USAR INTERNA SOLAMENTE

SOPRA

Curso de Maquetación

HTML5

Versión 1.00 del miércoles, 03 de agosto de 2016

Estado : Trabajo

Índice

1. Herramientas a utilizar en el curso

Navegador **Firefox** con los plugins:

- Colorzilla
- Pixel Perfect
- Firebug
- Greasemonkey
- IE Tab V2
- Firesizer
- Status-4-Evar
- Senseo

Navegador **Chrome** con plugins:

- Colorzilla
- Window Resizer
- IE Tab
- Perfect pixel
- Meta SEO Inspector

2. Qué es HTML5

HTML5 es un **lenguaje markup** (de hecho, las siglas de HTML significan Hyper Text Markup Language) usado para **estructurar y presentar el contenido para la web**. Es uno de los aspectos fundamentales para el funcionamiento de los sitios, pero no es el primero. Es de hecho la quinta revisión del estándar que fue creado en 1990. A fines del año 2012, la W3C la recomendó para transformarse en el estándar a ser usado en el desarrollo de proyectos venideros. Por así decirlo, qué es HTML5 está relacionado también con la entrada en decadencia del viejo estándar HTML 4, que se combinaba con otros lenguajes para producir los sitios que podemos ver hoy en día. Con HTML5, tenemos otras posibilidades para explotar **usando menos recursos**. Con HTML5, también entra en desuso el formato XHTML, dado que ya no sería necesaria su implementación.

HTML4 fue "declarado" el lenguaje oficial de la web en el año 2000, y tomó una década para comenzar a implementar el desarrollo de su nueva revisión. Esta nueva generación de HTML, se dice, pronto dominará el desarrollo en internet, pero introduce algunos cambios importantes que veremos dentro de algunas líneas.

Pero… ¿Qué es HTML5?... Se trata de un sistema para formatear el *layout* de nuestras páginas, así como hacer algunos ajustes a su aspecto. Con HTML5, los navegadores como Firefox, Chrome, Explorer, Safari y más pueden saber cómo mostrar una determinada página web, saber dónde están los elementos, dónde poner las imágenes, dónde ubicar el texto. En este sentido, el HTML5 no se diferencia demasiado de su predecesor, sin embargo, el nivel de sofisticación del código que podremos construir usando HTML5 es mucho mayor.

3. Semántica HTML5

3.1. QUE ES

Una de las novedades de HTML5, como muchos sabrán, es las etiquetas semánticas que pueden ser utilizadas en nuestras páginas estáticas. Estas nuevas etiquetas se podrían clasificar en dos grupos:

- Etiquetas que extienden a las actuales, como <video>, <audio> o <canvas>, y que además añaden nuevas funcionalidades a los documentos HTML, que se pueden controlar desde JavaScript.
- Etiquetas que componen la web semántica, es decir, que no proponen nuevas funcionalidades pero sirven para estructurar sitios web, y añadir un significado concreto. Por ejemplo, veremos cómo crear una estructura que utiliza las etiquetas estructurales como <nav>, <header>, <footer>, <aside>, <section>, o <article>.

3.2. CABECERA DE UN DOCUMENTO

Además de las nuevas etiquetas introducidas por HTML5 (que veremos más adelante), el nuevo estándar propone pequeñas mejoras que podemos aplicar en la definición de nuestros documentos, en concreto en la cabecera de los mismos.

3.3. DOCTYPE

El estándar XHTML deriva de XML, por lo que comparte con él muchas de sus normas y sintaxis. Uno de los conceptos fundamentales de XML es la utilización del DTD o Document Type Definition ("Definición del Tipo de Documento"). El estándar XHTML define el DTD que deben seguir las páginas y documentos XHTML. En este documento se definen las etiquetas que se pueden utilizar, los atributos de cada etiqueta y el tipo de valores que puede tener cada atributo.

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Ésta es una de las 15 declaraciones posibles declaradas en los estándares HTML4 y XHTML. En HTML5 se reduce la definición del tipo de documento a una única posibilidad, por lo que no tenemos que preocuparnos de elegir el tipo de documento correcto:

```
<!DOCTYPE html>
```

3.4. ELEMENTO RAÍZ HTML

En todo documento HTML, su elemento raíz o nodo superior siempre es la etiqueta <html>. Hasta ahora, este elemento raíz se definía de la siguiente manera:

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

No hay ningún problema en mantener esta sintaxis. Si se desea, se puede conservar, ya que es válido en HTML5. Sin embargo, algunas de sus partes ya no son necesarias, por lo que podemos eliminarlas.

El primer elemento del que podemos prescindir es el atributo xmlns. Se trata de una herencia de XHTML 1.0, que dice que los elementos de esta página están en el espacio de nombres XHTML, http://www.w3.org/1999/xhtml. Sin embargo, los elementos de HTML5 están siempre en este espacio de nombres, por lo que ya no es necesario declararlo explícitamente. Eliminar el atributo xmlns nos deja con este elemento de la siguiente manera:

```
<html lang="es" xml:lang="en">
```

En este caso ocurre lo mismo con el atributo xml:lang, es una herencia de XHTML que podemos eliminar, quedando finalmente la etiqueta de la siguiente manera:

```
<html lang="en">
```

3.5. ELEMENTO HEAD

El primer hijo del elemento raíz es generalmente el elemento head. El elemento head contiene los metadatos que aportan información extra sobre la página, como su título, descripción, autor, etc. Además, puede incluir referencias externas a contenidos necesarios para que el documento se muestre y comporte de manera correcta (como hojas de estilos o scripts). Este elemento ha sufrido pequeñas variaciones, pero que debemos tener en cuenta:

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Primera página</title>
    <link rel="stylesheet" type="text/css" href="styles.css" />
    <link rel="alternate" type="application/atom+xml" title="Primera página RSS Feed" href="/feed/" />
    <link rel="search" type="application/opensearchdescription+xml" href="opensearch.xml "/>
    <link rel="shortcut icon" href="/favicon.ico" />
</head>
```

3.5.1. Codificación del contenido

Cuando se piensa en "texto", probablemente nos venga a la cabeza una definición de "caracteres y símbolos que veo en la pantalla de mi ordenador". Pero realmente se tratan de bits y bytes. Cada cadena de caracteres que se muestra en la pantalla, se almacena con una codificación de caracteres en particular. Hay cientos de codificaciones de caracteres diferentes, algunos optimizado para ciertos idiomas como el ruso, el chino o inglés, y otros que se pueden utilizar para múltiples idiomas. En

términos generales, la codificación de caracteres proporciona una correspondencia entre lo que se muestra en la pantalla y lo que un equipo realmente almacena en la memoria y en el disco.

Se puede pensar en la codificación de caracteres como una especie de clave de descifrado del texto. Cuando accedemos a una secuencia de bytes, y decidimos que es "texto", lo que necesitamos saber es qué codificación de caracteres se utiliza para que pueda decodificar los bytes en caracteres y mostrarlos (o transformarlos) de manera correcta.

Lo ideal es establecer esta codificación en el servidor, indicando el tipo en las cabeceras de respuesta:

```
Content-Type: text/html; charset="utf-8"
```

Por desgracia, no siempre podemos tener el control sobre la configuración de un servidor HTTP. Por ejemplo, en la plataforma Blogger, el contenido es proporcionado por personas, pero los servidores son administrados por Google, por lo que estamos supeditados a su configuración. Aún así, HTML 4 proporciona una manera de especificar la codificación de caracteres en el documento HTML:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

El encabezado HTTP es el método principal, y anula la etiqueta <meta> si está presente. Pero no todo el mundo puede establecer encabezados HTTP, por lo que la etiqueta <meta> todavía tiene sentido. En el caso de HTML5, es aún más sencillo definir esta etiqueta meta:

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

Debemos acostumbrarnos a especificar la codificación de nuestros documentos, aunque no vayamos a utilizar caracteres especiales o nuestro documentos no se presente en otros idiomas. Si no lo hacemos, podemos exponernos a problemas de seguridad.

3.5.2. Links

Dentro del elemento head, las etiquetas <link> son una manera de acceder o declarar contenido externo al documento actual, que puede cumplir distintos objetivos:

- Una hoja de estilo contiene las reglas CSS que su navegador debe aplicar al presente documento.
- Un feed que contiene el mismo contenido que esta página, pero en un formato estándar (RSS).
- Una traducción de esta página en otro idioma.
- Contenido en formato PDF.
- Siguiente capítulo de un libro en línea de la cual esta página es también una parte.

En HTML5, se separan estas relaciones de enlace en dos categorías:

- Enlaces a recursos externos que se van a utilizar para mejorar el documento actual.
- Enlaces de hipervínculos que son enlaces a otros documentos.

El tipo de relación más utilizado (literalmente) es el siguiente:

```
<link rel="stylesheet" href="style-original.css" type="text/css" />
```

Esta relación es utilizada para indicar el archivo donde se almacenan las reglas CSS que se desean aplicar al documento. Una pequeña optimización que se puede hacer en HTML5 es eliminar el atributo type. Sólo hay un lenguaje de estilo para la web, CSS, así que ese es el valor predeterminado para el atributo type:

