron utilizar un limitado grupo de fuentes confiables (un grupo básico que prácticamente todos los usuarios tienen instalados) y así presentar la información en pantalla. La propiedad **@font-face** permite a los diseñadores proveer un archivo conteniendo una fuente específica para mostrar sus textos en la página. Ahora podemos incluir cualquier fuente que necesitemos con solo proveer el archivo adecuado:

```
body
{
text-align: center;
}
 #principal
{
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
```

```
-moz-border-radius: 20px;
-webkit-border-radius: 20px;
border-radius: 20px;
-moz-box-shadow: rgb(150,150,150) 5px 5px 10px;
-webkit-box-shadow: rgb(150,150,150) 5px 5px 10px; box-shadow: rgb(150,150,150) 5px 5px 10px;
 }
#titulo
{
font: bold 36px MiNuevaFuente, verdana, sans-serif;
text-shadow: rgb(0,0,150) 3px 3px 5px;
}
@font-face
{
 font-family: 'MiNuevaFuente';
src: url('font.ttf');
 }
```

--Nota: Consultar frente a los atributos -moz-... y -webkit...

Transform y transition

Los elementos HTML, cuando son creados, son como bloques sólidos e inamovibles. Pueden ser movidos usando código Javascript o aprovechando librerías populares como jQuery (www.jquery.com), por ejemplo, pero no existía un procedimiento estándar para este propósito hasta que CSS3 presentó las propiedades **transform y transition**. Ahora ya no tenemos que pensar en cómo hacerlo. En su lugar, solo tenemos que conocer cómo ajustar unos pocos parámetros y nuestro sitio web puede ser tan flexible y dinámico como lo imaginamos. La propiedad **transform** puede operar cuatro transformaciones básicas en un elemento: scale (escalar), rotate (rotar), skew (inclinar) y translate (trasladar o mover). Cómo funcionan:

Transform: scale

```
#principal {
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
-moz-transform: scale(2);
-webkit-transform: scale(2);
}
```

Ejemplo 2

```
#principal {
display: block;
width: 500px;
```

```
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
-moz-transform: scale(1,-1);
-webkit-transform: scale(1,-1);
}
```

Transform: rotate La función rotate rota el elemento en la dirección de las agujas de un reloj. El valor debe ser especificado en grados usando la unidad “deg”

```
#principal {
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
-moz-transform: rotate(30deg);
-webkit-transform: rotate(30deg);
}
```

Transform: skew Esta función cambia la simetría del elemento en grados y en ambas dimensiones.

```
#principal {
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
 -moz-transform: skew(20deg);
-webkit-transform: skew(20deg);
}
```

Transform: translate Similar a las viejas propiedades top y left, la función translate mueve o desplaza el elemento en la pantalla a una nueva posición.

```
#principal {
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;
-moz-transform: translate(100px);
```

```
 -webkit-transform: translate(100px);
}
```

Ejercicio Práctico

```
#principal { display: block; width: 500px; margin: 50px auto; padding: 15px; text-align: center; border: 1px solid #999999; background: #DDDDDD; -moz-transform: translateY(100px) rotate(45deg) scaleX(0.3); -webkit-transform: translateY(100px) rotate(45deg) scaleX(0.3); }
```

Transiciones

De ahora en más, hermosos efectos usando transformaciones dinámicas son accesibles y fáciles de implementar. Sin embargo, una animación real requiere de un proceso de más de dos pasos. La propiedad transition fue incluida para suavizar los cambios,

```
#principal {
display: block;
width: 500px;
margin: 50px auto;
padding: 15px;
text-align: center;
border: 1px solid #999999;
background: #DDDDDD;

-moz-transition:
-moz-transform 1s ease-in-out 0.5s;
-webkit-transition: -webkit-transform 1s ease-in-out 0.5s; }
#principal:hover{
 -moz-transform: rotate(5deg);
-webkit-transform: rotate(5deg);
}
```

Ejercicios Prácticos BACKGROUND

Ejercicio Práctico 1

```
background: lightblue url("img_tree.gif") no-repeat fixed center;
body {
  background-image: url("img_tree.gif");
  background-repeat: no-repeat;
  background-attachment: fixed;
}
```