```
<link rel="stylesheet" href="style-original.css" />
```

3.6. NUEVAS ETIQUETAS SEMÁNTICAS

En 2004, Ian Hickson, el autor de la especificación de HTML5, analizó 1.000.000.000 de páginas web utilizando el motor de Google, intentando identificar la manera en la que la web real estaba construida. Uno de los resultados de este análisis, fue la publicación de una lista con los nombres de clases más utilizados. Este estudio revela que los desarrolladores utilizan clases o IDs comunes para estructurar los documentos. Esto llevó a considerar que quizás fuese una buena idea crear etiquetas concretas para reflejar estas estructuras.

Este tipo de etiquetas que componen la web semántica nos sirven para que cualquier mecanismo automático (un navegador, un motor de búsqueda, un lector de feeds...) que lea un sitio web sepa con exactitud qué partes de su contenido corresponden a cada una de las partes típicas de un sitio. Observando esas etiquetas semánticas estructurales, cualquier sistema podrá procesar la página y saber cómo está estructurada. Veamos algunas de estas etiquetas que introduce HTML5 en este sentido.

- SECTION: se utiliza para representar una sección "general" dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones y si lo acompañamos de h1-h6 podemos estructurar mejor toda la página creando jerarquías del contenido, algo muy favorable para el buen posicionamiento web.
- ARTICLE: representa un componente de una página que consiste en una composición autónoma en un documento, página, aplicación, o sitio web con la intención de que pueda ser reutilizado y repetido. Podría utilizarse en los artículos de los foros, una revista o el artículo de periódico, una entrada de un blog, un comentario escrito por un usuario, un widget interactivo o cualquier otro artículo independiente de contenido. Cuando los elementos de <article> son anidados, los elementos interiores representan los artículos que en principio son relacionados con el contenido del artículo externo. Por ejemplo, un artículo de un blog que permite comentarios de usuario, dichos comentarios se podrían representar con <article>.
- ASIDE: representa una sección de la página que abarca un contenido relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Este elemento puede utilizarse para efectos tipográficos, barras laterales, elementos publicitarios,

para grupos de elementos de la navegación, u otro contenido que se considere separado del contenido principal de la página.

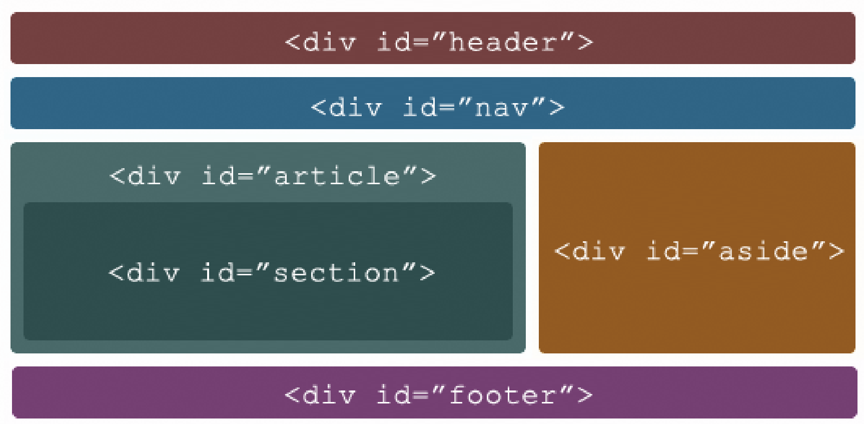
- HEADER: representa un grupo de artículos introductorios o de navegación. Está destinado a contener por lo general la cabecera de la sección (un elemento h1-h6 o un elemento hgroup), pero no es necesario.
- NAV: representa una sección de una página que enlaza a otras páginas o a otras partes dentro de la página. No todos los grupos de enlaces en una página necesita estar en un elemento nav, sólo las secciones que constan de bloques de navegación principales son apropiados para el elemento de navegación.
- FOOTER: representa el pie de una sección, con información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright o el año.
- HGROUP: representa el encabezado de una sección. El elemento se utiliza para agrupar un conjunto de elementos h1-h6 cuando el título tiene varios niveles, tales como subtítulos o títulos alternativos.
- TIME: representa o bien una hora (en formato de 24 horas), o una fecha precisa en el calendario gregoriano (en formato ISO), opcionalmente con un tiempo y un desplazamiento de zona horaria.

3.7. ESTRUCTURA DE UN DOCUMENTO HTML5

Como hemos visto con las nuevas etiquetas semánticas introducidas en HTML5, éstas aportan un significado concreto al documento que estamos definiendo, y por lo tanto, afectan de manera directa a la forma en la estructuramos el contenido. Vamos a ver cómo podemos convertir nuestra actual página con las nuevas etiquetas introducidas en HTML5.

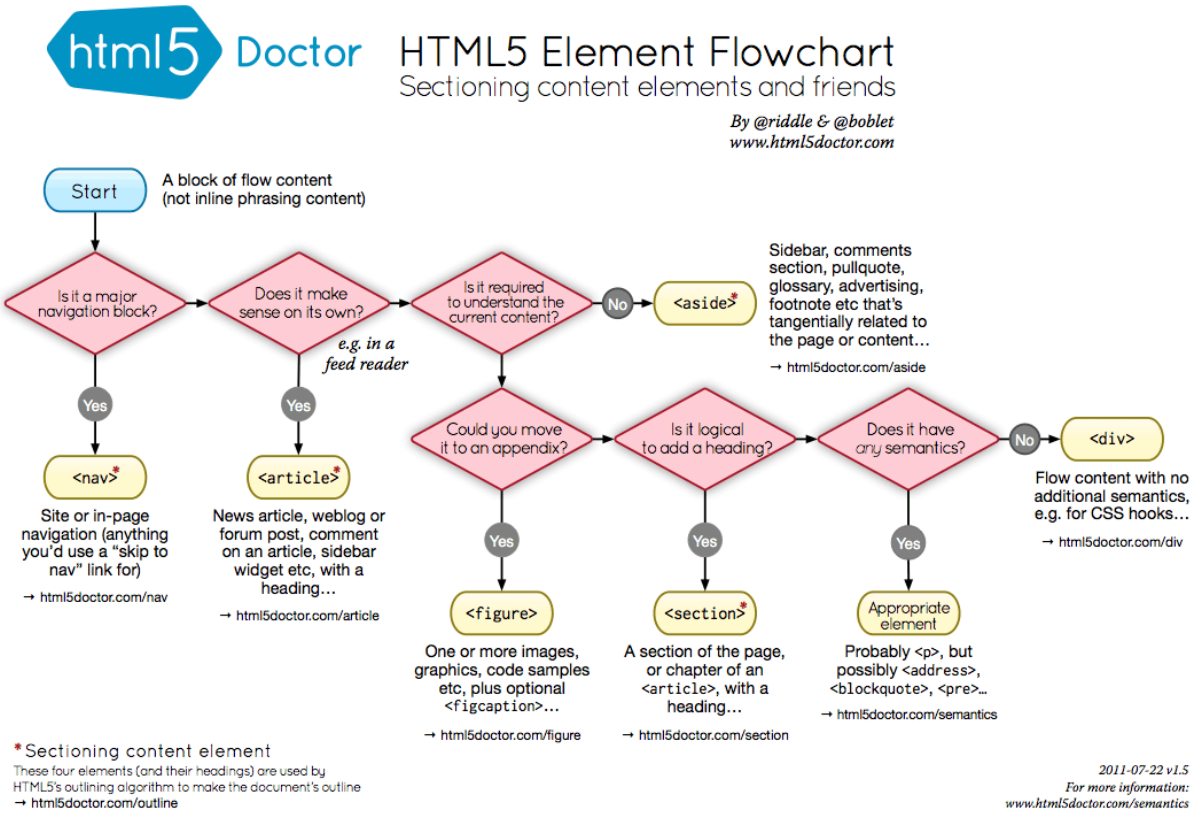
3.7.1. Estructura tradicional de un documento en HTML 4

El siguiente “dibujo” muestra una estructura "clásica" de documento HTML, donde los diferentes contenidos de la web se encuentran agrupados por etiquetas <div>. Por sí mismas, estas etiquetas no aportan ningún tipo de significado, y el atributo id tampoco se lo proporciona. Si cambiamos <div id="header"> por <div id="whatever">, el significado sigue siendo el mismo, ninguno.



Partiendo de [illegible]s utilizar. Esto no siempre es así, y cuando estructuramos contenidos de mayor alcance, lo normal es que surjan dudas. Un

sencillo algoritmo que nos puede ayudar en la correcta selección de etiquetas, es el que proponen en HTML5 Doctor.



3.8. USO DE LAS NUEVAS ETIQUETAS SEMÁNTICAS

3.8.1. HEADER

Según la especificación, un elemento <header> representa lo siguiente:

> *Un grupo de navegación o contenido introductorio. Un elemento header normalmente contiene una sección de encabezado (un elemento h1-h6 o un elemento hgroup), pero puede contener otro tipo de elementos, como una tabla de contenidos, un formulario de búsqueda o cualquier logo importante.*

En nuestro ejemplo, y en la mayoría de los sitios web, la cabecera contiene los primeros elementos de la página. Tradicionalmente el título de la web, su logo, enlaces para volver al inicio... De la manera más simple, nuestra cabecera quedaría de esta forma:

```
<header>
    <a href="/"><img src=logo.png alt="home"></a>
    <h1>Title</h1>
</header>
```

También es muy común que los sitios web muestren un lema o subtítulo bajo el título principal. Para dar mayor importancia a este subtítulo, y relacionarlo de alguna manera con el título principal de la web, es posible agrupar los dos titulares bajo un elemento <hgroup>.

```
<header>
    <a href="/"><img src=logo.png alt="home"></a>
    <hgroup>
        <h1>Title</h1>
        <h2 class="tagline">
            A lot of effort went into making this effortless.
        </h2>
    </hgroup>
</header>
```

3.8.2. NAV

Según la especificación, un elemento <nav> representa lo siguiente:

> *El elemento <nav> representa una sección de una página que enlaza con otras páginas o partes de la misma página: una sección con enlaces de navegación.*

El elemento <nav> ha sido diseñado para identificar la navegación de un sitio web. La navegación se define como un conjunto de enlaces que hacen referencia a las secciones de una página o un sitio, pero no todos los enlaces son candidatos de pertenecer a un elemento <nav>: una lista de enlaces a patrocinadores o los resultados de una búsqueda, no forman parte de la navegación principal, sino que corresponden con el contenido de la página.

Como ocurre con los elementos <header>, <footer> y el resto de nuevas etiquetas, no estamos obligados a utilizar un único elemento <nav> en toda la página. Es posible que tengamos una navegación principal en la cabecera de la página, una tabla de contenidos o enlaces en el pie de la página, que apuntan a contenidos secundarios. Todos ellos son candidatos a pertenecer a un elemento <nav>.

```
<nav>
    <ul>
        <li><a href="#">home</a></li>
        <li><a href="#">blog</a></li>
        <li><a href="#">gallery</a></li>
        <li><a href="#">about</a></li>
    </ul>
</nav>
```

3.8.3. FOOTER

Según la especificación, un elemento <footer> representa lo siguiente:

> *Representa el pie de una sección. Un pie tradicionalmente contiene información acerca de su sección, como quién escribió el contenido, enlaces relacionados, copyright y similares.*

Al igual que ocurre con el elemento <nav>, podemos tener tantos elementos <footer> como sea necesario. Lo normal es que nuestro sitio web disponga de al menos un pie principal, que contiene los avisos legales (privacidad, condiciones del servicio, copyright...), mapa del sitio web, accesibilidad, contacto y otros muchos enlaces que pueden ir incluidos en un elemento <nav>.

3.8.4. ARTICLE

Según la especificación, un elemento <article> representa lo siguiente:

> *Este elemento representa un contenido completo, auto-contenido en un documento, página, aplicación o sitio web, que es, en principio, independiente de ser distribuido y reutilizado, por ejemplo en un RSS. Puede ser un post de un foro, un artículo de un periódico o revista, una entrada de un blog, un comentario de un usuario, un widget o cualquier otro elemento independiente.*

Cuando los artículos están anidados, los artículos interiores representan contenido que en principio está relacionado con el artículo que los contiene. Por ejemplo, una entrada de un blog puede aceptar comentarios de usuarios, que están incluidos dentro del contenido principal y relacionados con el mismo.

Por lo tanto, la etiqueta <article> se utiliza para encapsular contenido, que tiene significado en sí mismo, y que puede ser distribuido y reutilizado en otros formatos de datos. No nos referimos únicamente a contenidos clásicos de texto, sino que incluso un contenido multimedia con su transcripción, un mapa o email pueden ser totalmente válidos para ser incluidos en una etiqueta <article>.