Ejercicio Práctico 2

```
div {
  background-repeat: no-repeat, repeat;
  background-image: url("img_tree.gif"), url("paper.gif");
```

```
 background-blend-mode: lighten;
}
```

Ejercicio Práctico 3

```
div {
 border: 10px dotted black;
 padding: 15px;
 background: lightblue;
 background-clip: padding-box;
}
```

Ejercicio Práctico 4

```
body {background-color: coral;}
```

Ejercicio Práctico 5

```
body {background-color: #92a8d1;}
```

Ejercicio Práctico 6

```
body {background-color: rgb(201, 76, 76);}
```

Ejercicio Práctico 7

```
body {background-color: rgba(201, 76, 76, 0.3);}
```

Ejercicio Práctico 8

```
body {background-color: hsl(89, 43%, 51%);}
```

Ejercicio Práctico 9

```
body {background-color: hsla(89, 43%, 51%, 0.3);}
```

Ejercicio Práctico 10

```
body {
 background-image: url("paper.gif");
 background-color: #cccccc;
}
```

Ejercicio Práctico 11

```
body {
 background-image: url("img_tree.gif"), url("paper.gif");
 background-color: #cccccc;
}
```

Ejercicio Práctico 12

```
#example1 {
 border: 10px dashed black;
 padding: 25px;
 background: url(paper.gif);
 background-repeat: no-repeat;
 background-origin: content-box;
}
```

Ejercicio Práctico 13

```
body {
 background-image: url('w3css.gif');
 background-repeat: no-repeat;
 background-attachment: fixed;
 background-position: center;
}
```

Ejercicio Práctico 14

```
body {
  background-image: url("paper.gif");
  background-repeat: repeat-y;
}
```

Ejercicio Práctico 15

```
#example1 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: auto;
}

#example2 {
  background: url(mountain.jpg);
  background-repeat: no-repeat;
  background-size: 300px 100px;
}
```

JAVASCRIPT -- Estructura Global

JavaScript es creado por Brendan Eich y vio la luz en el año 1995 con el nombre de LiveScript, que luego fue nombrado JavaScript, nace como un lenguaje sencillo destinado a añadir algunas características interactivas a las páginas web. Sin embargo, hoy en día ha crecido de manera acelerada y es el lenguaje de programación que se utiliza en casi todos los sitios web en el mundo.

El poder de JavaScript está disponible principalmente en lado frontend, agregando mayor interactividad a la web, también puedes usar las librerías y framework como: jquery, angular, backbone, react y demás, escritas sobre JavaScript, y que te ayudan a crear una mejor experiencia de usuario en nuestros sitios web. De igual manera JavaScript se puede utilizar en los servidores web. Node.JS es tu mejor opción para usar este lenguaje del lado del servidor.

Actualmente existen tres métodos disponibles para **referenciar** elementos HTML desde Javascript:
•getElementsByTagName referencia un elemento por su nombre o palabra clave.
•getElementById referencia un elemento por el valor de su atributo id.
•getElementsByClassName es una nueva incorporación que nos permite referenciar un elemento por el valor de su atributo class.

Nuevos Selectores

Los elementos HTML tienen que ser referenciados desde Javascript para ser afectados por el código, CSS, y especialmente CSS3, ofrece un poderoso sistema de referencia y selección que no tiene comparación con los pocos métodos provistos por Javascript para este propósito. Los métodos **getElementById**, **getElementsByTagName** y **getElementsByClassName** no son suficientes para contribuir a la integración que este lenguaje necesita y sostener la relevancia que posee dentro de la especificación de HTML5. Para elevar Javascript al nivel que las circunstancias requieren, nuevas alternativas debieron ser incorporadas. Desde ahora podemos seleccionar elementos HTML aplicando toda clase de selectores CSS por medio de los nuevos métodos **querySelector()** y **querySelectorAll().** **querySelector()** Este método retorna el primer elemento que concuerda con el grupo de selectores especificados entre paréntesis. Los selectores son declarados usando comillas y la misma sintaxis CSS, como en el siguiente ejemplo:

```
function hacerclic()
{
document.querySelector(“#principal p:first- child”).onclick=mostraralerta;
}
function mostraralerta()
{
alert('hizo clic!');
}
window.onload=hacerclic;
```

querySelectorAll()