```
<section>
    <h1>Comments</h1>
    <article id="c1">
        <footer>
            <p>Posted by: <span>George Washington</span></p>
            <p><time datetime="2009-10-10">15 minutes ago</time></p>
        </footer>
        <p>Yeah! Especially when talking about your lobbyist friends!</p>
    </article>
    <article id="c2">
        <footer>
            <p>Posted by: <span itemprop="name">George Hammond</span></p>
            <p><time datetime="2009-10-10">5 minutes ago</time></p>
        </footer>
        <p>Hey, you have the same first name as me.</p>
    </article>
</section>
```

3.8.5. SECTION

A diferencia del elemento <article>, este elemento es utilizado para dividir el documento (o artículos) en diferentes áreas, o como su propio nombre indica, en secciones. Según la especificación, un elemento <section> representa lo siguiente:

> *Representa una sección genérica de un documento o aplicación. Una sección, en este contexto, es un grupo temático de contenido, que generalmente incluye una cabecera.*

Consideremos el siguiente marcado válido en HTML 4:

```
<h1>Rules for Munchkins</h1>
<h2>Yellow Brick Road</h2>
<p>It is vital that Dorothy follows it—so no selling
    bricks as "souvenirs"</p>
<h2>Fan Club uniforms</h2>
<p>All Munchkins are obliged to wear their "I'm a friend
    of Dorothy!" t-shirt when representing the club</p>
<p><strong>Vital caveat about the information above:
    does not apply on the first Thursday of the month.</strong></p>
```

En este caso, y desde un punto de vista semántico, es complicado deducir si el texto Vital caveat about the information above: does not apply on the first Thursday of the month. pertenece al contenido completo o está relacionado con la sección Fan Club uniforms. Gracias a la etiqueta <section>, es muy sencillo separar e identificar a qué sección pertenece cada contenido:

```
<article>
```

```
    <h1>Rules for Munchkins</h1>
    <section>
        <h2>Yellow Brick Road</h2>
        <p>It is vital that Dorothy follows it—so no selling
            bricks as "souvenirs"</p>
    </section>
    <section>
        <h2>Fan Club uniforms</h2>
        <p>All Munchkins are obliged to wear their "I'm a friend
            of Dorothy!" t-shirt when representing the club</p>
    </section>
    <p><strong>Vital caveat about the information above:
        does not apply on the first Thursday of the month.</strong></p>
</article>

<article>
    <h1>Rules for Munchkins</h1>
    <section>
        <h2>Yellow Brick Road</h2>
        <p>It is vital that Dorothy follows it—so no selling
            bricks as "souvenirs"</p>
    </section>
    <section>
        <h2>Fan Club uniforms</h2>
        <p>All Munchkins are obliged to wear their "I'm a friend
            of Dorothy!" t-shirt when representing the club</p>
        <p><strong>Vital caveat about the information above:
            does not apply on the first Thursday of the month.</strong></p>
    </section>
</article>
```

Como podemos observar en los dos ejemplos anteriores, es muy sencillo agrupar contenido que pertenece a una misma sección, permitiendo incluirlo dentro de un contexto semántico.

Otra de las posibilidades que nos ofrece esta etiqueta, es la de dividir nuestro documento en secciones, que incluyen contenido de temáticas diferentes entre sí. Si además queremos separar estos contenidos visualmente en dos columnas, lo lógico sería utilizar las tradicionales etiquetas para agrupar los artículos según su temática, y posteriormente aplicar estilos CSS o JavaScript para presentarlos en forma de pestañas.

En este caso, la etiqueta no nos aporta ningún significado semántico, tan sólo estructural. La etiqueta <section> es la encargada de añadir semántica en estos casos:

```
<section>
    <h1>Articles about llamas</h1>
    <article>
        <h2>The daily llama: Buddhism and South American camelids</h2>
        <p>blah blah</p>
    </article>
    <article>
        <h2>Shh! Do not alarm a llama</h2>
        <p>blah blah</p>
    </article>
</section>
<section>
    <h1>Articles about root vegetables</h1>
    <article>
        <h2>Carrots: the orange miracle</h2>
        <p>blah blah</p>
    </article>
    <article>
        <h2>Eat more Swedes (the vegetables, not the people)</h2>
        <p>blah blah</p>
    </article>
</section>
```

3.8.6. ASIDE

Según la especificación, un elemento <aside> representa lo siguiente:

> *Una sección de una página que consiste en contenido tangencialmente relacionado con el contenido alrededor del elemento, y puede considerarse separado de este contenido. Estas secciones son normalmente representadas como elementos laterales en medios impresos. Este elemento puede utilizarse contener citas, anuncios, grupos de elementos de navegación y cualquier otro contenido separado del contenido principal de la pagina.*

Dentro de un artículo, por ejemplo, puede ser utilizado para mostrar contenido relacionado como citas u otros artículos relacionados.

3.8.7. FIGURE y FIGCAPTION

Según la especificación, un elemento <figure> representa lo siguiente:

> *The figure element represents some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.*
>
> *The element can be used to annotate illustrations, diagrams, photos, code listings, etc. This includes, but is not restricted to, content referred to from the main part of the document, but*

> *that could, without affecting the flow of the document, be moved away from that primary content, e.g. to the side of the page, to dedicated pages, or to an appendix.*

Hasta ahora, no había una manera correcta de poder añadir un subtítulo o una leyenda a un contenido concreto, como explicar una figura o atribuir una imagen a un fotógrafo. Gracias a la etiqueta <figure> podemos contener una imagen (o un vídeo, ilustración o bloque de código) en un elemento y relacionarlo con un contenido concreto:

```
<figure>
    <img src="welcome.jpg" alt="Alternative text">
    <figcaption>
        Bruce and Remy welcome questions
            <small>Photo &copy; Bruce's mum</small>
    </figcaption>
</figure>
```

En conveniente recordar que el atributo alt indica el texto a mostrar cuando la imagen no está disponible, y no está pensada para contener una descripción de la imagen, y mucho menos para duplicar lo ya indicado en la etiqueta <figcaption>.

3.8.8. HGROUP

La etiqueta `<hgroup>` representa el encabezado de una sección en grupo. Es decir, que nos permite encerrar títulos en bloques para que se tomen como un único elemento.

En el siguiente ejemplo lo que conseguimos es que, en el análisis semántico, cuando se cierre la sección se cierre el contexto del `<h1>` y `<h2>` y no solamente del `<h2>`, que es lo que ocurriría si no tuviese la etiqueta `<hgroup>`.

```
<section>
   <hgroup>
      <h1>Primera sección de nivel 1</h1>
      <h2>Breve descripción de la sección de nivel 1</h2>
   </hgroup>
   <p>En esta nueva sección hablaremos de bla bla bla y de blu blu blu. Bla bli blo blu...</p>
</section>
```

3.8.9. DETAILS y SUMMARY

La etiqueta `<details>` se utiliza para especificar detalles adicionales a un texto. Dentro de esta etiqueta podemos encontrar la etiqueta `summary` que es un resumen de lo contenido en `<details>`. Además, proporciona un evento de click para que el usuario pueda ver u ocultar los detalles.

```
<details>
```

```
  <summary>Texto en el que pulsar</summary>
  <p>Este texto se muestra/oculta al pulsar en el texto anterior</p>
</details>
```

3.9. ATRIBUTOS GLOBALES

HTML5 también incluye nuevos atributos globales que pueden ser asignados a cualquier elemento. Son los siguientes:

3.9.1. ACCESSKEY

El atributo accesskey permite a los desarrolladores especificar un atajo de teclado que permite activar un elemento a asignarle el foco. Este atributo ya existía en HTML 4, aunque ha sido utilizado en muy pocas ocasiones. Como HTML5 está pensado para aplicaciones, y algunos usuarios siguen prefiriendo los atajos de teclado, este atributo no ha sido eliminado, y ahora está disponible para cualquier elemento.

Para evitar conflictos con otros atajos de teclado, o con los propios del navegador, ahora esta etiqueta permite asignar alternativas en este atributo. Un ejemplo incluido en la especificación:

```
<input type="search" name="q" accesskey="s 0">
```

Esto quiere decir que este elemento es accesible a través de dos atajos de teclado, a través de la tecla s o a través de la tecla 0 (en ese orden).

3.9.2. CONTENTEDITABLE

Inventado por Microsoft, e implementado por el resto de los navegadores, la etiqueta contenteditable es ahora parte de la especificación oficial.

La introducción de esta etiqueta significa principalmente dos cosas:

- Primero, los usuarios pueden editar los contenidos de un elemento que incluya esta etiqueta. Este elemento debe ser seleccionable y el navegador debe proporcionar una marca que indique la posición actual del cursor.
- Y segundo, es posible cambiar el formato del texto del contenido, añadiendo negritas, cambiar la fuente, añadir listas, etc.

Este atributo es de tipo booleano, por lo que su valor puede ser true o false. Al acceder desde JavaScript a este atributo, hay que tener en cuenta su notación lowerCamelCase, siendo el nombre de la propiedad del DOM contentEditable. Además, existe otra propiedad llamada isContentEditable, que indica si el elemento es editable o no.

Finalmente, el contenido que ha sido seleccionado por el usuario, puede ser objeto de modificaciones, como hemos comentado antes. A través del comando element.execCommand() es posible indicar el tipo de modificación (poner en negrita, copiar, cambiar la fuente...), siempre que el documento se haya indicado como editable.

document.designMode = 'on';

Si se desea almacenar los cambios realizados en el contenido, es necesario enviarlo al servidor. No existe ningún API o método en JavaScript que nos posibilite esta acción, por lo que debemos utilizar algún tipo de tecnología tipo AJAX.

3.9.3. DATA-* (CUSTOM DATA ATTRIBUTES)

HTML5 permite crear atributos personalizados para los elementos. Estos atributos son utilizados para pasar información a JavaScript. Como veremos en el capítulo correspondiente, hasta ahora se utilizaba el atributo class para de alguna manera almacenar información asociada con elementos, pero esto cambia radicalmente con estos atributos.

```
<ul id="vegetable-seeds">
  <li data-spacing="10cm" data-sowing-time="March to June">Carrots</li>
  <li data-spacing="30cm" data-sowing-time="February to March">Celery</li>
  <li data-spacing="3cm" data-sowing-time="March to September">Radishes</li>
</ul>
```

3.9.4. DRAGGABLE

Este atributo indica que el elemento indicado puede ser arrastable. Lo veremos en el capítulo correspondiente.

4. Los Metadatos

La etiqueta ***meta*** representa varios tipos de metadatos que no pueden ser expresados con las etiquetas *title*, *base*, *link*, *style* y *script*. Con ella podemos indicar el tipo de codificación de nuestra página entre otros tipos de metadatos. Como muchos ya sabrán, para esta tarea, se utilizan los atributos ***name***, ***http-equiv***, ***charset***, ***scheme*** e ***itemprop***.

El atributo ***charset*** se utiliza para especificar la codificación usada en nuestra página. Únicamente debe existir una etiqueta *meta* con el atributo *charset* en la página. En él indicaremos si nuestra página esta codificada en UTF-8, ISO o cualquier otro tipo de codificación. Este elemento debe estar dentro del los primeros 512 bytes de la página:

El atributo ***content*** se utiliza para dar valores a los metadatos de la página o a las directivas **pragma**. No existe una lista de valores permitidos.

El atributo ***name*** asigna un metadato a la página. Un metadato esta expresado como un par nombre-valor. El atributo *name* es el encargado de especificar que metadato queremos crear, y el atributo *content* proporciona su valor. En resumen, es el nombre de la propiedad que se define y no existe una lista oficial de propiedades.

El atributo ***http-equiv*** se dice que es una directiva pragma. En ocasiones, reemplaza al atributo “name” y lo emplean los servidores para adaptar sus respuestas al documento.

El atributo ***scheme*** indica el esquema que se debe emplear para interpretar el valor de la propiedad. Permite proporcionar a los agentes de usuario más contexto para la interpretación correcta de los metadatos. A veces, esta información adicional puede ser crítica, por ejemplo cuando los metadatos pueden ser especificados según formatos diferentes. Por ejemplo, un autor podría especificar una fecha en el formato ambiguo "10-9-97"; ¿significa esto 9 de octubre de 1997 o 10 de septiembre de 1997? El valor "Mes-Día-Año" para el atributo scheme eliminaría la ambigüedad de este valor de fecha. En otras ocasiones, el atributo scheme puede proporcionar información útil aunque no crítica.

4.1. APPLICATION-NAME

Representa el nombre que tendrá el documento web.

4.2. AUTHOR

Indica el nombre del creador de la página.

4.3. KEYWORDS

Es el conjunto de palabras más relevantes de nuestro documento. Por ejemplo, las palabras más relevantes para un documento que hable sobre HTML5, podrían ser: HTML5, Video, Tags, Enlaces. Este metadato ya no tiene tanta relevancia como hace años.

4.4. DESCRIPTION

Pretende dar una pequeña descripción de lo que contiene nuestra página. Este valor será muy utilizado por los directorios de páginas y buscadores.

4.5. GENERATOR

Se utiliza para indicar con que software se ha creado la página. Si tú has creado el documento a mano no necesitas incluir este metadato en él.

4.6. ROBOTS

Define el comportamiento que los robots deben tener con la página. Podremos incluir todos los valores que queramos separados por comas:

Valor	Descripción	Usado
`index`	Permite al robot indexar la página	Todos
`noindex`	Indica al robot que no indexe la página	Todos
`follow`	Permite al robot seguir los links en la página	Todos
`nofollow`	Indica al robot que no puede seguir los links de la página	Todos
`noodp`	Previene el uso de la descripción Open Directory Project, si la hay, como la descripción de la página en los resultados de una búsqueda	Google, Yahoo, Bing
`noarchive`	Previene que los motores de búsqueda cachear el contenido de la página	Google, Yahoo
`nosnippet`	Previene que se muestre la descripción de la página en los resultados de una búsqueda	Google
`noimageindex`	Previene que la página aparezca referida de un indice de imágenes	Google
`noydir`	Previene el uso de la descripción Yahoo Directory, si la hay, como descripción de la página en los resultados de una búsqueda	Yahoo

Valor	Descripción	Usado
`nocache`	Sinónimo de `noarchive`	Bing

4.7. CREATOR

Define el nombre del creador de la página. Si existe más de uno se deberán usar varios de estos elementos.

4.8. GOOGLEBOT

Es sinónimo de robots, pero solo para Googlebot.

4.9. VIEWPORT

Ayuda a conocer el tamaño inicia del viewport. Este pragma es utilizado únicamente por dispositivos móviles.

Valor	Posibles valores	Descripción
`width`	un entero positivo o la cadena `device-width`	define el ancho, en pixels, del viewport
`heigth`	un entero positivo o la cadena `device-height`	define la altura, en pixels, del viewport
`initial-scale`	un número positivo entre `0.0` y `10.0`	define el ratio entre el ancho del dispositivo y el tamaño del viewport
`maximum-scale`	un número positivo entre `0.0` y `10.0`	define el valor máximo del zoom; éste debe ser mayor o igual a la `minimum-scale` o el comportamiento sera indeterminado.
`minimum-scale`	un número positivo entre `0.0` y `10.0`	define el valor mínimo del zoom; este debe ser menor o igual a la `maximum-scale` o el comportamiento será indeterminado
`user-scalable`	un valor booleano (`yes` or `no`)	Si tiene el valor `no`, el usuario no podrá hacer zoom en la página. Por defecto el valor es `yes`.

4.10. MSAPPLICATION-TASK

Crea una acción asociada a la página. Esta tarea aparecera en el menu del sitio anclado en windows7.

4.11. MSAPPLICATION-STARTURL

Contienen la dirección URL raíz de la aplicación, en un sitio anclado.

4.12. MSAPPLICATION-TOOLTIP

Proporcionan texto de información adicional sobre herramientas que aparece cuando mantiene el mouse sobre el acceso directo del sitio anclado en el menú Inicio de Windows o en el escritorio.

4.13. MSAPPLICATION-NAVBUTTON-COLOR

Definen el color personalizado de los botones Atrás y Adelante en la ventana del explorador del sitio anclado.

4.14. MSAPPLICATION-WINDOW

Establecen el tamaño inicial de la ventana del sitio anclado cuando el sitio se inicia por primera vez.

4.15. MOBILEOPTIMIZED

Indica el ancho de que tendrá la página en un dispositivo móvil.

4.16. HANDHELDFRIENDLY

Se utiliza para indicar a los navegadores portátiles que la página está optimizada para ser vista en un dispositivo móvil. Sin ella, las tablas, el código JavaScript y ciertas etiquetas o funcionalidades relacionadas con las imágenes se perderán cuando se descarga la página. Sus valores son o “true” o “false”

4.17. PALMCOMPUTINGPALTFORM

Indica que la aplicación está diseñada específicamente para dispositivos Palm. Sus valores son o “true” o “false”.

4.18. HTTP-EQUIV=”CONTENT-TYPE”

Es la alternativa al atributo charset, las dos tienen la misma funcionalidad. En este tipo de declaración el atributo *content* debe tener como valor la cadena, seguido por la cadena *“charset=”*, y el identificador de la codificación.

4.19. HTTP-EQUIV=”REFRESH”

Este pragma indica el tiempo para que se refresque la página. El tiempo se representa con un entero, que indica el numero de segundos para que se refresque la página. Si añadimos una URL este será el tiempo que tardara la pagina en refrescarse y redireccionarse a la URL.

4.20. HTTP-EQUIV=”X-UA-COMPATIBLE”

X-UA-Compatible que permite a los desarrolladores de Internet elegir qué versión de Internet Explorer se desea que se ejecute. Por ejemplo, es utilizado por Internet Explorer para especificar si una página debe ser ejecutada en modo de vista de compatibilidad, en modo emulación o en modo “edge”. También permite activar Google Chrome Frame (complemento que funciona con Internet Explorer 6, 7, 8 y 9 y permite que sitios web programados adecuadamente sean mostrados de manera correcta bajo Internet Explorer usando las versiones de Google Chrome del motor de renderizado WebKit y V8.).

4.21. EXPIRES

Esta etiqueta declara a los motores de búsqueda cuando expirará el contenido del sitio web. Define la fecha y hora de caducidad del documento que se indexa. Por ejemplo, si un sitio web está ejecutando un evento dentro de un intervalo de tiempo limitado se debe establecer este metadato para indicar a los motores de búsqueda cuando eliminar su página web desde su base de datos.

También se utiliza comúnmente en conjunción con la etiqueta **revisit** como un medio para conseguir que los motores de búsqueda vuelvan a visitar un sitio web cada pocos días.

A modo de añadido, si se reemplaza o cambia el nombre de una página web se recomienda realizar una redirección 301 a fin de señalar a los motores de búsqueda la dirección correcta.

Hay que estar seguro de que el valor de content está en formato RFC1123 (por ejemplo `tue, 01 Jun 2010 19:45:00 GMT`).

Los valores más comunes son “`FECHA`” para indicar hasta cuando es válida o “`0`” que indica que la página se ha eliminado.

4.22. COPYRIGHT

Se utiliza para indicar que las fotos y/o textos están protegidos. No tiene ninguna influencia en el posicionamiento en buscadores.

4.23. RATING

Si se desea la evaluación de una página web se debe utilizar la etiqueta **rating**.

Esta etiqueta se utiliza a menudo para permitir que los internautas más jóvenes sepan que el contenido es apropiado. Si utiliza esta etiqueta por el camino equivocado los motores de búsqueda podrían marcar la web como inapropiada e, incluso, expulsarla. Por esta razón, esta etiqueta no tiene ninguna influencia en su posición en los motores de búsqueda a menos que utilice de la manera malintencionada.

Sus posibles valores son **general**, **mature**, **restricted**, **14 years** y **safe for kids**.

4.24. METADATOS DUBLIN CORE

Dublin Core es un modelo de metadatos elaborado por la DCMI (Dublin Core Metadata Initiative) y compuesto por 15 definiciones semánticas descriptivas. Es una organización que se dedica a fomentar la adopción extensa de los estándares interoperables de los metadatos y a promover el desarrollo de los vocabularios especializados de metadatos para describir recursos para permitir sistemas más inteligentes el descubrimiento del recurso. Las normas que sigue la especificación Dublin Core son la ISO 15836, y la norma NISO Z39.85-2012.

Las definiciones son opcionales, pueden repetirse y pueden aparecer en cualquier orden.

4.24.1. DC. Title

Más Detalles en http://purl.org/dc/elements/1.1/title
Es el nombre formal por el que es conocido el recurso.

4.24.2. DC. Creator

Más Detalles en http://purl.org/dc/elements/1.1/creator
Representa la entidad principalmente responsable de la creación del contenido intelectual del recurso.

4.24.3. DC. Subject

Más Detalles en http://purl.org/dc/elements/1.1/subject
Es el tema del contenido del recurso. Normalmente está expresado en forma de palabras clave (keywords), frases clave o códigos de clasificación que describen el tema de un recurso.

4.24.4. DC. Description

Más Detalles en http://purl.org/dc/elements/1.1/description
Es la descripción del contenido del recurso. La descripción puede ser un resumen, una tabla de contenidos, una referencia a una representación gráfica de contenido o una descripción de texto libre del contenido.

4.24.5. DC. Publisher

Más Detalles en http://purl.org/dc/elements/1.1/publisher
Es la entidad, normalmente la empresa o persona que posee la propiedad intelectual, responsable de publicar el recurso.

4.24.6. DC. Contributor

Más Detalles en http://purl.org/dc/elements/1.1/contributor
Representa a una persona u organización que ha colaborado en el contenido del recurso.

4.24.7. DC. Date

Más Detalles en http://purl.org/dc/elements/1.1/date
Representa la fecha asociada con algún evento en el ciclo de vida del recurso. Típicamente, la fecha será asociada con la creación o disponibilidad del recurso. Normalmente se debe expresar bajo la norma ISO 8601 que sigue el formato YYYY-MM-DD.

4.24.8. DC. Type

Más Detalles en http://purl.org/dc/elements/1.1/type
Representa la naturaleza o categoría del contenido del recurso. Debe incluir términos que describan las categorías generales, funciones, géneros o niveles de agregación del contenido y debe seguir un vocabulario controlado por la DCMI Types de la normativa Dublin Core.

4.24.9. DC. Format

Más Detalles en http://purl.org/dc/elements/1.1/format
Representa el fichero de manifiesto, el medio físico o las dimensiones del recurso como el tamaño y la duración. Se recomienda seleccionar un valor de un vocabulario controlado por las definiciones de la Media Types de la Dublin Core.

4.24.10. DC. Identifier

Más Detalles en http://purl.org/dc/elements/1.1/identifier
Es una referencia no ambigua para el recurso dentro de un contexto dado. Normalmente este recurso estará establecido a través de un sistema de identificación formal como una URI compuesta de una URL con un identificador DOI o un identificador en formato ISBN.

4.24.11. DC. Source

Más Detalles en http://purl.org/dc/elements/1.1/source
Es una referencia a el recurso del cual se deriva o proviene el recurso actual. El recurso actual puede derivarse, en todo o en parte, de un recurso fuente. Normalmente será la referencia no ambigua para el recurso del que deriva o proviene (DC.Identifier).

4.24.12. DC. Language

Más Detalles en http://purl.org/dc/elements/1.1/language
El idioma o lenguaje en el que está escrito el recurso. Este valor estará representado por la RFC 3066 (http://www.ietf.org/rfc/rfc3066.txt) en conjunción con la ISO 639-2 (http://www.loc.gov/standards/iso639-2/), que define las etiquetas de tres letras primarias para lenguaje, con sub-etiquetas opcionales.

4.24.13. DC. Relation

Más Detalles en http://purl.org/dc/elements/1.1/relation
Es una referencia a un recurso relacionado. Normalmente será la referencia no ambigua del recurso con el que tiene relación (DC.Identifier).

4.24.14. DC. Coverage

Más Detalles en http://purl.org/dc/elements/1.1/coverage
Es la extensión o ámbito del contenido del recurso. La cobertura incluiría la localización espacial (un nombre de lugar o coordenadas geográficas), el período temporal (una etiqueta del período, fecha o rango de datos) o jurisdicción (tal como el nombre de una entidad administrativa). Se recomienda seleccionar un valor de un vocabulario controlado (por ejemplo, del Thesaurus of Geographic Names (TGN) y que, donde sea apropiado, se usen preferentemente los nombres de lugares o períodos de tiempo antes que los identificadores numéricos tales como un conjunto de coordenadas o rangos de datos.

4.24.15. DC. Rights

Más Detalles en http://purl.org/dc/elements/1.1/rights
La información sobre los derechos de propiedad y sobre el recurso. Este elemento podrá contener un estamento de gestión de derechos para el recurso, o referencia a un servicio que provea tal información. La información sobre derechos a menudo corresponde a los derechos de propiedad intelectual, copyright y otros derechos de propiedad.

4.25. METADATOS OPEN GRAPH

El protocolo Open Graph permite a cualquier página web convertirse en un objeto rico en un gráfico social. Por ejemplo, esto se utiliza en Facebook para permitir a cualquier página web para tener la misma funcionalidad que cualquier otro objeto en Facebook.

Mientras que muchas tecnologías y esquemas diferentes existen y podrían combinarse juntas, no hay una única tecnología que proporcione suficiente información para enriquecer una página web dentro del marco social. El protocolo Open Graph se construye sobre estas tecnologías existentes y ofrece a los desarrolladores una “fórmula” para implantar esta carencia. El desarrollador simplemente es un objetivo clave del protocolo Open Graph que haya informado a muchas de las decisiones de diseño técnico.

4.25.1. og:title

Es el título del objeto, por ejemplo, "The Rock".

4.25.2. og:type

Es el tipo o medio del objeto, por ejemplo, "video.movie". Dependiendo del tipo especificado, otras propiedades podrían ser necesarias.

4.25.3. og:image

Es la URL de la imagen que representa el objeto.

4.25.4. og:url

Es la URL canónica del objeto que será utilizada como identificación permanente para el objeto, por ejemplo, "http://www.imdb.com/title/tt0117500/".

4.25.5. og:audio

Especifica la URL que contiene el audio que acompaña al objeto.

4.25.6. og:description

Es la descripción del objeto.

4.25.7. og:determiner

Es la palabra que aparece antes del título del objeto en una oración. Viene respresentado por los valores ("a", "an", "the", "", "auto"). Si se elige "auto", los valores válidos son "a" o "an". Por defecto es "".

4.25.8. og:locale

La configuración regional descrita en formato de la ISO 8859-1. Por defecto es "en_US".

4.25.9. og:locale:alternate

Es un array con otros lugares dónde está disponible esta página.

4.25.10. og:site_name

Si el objeto forma parte de una web mayor (más grande) se representa aquí. Por ejemplo, "IMDb".

4.25.11. og:video

Es la URL del video que contiene el objeto

4.25.12. og:image:url

Es identical a `og:image`.

4.25.13. og:image:secure_url

URL alternativa a utilizar si la página web requiere HTTPS.

4.25.14. og:image:type

Es el format MIME de la imagen.

4.25.15. og:image:width

Es el ancho de la imagen en pixels.

4.25.16. og:image:height

Es el alto de la imagen en pixels.

4.26. OTROS METADATOS

4.26.1. Twitter Cards

En el caso de Twitter podemos definir cómo presentar nuestra información a través de las Twitter cards. Mediante su uso, se puede controlar cómo se mostrará el contenido, eligiendo la manera más adecuada al tipo de contenido que se haya creado. Si, por ejemplo, seleccionamos Summary se verá un tweet que incluirá un enlace "ver resumen" en el que ampliar información sobre la publicación. Si seleccionamos el tipo "gallery" estaremos indicando que el contenido es una galería de imágenes.

- **twitter:card** – Define el tipo de presentación que se desea para el tweet. Admite distintos valores como "summary", "photo", "gallery", etc.
- **twitter:site** – Nombre del portal web.
- **twitter:url** – Dirección a los contenidos que queremos compartir.
- **twitter:title** – Título del contenido.
- **twitter:descrpition** – Breve resumen del contenido.
- **twitter:image:src** – Imagen que acompañará al contenido.

4.26.2. Open Graph Facebook

En el caso de Facebook, pasa un poco igual que con las Twitter Cards.

- **fb:app_id** – Indica el APP ID de Facebook Insights en la página. Insights permite mostrar datos de tráfico de un sitio de Facebook. Este ID, normalmente, se puede consultar en el App Dashboard de Facebook.
- **fb:admins** – Es el ID del usuario de Facebook.
- **fb:pages** – Normalmente es el ID de usuario de Facebook.

4.27. EJEMPLOS DE METADATOS