En lugar de uno, el método querySelectorAll() retorna todos los elementos que concuerdan con el grupo de selectores declarados entre paréntesis. El valor retornado es un arreglo (array) conteniendo cada elemento encontrado en el orden en el que aparecen en el documento.

```
function hacerclic(){
var lista=document.querySelectorAll(“#principal p”);
lista[0].onclick=mostraralerta;
}
function mostraralerta()
{
alert('hizo clic!');
}
window.onload=hacerclic;
```

Manejadores de eventos

Javascript es normalmente ejecutado luego de que el usuario realiza alguna acción. Estas acciones y otros eventos son procesados por manejadores de eventos y funciones Javascript asociadas con ellos.

Existen tres diferentes formas de registrar un evento para un elemento HTML: podemos agregar un nuevo atributo al elemento, registrar un manejador de evento como una propiedad del elemento o usar el nuevo método estándar **addEventListener().**

Conceptos básicos: En Javascript las acciones de los usuarios son llamadas eventos. Cuando el usuario realiza una acción, como un clic del ratón o la presión de una tecla, un evento específico para cada acción y cada elemento es disparado. Además de los eventos producidos por los usuarios existen también otros eventos disparados por el sistema (por ejemplo, el evento load que se dispara cuando el documento es completamente cargado). Estos eventos son manejados por códigos o funciones. El código que responde al evento es llamado manejador. Cuando registramos un manejador lo que hacemos es definir cómo nuestra aplicación responderá a un evento en particular. Luego de la estandarización del método **addEventListener(),** este procedimiento es usualmente llamado "escuchar al evento", y lo que hacemos para preparar el código que responderá a ese evento es "agregar una escucha" a un elemento en particular.

El método addEventListener()

El método **addEventListener()** es la técnica ideal y la que es considerada como estándar por la especificación de HTML5. Este método tiene tres argumentos: el nombre del evento, la función a ser ejecutada y un valor booleano (falso o verdadero) que indica cómo un evento será disparado en elementos superpuestos.

```
<!DOCTYPE html>
<html lang="es">
<head>
	<title>Este texto es el título del documento</title>
	<script>
		function mostraralerta()
		{
			alert('hizo clic!');
		}
		Function hacerclic()
		{
			var elemento=document.getElementsByTagName('p')[0];
			 elemento.addEventListener('click', mostraralerta, false);
		}
		window.addEventListener('load', hacerclic, false);
	</script>
	</head>
	<body>
		<div id="principal">
			<p>Hacer Clic</p>
			<p>No puede hacer Clic</p>
		</div>
	</body>
</html>
```

APIs Canvas

Canvas es una API gráfica que provee una básica pero poderosa superficie de dibujo. Esta es la más maravillosa y prometedora API de todas. La posibilidad de generar e imprimir gráficos en pantalla, crear animaciones o manipular imágenes y videos (combinado con la funcionalidad restante de HTML5) abre las puertas para lo que nos podamos imaginar. Canvas genera una imagen con pixeles que son creados y manipulados por funciones y métodos provistos específicamente para este propósito.

Drag and Drop

Drag and Drop incorpora la posibilidad de arrastrar y soltar elementos en la pantalla como lo haríamos comúnmente en aplicaciones de escritorio. Ahora, con unas pocas líneas de código, podemos hacer que un elemento esté disponible para ser arrastrado y soltado dentro de otro elemento en la pantalla. Estos elementos pueden incluir no solo gráficos sino además textos, enlaces, archivos o datos en general.

Geolocation

Geolocation es utilizada para establecer la ubicación física del dispositivo usado para acceder a la aplicación. Existen varios métodos para acceder a esta información, desde señales de red hasta el Sistema de Posicionamiento Global (GPS). Los valores retornados incluyen latitud y longitud, posibilitando la integración de esta API con otras como Google Maps, por ejemplo, o acceder a información de localización específica para la construcción de aplicaciones prácticas que trabajen en tiempo real.