```
<!-- Genéricas -->
<meta name="robots" content="index, follow" />
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1" />

<!-- Anclar un sitio con Windows 7 -->
<meta name="application-name" content="Example" />
<meta name="msapplication-tooltip" content="Entrada de ejemplo del curso" />
<meta name="msapplication-starturl" content="http://example.com" />
<meta name="msapplication-navbutton-color" content="black" />
<meta name="msapplication-task" content=" name=Codigobit;action-uri=http://example.com;icon-
uri=http://example.com/favicon.ico" />

<!-- Redirecciona la página después de 3 segundos -->
<meta http-equiv="refresh" content="3;url=http://example.com" />
<meta http-equiv="X-UA-Compatible" content="IE-edge" />
<meta http-equiv="expires" content="31 Dec 2018 08:00:00 GMT" />

<meta name="date" content="2009-01-01" scheme="YYYY-MM-DD" />
<meta name="identificator" content="0-8230-2355-9" scheme="ISBN" />
<meta name="copyright" content="&copy; 2016 Sopra Steria Group" />
<meta name="language" content="spanish" />
<meta name="rating" content="general" />

<!-- Definición de un objeto con Open Graph -->
<meta name="og:title" content="The Rock" />
<meta name="og:type" content="video.movie" />
<meta name="og:url" content="http://www.imdb.com/title/tt0117500/" />
<meta name="og:image" content="http://ia.media-imdb.com/images/rock.jpg" />