Storage

Dos APIs fueron creadas con propósitos de almacenamiento de datos: Web Storage e Indexed Database. Básicamente, estas APIs transfieren la responsabilidad por el almacenamiento de datos del servidor al ordenador del usuario, pero en el caso de Web Storage y su atributo sessionStorage, esta incorporación también incrementa el nivel de control y la eficiencia de las aplicaciones web. Web Storage contiene dos importantes atributos que son a veces considerados APIs por sí mismos: sessionStorage y localStorage.

File

Bajo el título de File, HTML5 ofrece varias APIs destinadas a operar con archivos. En este momento existen tres disponibles: File, File: Directories & System, y File: Writer. Gracias a este grupo de APIs, ahora podemos crear y procesar archivos en el ordenador del usuario.

Communication

Algunas API tienen un denominador común que nos permite agruparlas juntas. Este es el caso para XMLHttpRequest Level 2, Cross Document Messaging, y Web Sockets. Internet ha

estado siempre relacionado con comunicaciones, por supuesto, pero algunos asuntos no resueltos hacían el proceso complicado y en ocasiones imposible. Tres problemas específicos fueron abordados en HTML5: la API utilizada para la creación de aplicaciones Ajax no estaba completa y era complicada de implementar a través de distintos navegadores, la comunicación entre aplicaciones no relacionadas era no existía, y no había forma de establecer una comunicación bidireccional efectiva para acceder a información en el servidor en tiempo real.

El primer problema fue resuelto con el desarrollo de XMLHttpRequest Level 2. XMLHttpRequest fue la API usada por mucho tiempo para crear aplicaciones Ajax, códigos que acceden al servidor sin recargar la página web. El nivel 2 de esta API incorpora nuevos eventos, provee más funcionalidad (con eventos que permiten hacer un seguimiento del proceso), portabilidad (la API es ahora estándar), y accesibilidad (usando solicitudes cruzadas, desde un dominio a otro). La solución para el segundo problema fue la creación de Cross Document Messaging. Esta API ayuda a los desarrolladores a superar las limitaciones existentes para comunicar diferentes cuadros y ventanas entre sí. Ahora una comunicación segura a través de diferentes aplicaciones es ofrecida por esta API utilizando mensajes. La solución para el último de los problemas listados anteriormente es Web Sockets. Su propósito es proveer las herramientas necesarias para la creación de aplicaciones de red que trabajan en tiempo real (por ejemplo, salas de chat). La API les permite a las aplicaciones obtener y enviar información al servidor en períodos cortos de tiempo, volviendo posible las aplicaciones en tiempo real para la web.

Web Workers

Esta es una API única que expande Javascript a un nuevo nivel. Este lenguaje no es un lenguaje multitarea, lo que significa que solo puede hacerse cargo de una sola tarea a la vez. Web Workers provee la posibilidad de procesar código detrás de escena (ejecutado aparte del resto), sin interferir con la actividad en la página web y del código principal. Gracias a esta API Javascript ahora puede ejecutar múltiples tareas al mismo tiempo.

History

Ajax cambió la forma en la que los usuarios interactúan con sitios y aplicaciones web. Y los navegadores no estaban preparados para esta situación. History fue implementada para adaptar las aplicaciones modernas a la forma en que los navegadores hacen seguimiento de la actividad del usuario. Esta API incorpora técnicas para generar artificialmente URLs por cada paso en el proceso, ofreciendo la posibilidad de retorna a estados previos de la aplicación utilizando procedimientos estándar de navegación.

Offline

Incluso hoy día, con acceso a Internet en cada lugar que vamos, quedar desconectado es aún posible. Dispositivos portátiles se encuentran en todas partes, pero no la señal para establecer comunicación. Y los ordenadores de escritorio también pueden dejarnos

desconectados en los momentos más críticos. Con la combinación de atributos HTML, eventos controlados por Javascript y archivos de texto, Offline permitirá a las aplicaciones trabajar en línea o desconectados, de acuerdo a la situación del usuario.