<!-- Twitter Card data -->
<meta name="twitter:card" content="summary"/>
<meta name="twitter:site" content="@publisher_handle"/>
<meta name="twitter:title" content="Page Title"/>
<meta name="twitter:description" content="Page description less than 200 characters"/>
<meta name="twitter:creator" content="@author_handle"/>
<-- Twitter Summary card images must be at least 120x120px -->
<meta name="twitter:image" content="http://www.example.com/image.jpg"/>

<-- Facebook metadata -->
<meta name="fb:admins" content="195867593781397" />
<meta name="fb:app_id" content="884158054985857" />
<meta name="fb:pages" content="195867593781397" />
```

5. Elementos de formulario

HTML5 hace que el desarrollo de formularios sea mucho más sencillo. Se han añadido dos nuevos métodos que pueden ser utilizados en la acción del formulario (`update` y `delete`), pero lo más interesante son los nuevos tipos de `input` y elementos de formulario que mejoran la experiencia del usuario y facilitan el desarrollo de los formularios. Estos nuevos elementos añaden en algunos casos, validaciones propias de sus datos, por lo que ya no es necesario JavaScript para realizar este proceso.

5.1. NUEVOS TIPOS DE INPUT

La especificación de HTML5 define 12 nuevos tipos de `input` que podemos utilizar en nuestros formularios. Esta especificación no define cómo deben mostrarse los nuevos tipos en los navegadores, ni los campos ni las validaciones. De hecho, y gracias a cómo está especificado HTML, los navegadores que no *comprendan* los nuevos tipos de entrada, mostrarán un campo de texto tradicional, por lo que la compatibilidad con navegadores antiguos está garantizada.

5.2. TIPO EMAIL

El nuevo tipo `<input type="email">` indica al navegador que no debe permitir que se envíe el formulario si el usuario no ha introducido una dirección de email *válida*, pero no comprueba si la dirección existe o no, sólo si el formato es válido. Como ocurre con el resto de campos de entrada, puede enviar este campo vacío a menos que se indique que es obligatorio.
El atributo `multiple` indica que el valor de este campo, puede ser una lista de emails válidos, separados por comas.

5.3. TIPO URL

El nuevo tipo `<input type="url">` indica al navegador que no debe permitir que se envíe el formulario si el usuario no ha introducido una URL correcta. Algunos navegadores ofrecen ayudas al usuario, como Opera que añade el prefijo `http://` a la URL si el usuario no lo ha introducido. Una URL no tiene que ser necesariamente una dirección web, sino que es posible utilizar cualquier formato de URI válido, como por ejemplo `tel:555123456`.

5.4. TIPO DATE

El nuevo tipo `<input type="date">` es de los más esperados y útiles. En muchos de los sitios web es normal disponer de campos específicos de fecha, donde el usuario debe especificar fechas (para un concierto, vuelo, reserva de hotel, etc). Al existir tantos formatos de fecha diferentes (DD-MM-YYYY o MM-DD-YYYY o YYYY-MM-DD), esto puede suponer un inconveniente para los desarrolladores o los propios usuarios.
Este nuevo tipo de campo resuelve estos problemas, ya que es el navegador el que proporciona la interfaz de usuario para el calendario, e independientemente del formato en el que se muestre, los datos que se envían al servidor cumplen la norma ISO para el formato de fechas.

5.5. TIPO TIME

El nuevo tipo `<input type="time">` permite introducir una hora en formato 24h, y validarlo. De nuevo, es el navegador el encargado de mostrar la interfaz de usuario correspondiente: puede ser un simple campo donde es posible introducir la hora y los minutos, o mostrar algo más complejo como un reloj de agujas.

5.6. TIPO DATETIME

Este nuevo tipo de campo es la combinación de los tipos `date` y `time`, por lo que se valida tanto la fecha como la hora introducida.

5.7. TIPO DATETIME-LOCAL

Este nuevo tipo de campo es la combinación de los tipos `date` y `time`, por lo que se valida tanto la fecha como la hora introducida pero, en la zona horaria local.

5.8. TIPO MONTH

El nuevo tipo `<input type="month">` permite la selección de un mes en concreto. La representación interna del mes es un valor entre 1 y 12, pero de nuevo queda en manos del navegador la manera de mostrarlo al usuario, utilizando los nombres de los meses por ejemplo.

5.9. TIPO WEEK

El nuevo tipo `<input type="week">` permite la selección de una semana del año concreta. La representación interna del mes es un valor entre 1 y 53, pero de nuevo queda en manos del navegador la manera de mostrarlo al usuario.

5.10. TIPO NUMBER

Como es de esperar, el nuevo tipo `<input type="number">` valida la entrada de un tipo de dato numérico. Este tipo de campo encaja perfectamente con los atributos `min`, `max` y `step`, que veremos más adelante.

5.11. TIPO RANGE

El nuevo tipo `<input type="range">`, muestra un control deslizante en el navegador. Para conseguir un elemento de este estilo, era necesario un gran esfuerzo para combinar imágenes, funcionalidad y accesibilidad, siendo ahora mucho más sencillo. Este tipo de campo encaja de nuevo perfectamente con los atributos `min`, `max` y `step`, que veremos más adelante.

5.12. TIPO TEL

El nuevo tipo `<input type="tel">`, espera que se proporcione un número de teléfono. No se realiza ninguna validación, ni se obliga a que únicamente se proporcionen caracteres numéricos, ya que un número de teléfono puede representarse de muchas maneras: *+44 (0) 208 123 1234*.
La gran ventaja de este campo es que en dispositivos con un teclado virtual, éste se adaptará para mostrar únicamente los caracteres asociados on números de teléfono.

5.13. TIPO SEARCH

El nuevo tipo `<input type="search">`, espera que se proporcione un término de búsqueda. La diferencia con un campo de texto normal, es únicamente estética, aunque puede ofrecer alguna funcionalidad extra como un histórico de últimos términos introducidos o una ayuda para el borrado. Por norma general, toma el aspecto de un campo de búsqueda del navegador o sistema operativo.

5.14. TIPO COLOR

El nuevo tipo `<input type="color">`, permite seleccionar un color de una paleta de colores mostrada por el navegador. Esta paleta de colores, coincide, por norma general, con la interfaz de selección de colores del sistema operativo.

6. Atributos nuevos

Al igual que los nuevos tipos de campo, el elemento `input` ha recibido nuevos atributos para definir su comportamiento y restricciones: `autocomplete`, `min`, `max`, `multiple`, `pattern`, `autofocus`, `placeholder`, `required` y `step`. Existe además un nuevo atributo, `list`, que hace referencia a otro elemento, permitiendo crear un nuevo tipo de entrada de datos.

6.1. ATRIBUTO LIST Y <DATALIST>

La combinación del atributo `list` y un elemento de tipo `<datalist>` da como resultado un campo de texto, donde el usuario puede introducir cualquier contenido, y las opciones definidas en el `<datalist>` se muestran como una lista desplegable. Hay que tener en cuenta que la lista tiene que estar contenida en un elemento `<datalist>` cuyo `id` coincide con el indicado en el atributo `list`:

```
<input id="form-person-title" type="text" list="mylist">
<datalist id="mylist">
    <option label="Mr" value="Mr">
    <option label="Ms" value="Ms">
    <option label="Prof" value="Mad Professor">
</datalist>
```

En este ejemplo se utiliza un campo de tipo `text`, pero puede ser utilizado igualmente con campos de tipo `url` y `email`.

6.2. ATRIBUTO AUTOFOCUS

El atributo *booleano* `autofocus` permite definir que control va a tener el foco cuando la página se haya cargado. Hasta ahora, esto se conseguía a través de JavaScript, utilizando el método `.focus()` en un elemento concreto, al cargarse el documento. Ahora es el navegador el encargado de esta tarea, y puede comportarse de manera más inteligente, como no cambiando el foco de un elemento si el usuario ya se encuentra escribiendo en otro campo (éste era un problema común con JavaScript).
Únicamente debe existir un elemento con este atributo definido en el documento. Desde el punto de vista de la usabilidad, hay que utilizar este atributo con cuidado. Hay que utilizarlo únicamente cuando el control que recibe el foco es el elemento principal de la página, como en un buscador, por ejemplo.

6.3. ATRIBUTO PLACEHOLDER

Una pequeña mejora en la usabilidad de los formularios, suele ser colocar un pequeño texto de ayuda en algunos campos, de manera *discreta* y que desaparece cuando el usuario introduce algún dato. Como con el resto de elementos, hasta ahora era necesario utilizar JavaScript para realizar esta tarea, pero el atributo `placeholder` resuelve esta tarea.
Es importante recordar que este atributo no sustituye a la etiqueta `<label>`.

6.4. ATRIBUTO REQUIRED

Este atributo puede ser utilizado en un <textarea> y en la gran mayoría de los elementos <input> (excepto en los de tipo hidden, image o botones como submit). Cuando este atributo está presente, el navegador no permite el envío del formulario si el campo en concreto está vacío.

6.5. ATRIBUTO MULTIPLE

Este atributo permite definir que un campo puede admitir varios valores, como URLs o emails. Un uso muy interesante de este atributo es utilizarlo en conjunto con el campo `<input type="file">`, ya que de esta manera nos permite seleccionar varios ficheros que podemos enviar al servidor al mismo tiempo.

```
<input type="file" multiple="multiple">
```

6.6. ATRIBUTO AUTOCOMPLETE

Algunos navegadores suelen incluir alguna funcionalidad de autocompletado en algunos campos de formulario. A pesar de haber sido introducido recientemente en el estándar de HTML5, es una característica que lleva mucho tiempo siendo utilizada, concretamente desde la versión 5 de Internet Explorer.
Este atributo permite controlar el comportamiento del autocompletado en los campos de texto del formulario (que por defecto está activado).

6.7. ATRIBUTOS MIN Y MAX

Como hemos visto en el campo `<input type="number">`, estos atributos restringen los valores que pueden ser introducidos; no es posible enviar el formulario con un valor menor que `min` o un valor mayor que `max`. También es posible utilizarlo en otro tipo de campos como `date`, para especificar fechas mínimas o máximas.

```
<input type="date" min="2010-01-01" max="2010-12-31">
```

6.8. ATRIBUTO STEP

El atributo `step` controla los *pasos* por los que un campo aumenta o disminuye su valor. Si un usuario quiere introducir un porcentaje, pero queremos que sea en múltiplos de 5, lo haríamos de la siguiente manera:

```
<input type="range" mix="0" max="100" step="5">
```

6.9. ATRIBUTO PATTERN

Algunos de los tipos de `input` que hemos visto anteriormente (`email`, `number`, `url`...), son realmente expresiones regulares que el navegador evalúa cuando se introducen datos. El atributo `pattern` nos

permite definir una expresión regular que el valor del campo debe cumplir. Por ejemplo, si el usuario debe introducir un número seguido de tres letras mayúsculas, podríamos definir esta expresión regular:

```
<input pattern="[0-9][A-Z]{3}" name="part"
    title="A part number is a digit followed by three uppercase letters.">
```

La especificación indica que la sintaxis de la expresión regular que utilicemos debe coincidir con la utilizada en JavaScript.

6.10. ATRIBUTO FORM

Tradicionalmente, los campos de un formulario van incluidos dentro de la correspondiente etiqueta `<form>`. Si por la razón que fuese (principalmente diseño) un elemento tuviese que mostrarse apartado del resto de elementos del formulario, se hacía casi necesario incluir toda la página dentro de una etiqueta `<form>`.
Con HTML5, los elementos que hasta ahora era obligatorio que estuviesen contenidos dentro del elemento `<form>`, pueden colocarse en cualquier lugar de la página, siempre que los relacionemos con el formulario concreto a través del atributo `form` y el `id` de dicho formulario.

```
<form id="foo">
    <input type="text">
    ...
</form>
<textarea form="foo"></textarea>
```

En este caso, el elemento `<textarea>` se encuentra fuera del formulario, pero realmente pertenece a él ya que hemos definido el atributo `form` con el identificador del formulario.

7. Dataset

Gracias a HTML5, ahora tenemos la posibilidad de incorporar atributos de datos personalizados en todos los elementos HTML. Hasta la aparición de estos atributos, la manera de lograr un comportamiento similar (asociar datos a elementos), era incluir estos datos como clases CSS en los elementos, y acceder a ellos a través de jQuery, de una manera como la siguiente:

```
<input class="spaceship shields-5 lives-3 energy-75">
```

Una vez definidos los "atributos", era necesario acceder a estas clases y realizar un trabajo extra para extraer su nombre y su valor (convertir `energy-75` en `energy = 75`).

Afortunadamente, esto ya no es necesario, gracias a los atributos `dataset`. Estos nuevos atributos de datos personalizados constan de dos partes:

- Nombre del atributo: el nombre del atributo de datos debe ser de al menos un carácter de largo y debe tener el prefijo `data-`. No debe contener letras mayúsculas.
- Valor del atributo: el valor del atributo puede ser cualquier *string* o cadena. Con esta sintaxis, podemos añadir cualquier dato que necesitemos a nuestra aplicación, como se muestra a continuación:

```
<ul id="vegetable-seeds">
  <li data-spacing="10cm" data-sowing-time="March to June">Carrots</li>
  <li data-spacing="30cm" data-sowing-time="February to March">Celery</li>
  <li data-spacing="3cm" data-sowing-time="March to September">Radishes</li>
</ul>
```

Ahora podemos usar estos datos almacenados en nuestro sitio para crear una experiencia de usuario más rica y atractiva. Imagina que cuando un usuario hace clic en un "vegetable", una nueva capa se abre en el explorador que muestra la separación de semillas e instrucciones de siembra. Gracias a los atributos `data-` que hemos añadido a nuestros elementos `<li>`, ahora podemos mostrar esta información al instante sin tener que preocuparnos de hacer ninguna llamada `AJAX` y sin tener que hacer ninguna consulta a las bases de datos del servidor.

Prefijar los atributos personalizados con `data-` asegura que van a ser completamente ignorados por el agente de usuario. Por lo que al navegador y al usuario final de la web se refiere, no existe esta información.

Custom data attributes are intended to store custom data private to the page or application, for which there are no more appropriate attributes or elements. These attributes are not intended for use by software that is independent of the site that uses the attributes. Every HTML element may have any number of custom data attributes specified, with any value.

W3C Specification

Esto es, los atributos de datos personalizados están destinadas a almacenar los datos personalizados que son de interés exclusivamente para la página o la aplicación, para los que no hay atributos o elementos más apropiados. Estos atributos no están destinados para un uso externo (a través de un software independiente al sitio que los utiliza). Cada elemento HTML puede tener un número indefinido de atributos de datos personalizados, con cualquier valor.

7.1. UTILIZACION DE LOS DATA ATTRIBUTES

Como los atributos de datos personalizados son válidos en HTML5, pueden ser utilizados en cualquier navegador que soporte *HTML5 doctypes*. Estas son algunas de las formas en las que pueden ser utilizados:

- Para almacenar la altura inicial o la opacidad de un elemento que pudiera ser necesaria en los cálculos de animación JavaScript posteriores.
- Para almacenar los parámetros para una película de Flash que se carga a través de JavaScript.
- Para almacenar los datos estadísticos de una web.
- Para almacenar los datos acerca de la salud, munición o vida de un elemento en un juego JavaScript.
- Para poder añadir subtítulos a un `<video>`.

Y éstas algunas de las situaciones para las que no se deben usar los *data attributes*:

- Los atributos de datos personalizados no deben usarse si hay un atributo o elemento existente que es más adecuado.
- Éstos tampoco tienen la intención de competir con microformatos. En la especificación queda claro que los datos no están pensados para ser usados públicamente. El software externo no debe interactuar con ellos.
- La presencia/ausencia de un atributo de datos personalizado no se deben utilizar como una referencia para los estilos de CSS. Si se hace, podría sugerir que los datos que se están almacenando son de importancia inmediata para el usuario y se deberían marcar de una manera más accesible.

7.2. *DATA ATTRIBUTES* Y JAVASCRIPT

Ahora que comprendemos el funcionamiento de los atributos de datos personalizados y cuándo se utilizan, deberíamos centrarnos en cómo interactuar con ellos utilizando JavaScript.

Si quisiéramos recuperar o actualizar estos atributos utilizando JavaScript, podríamos hacerlo utilizando los métodos `getAttribute` y `setAttribute`.

```
<div id="strawberry-plant" data-fruit="12"></div>
<script>
// "Getting" data-attributes using getAttribute
var plant = document.getElementById("strawberry-plant");
```

```
var fruitCount = plant.getAttribute("data-fruit"); // fruitCount = "12"

// "Setting" data-attributes using setAttribute
plant.setAttribute("data-fruit","7"); // Pesky birds
</script>
```

Este método funcionará en todos los navegadores modernos, pero no es la manera en la que los *data attributes* deben ser utilizados. La mejor manera para lograr lo mismo es mediante el acceso a la propiedad `dataset` de un elemento. Al utilizar este método, en lugar de utilizar el nombre del atributo completo, se puede prescindir del prefijo `data-` y referirse a los atributos de datos personalizados utilizando directamente los nombres que se han asignado.

```
<div id="sunflower" data-leaves="47" data-plant-height="2.4m"></div>
<script>
// "Getting" data-attributes using dataset
var plant = document.getElementById("sunflower");
var leaves = plant.dataset.leaves; // leaves = 47;

// "Setting" data-attributes using dataset
var tallness = plant.dataset.plantHeight; // "plant-height" -> "plantHeight"
plant.dataset.plantHeight = "3.6m";  // Cracking fertiliser
</script>
```

Si en algún momento un atributo `data-` específico ya no es necesario, es posible eliminarlo por completo del elemento DOM estableciendo un valor nulo.

```
plant.dataset.leaves = null;
```

En conclusión, los *data attributes* personalizados son una buena manera de simplificar el almacenamiento de datos de la aplicación en las páginas web.

8. La API Forms

A muchos no les sorprenderá que los formularios de HTML5 cuenten con su propia API para personalizar todos los aspectos de procesamiento y validación.

Existen diferentes metodologías para aprovechar el proceso de validación en HTML5, a través de los tipos de campo para activar el proceso de validación por defecto (por ejemplo, email) , a través de atributos como required,… y, ahora, también se pueden crear tipos de campo especiales usando **pattern** para personalizar requisitos de validación. Sin embargo, cuando se trata de aplicar mecanismos complejos de validación (por ejemplo, combinando campos o comprobando los resultados de un cálculo) deberemos recurrir a nuevos recursos provistos por esta API.

8.1. SETCUSTOMVALIDITY()

Los navegadores que soportan HTML5 muestran un mensaje de error cuando el usuario intenta enviar un formulario que contiene un campo inválido.

Podemos crear mensajes para nuestros propios requisitos de validación usando el método setCustomValidity(mensaje).

Con este método especificamos un error personalizado que mostrará un mensaje cuando el formulario es enviado. Cuando un mensaje vacío es declarado, el error es anulado.

El siguiente código presenta una situación de validación compleja. El supuesto es que el formulario sólo será inválido cuando ambos campos se encuentran vacíos. El usuario necesitará ingresar sólo uno de los campos, o su nombre o su apellido, para validar la entrada.

Evidentemente, en casos como éste no es posible usar el atributo required debido a que no sabemos cuál campo el usuario decidirá utilizar. Sólo con código Javascript y errores personalizados podremos crear un mecanismo efectivo de validación para este escenario.

Nuestro código comienza a funcionar cuando el evento load es disparado. La función iniciar() será llamada para responder al evento. Esta función creará referencias para los dos elementos input y agregará una ‘listener’ para el evento input en ambos. Estas escuchas llamarán a la función validacion() cada vez que el usuario escribe dentro de los campos.

Debido a que los elementos input se encuentran vacíos cuando el documento se carga, deberemos declarar una condición inválida para no permitir que el usuario envíe el formulario antes de ingresar al menos uno de los valores. Por esta razón la función validacion() se ejecutará en el proceso de carga. Si ambos campos están vacíos se generará un error y el color de fondo del campo será cambiado a rojo. Sin embargo, si esta condición ya no es verdad porque al menos uno de los campos este completado, el error será eliminado y el color del fondo será restablecido a blanco.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Formularios</title>
        <script>
            function iniciar(){
                nombre1=document.getElementById("nombre");
                nombre2=document.getElementById("apellido");
                nombre1.addEventListener("input", validacion, false);
                nombre2.addEventListener("input", validacion, false);
                validacion();
            }
            function validacion(){
                if(nombre1.value=='' && nombre2.value==''){
                    nombre1.setCustomValidity('inserte al menos un nombre');
                    nombre1.style.background='#FFDDDD';
                } else {
                    nombre1.setCustomValidity('');
                    nombre1.style.background='#FFFFFF';
                }
            }
            window.addEventListener("load", iniciar, false);
        </script>
    </head>
    <body>
        <section>
            <form name="registracion" method="get">
                Nombre:
                <input type="text" name="nombre" id="nombre" />
                Apellido:
                <input type="text" name="apellido" id="apellido" />
                <input type="submit" id="send" value="ingresar" />
            </form>
        </section>
    </body>
</html>
```

8.2. EL EVENTO INVALID

Cada vez que el usuario envía un formulario, el evento invalid es disparado si se detecta la condición de invalidación. Se puede agregar una 'listener' para este evento y así ofrecer una respuesta personalizada, como en el siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Formularios</title>
        <script>
            function iniciar(){
                edad=document.getElementById("miedad");
                edad.addEventListener("change", cambiarrango, false);
                document.informacion.addEventListener("invalid", validacion, true);
                document.getElementById("enviar").addEventListener("click", enviar, false);
            }
            function cambiarrango(){
                var salida=document.getElementById("rango");
                var calc=edad.value-20;
                if(calc<20){
                    calc=0;
                    edad.value=20;
                }
                salida.innerHTML=calc+' a '+edad.value;
            }
            function validacion(e){
                var elemento=e.target;
                elemento.style.background='#FFDDDD';
            }
            function enviar(){
                var valido=document.informacion.checkValidity();
                if(valido){ document.informacion.submit(); }
            }
            window.addEventListener("load", iniciar, false);
        </script>
    </head>
    <body>
        <section>
            <form name="informacion" method="get">
                Usuario:
                <input pattern="[A-Za-z]{3,}" name="usuario" id="usuario" maxlength="10" required />
                Email:
                <input type="email" name="miemail" id="miemail" required>
                Rango de Edad:
                <input type="range" name="miedad" id="miedad" min="0" max="80" step="20" value="20"/>
                <output id="rango">0 a 20</output>
                <input type="button" id="enviar" value="ingresar"/>
            </form>
        </section>
    </body>
</html>
```

El campo usuario tiene tres atributos para validación: el atributo pattern sólo admite el ingreso de un texto de tres caracteres mínimo, desde la A a la Z (mayúsculas o minúsculas), el atributo maxlength limita la entrada a 10 caracteres máximo, y el atributo required invalida el campo si está vacío.

El campo miemail cuenta con sus limitaciones naturales debido a su tipo y además no podrá enviarse vacío gracias al atributo required.

El campo miedad usa los atributos min, max, step y value para configurar las condiciones del rango.

El elemento **output** se utiliza para mostrar en pantalla una referencia del rango seleccionado.

Cuando el usuario haga clic en el botón "ingresar", un evento **invalid** será disparado desde cada campo inválido y el color de fondo de esos campos será cambiado a rojo por la función validacion().

8.3. VALIDACIÓN EN TIEMPO REAL

Si ejecutásemos en el navegador el código anterior, observaríamos que no existe una validación en tiempo real. Los campos son sólo validados cuando el botón "ingresar" es presionado. Para hacer más práctico nuestro sistema personalizado de validación, podemos aprovechar los atributos provistos por el objeto **ValidityState**.

```
function iniciar(){
    edad=document.getElementById("miedad");
    edad.addEventListener("change", cambiarrango, false);
    document.informacion.addEventListener("invalid",
    validacion, true);
    document.getElementById("enviar").addEventListener("click",
    enviar, false);
    document.informacion.addEventListener("input", controlar, false);
}
function controlar(e){
    var elemento=e.target;
    if(elemento.validity.valid){
        elemento.style.background='#FFFFFF';
    }else{
        elemento.style.background='#FFDDDD';
    }
}
```

En este código se ha añadido un nuevo 'listener' que envía cualquier cambio a la función controlar(). Esta función aprovecha la propiedad target para crear una referencia hacia el elemento que disparó el evento input. La validez del campo será controlada por medio del estado **valid** provisto por el atributo validity en la construcción elemento.validity.valid.

El estado **valid** será true (verdadero) si el elemento es válido y false (falso) si no lo es. Utilizando este valor booleano podremos establecer el color del fondo del elemento.

8.4. PROPIEDADES DE VALIDACIÓN

Podemos encontrar ocho posibles estados de validez para las diferentes condiciones:

8.4.1. valueMissing

Este estado es true (verdadero) cuando el atributo required fue declarado y el campo está vacío.

8.4.2. typeMismatch

Este estado es verdadero cuando la sintaxis de la entrada no corresponde con el tipo especificado como por ejemplo cuando el texto insertado en un tipo de campo email no es una dirección de email válida.

8.4.3. patternMismatch

Este estado es verdadero cuando la entrada no corresponde con el patrón provisto por el atributo **pattern**.

8.4.4. tooLong

Este estado verdadero cuando el atributo maxlength fue declarado y la entrada es más extensa que el valor especificado para este atributo.

8.4.5. rangeUnderflow

Este estado es true (verdadero) cuando el atributo min fue declarado y la entrada es menor que el valor especificado para este atributo.

8.4.6. rangeOverflow

Esta estado es true (verdadero) cuando el atributo max fue declarado y la entrada es más grande que el valor especificado para este atributo.

8.4.7. stepMismatch

Este estado es true (verdadero) cuando el atributo step fue declarado y su valor no corresponde con los valores de atributos como min, max y value.

8.4.8. customError

Este estado es true (verdadero) cuando declaramos un error personalizado usando el método setCustomValidity() estudiado anteriormente. Para controlar estos estados de validación, debemos utilizar la sintaxis elemento.validity.

8.5. WILLVALIDATE

En aplicaciones dinámicas es posible que los elementos involucrados no tengan que ser validados. Este puede ser el caso, por ejemplo, con botones, campos ocultos o elementos como <output>. La API nos ofrece la posibilidad de detectar esta condición usando el atributo willValidate y la sintaxis elemento.willValidate.

9. La API Canvas

Esta API ofrece una de las más poderosas características de HTML5. Permite a los desarrolladores trabajar con un medio visual e interactivo para proveer capacidades de aplicaciones de escritorio a la web.

Complementos o plug-ins, como Flash o Java applets, por ejemplo, ya están en desuso y están siendo remplazados por esta tecnología.

La API Canvas se hace cargo del aspecto gráfico y lo hace de una forma extremadamente efectiva. Canvas nos permite dibujar, presentar gráficos en pantalla, animar y procesar imágenes y texto, y trabaja junto con el resto de la especificación para crear aplicaciones completas e incluso video juegos en 2 y 3 dimensiones para la web.

9.1. EL ELEMENTO CANVAS

Este elemento genera un espacio rectangular vacío en la página web (lienzo) en el cual se mostrarán los resultados de ejecutar los métodos provistos por la API Canvas. Cuando se crea, se produce sólo un espacio en blanco, como un elemento DIV vacío, pero con un propósito totalmente diferente.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Canvas API</title>
        <script src="canvas.js"></script>
    </head>
    <body>
        <section id="cajalienzo">
            <canvas id="lienzo" width="500" height="300">
                Su navegador no soporta el elemento canvas
            </canvas>
        </section>
    </body>
</html>
```

El elemento CANVAS tiene dos propiedades, **width** y **height** y dos métodos **getContext()** y **toDataURL()**.

Los atributos **width** (ancho) y **height** (alto) declaran el tamaño del lienzo en pixeles. Estos atributos son necesarios debido a que todo lo que sea dibujado sobre el elemento tendrá esos valores como referencia. Al atributo **id**, como en otros casos, nos facilita el acceso al elemento desde el código Javascript.

9.2. GETCONTEXT()

El método getContext() es el primer método que tenemos que llamar para dejar al elemento CANVAS listo para trabajar. Genera un contexto de dibujo que será asignado al lienzo. A través de la referencia que retorna podremos aplicar el resto de la API.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
}
window.addEventListener("load", iniciar, false);
```

El método getContext() puede tomar dos valores: **2d** y **3d**. Esto es, por supuesto, para ambientes de 2 y 3 dimensiones. Por el momento sólo el contexto 2d está disponible, pero serios esfuerzos están siendo volcados en el desarrollo de una API estable en 3 dimensiones.

El contexto de dibujo del lienzo será una rejilla de pixeles listados en filas y columnas de arriba a abajo e izquierda a derecha, con su origen (el pixel 0,0) ubicado en la esquina superior izquierda del lienzo.

9.3. DIBUJANDO RECTÁNGULOS

Normalmente deberemos preparar la figura en el contexto, pero existen algunos métodos que nos permiten dibujar directamente en el lienzo, sin preparación previa. Estos métodos son específicos para formas rectangulares y son los únicos que generan una forma primitiva (para obtener otras formas tendremos que combinar otras técnicas de dibujo y trazados complejos). Los métodos disponibles son los siguientes:

9.3.1. fillRect(x, y, ancho, alto)

Este método dibuja un rectángulo sólido. La esquina superior izquierda será ubicada en la posición especificada por los atributos x e y. Los atributos ancho y alto declaran el tamaño.

9.3.2. strokeRect(x, y, ancho, alto)

Similar al método anterior, éste dibujará un rectángulo vacío (solo su contorno).

9.3.3. clearRect(x, y, ancho, alto)

Esta método es usado para substraer pixeles del área especificada por sus atributos. Es un borrador rectangular.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.strokeRect(100,100,120,120);
    lienzo.fillRect(110,110,100,100);
    lienzo.clearRect(120,120,80,80);
}
window.addEventListener("load", iniciar, false);
```

El primer método usado en la función, strokeRect(100,100,120,120), dibuja un rectángulo vacío con la esquina superior izquierda en la posición 100,100 y un tamaño de 120 pixeles (este es un cuadrado de 120 pixeles). El segundo método, fillRect(110,110, 100,100), dibuja un rectángulo sólido, esta vez comenzando desde la posición 110,110 del lienzo. Y finalmente, con el último método, clearRect(120,120,80,80), un recuadro de 80 pixeles es substraído del centro de la figura.

9.4. COLORES

Hasta el momento hemos usado el color otorgado por defecto, el negro, pero podemos especificar el color que queremos aplicar mediante sintaxis CSS utilizando las siguientes propiedades:

9.4.1. strokeStyle

Esta propiedad declara el color para el contorno de la figura.

9.4.2. fillStyle

Esta propiedad declara el color para el interior de la figura.

9.4.3. globalAlpha

Esta propiedad no es para definir color sino transparencia. Especifica la transparencia para todas las figuras dibujadas en el lienzo.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.fillStyle="#000099";
    lienzo.strokeStyle="#990000";
    lienzo.strokeRect(100,100,120,120);
    lienzo.fillRect(110,110,100,100);
    lienzo.clearRect(120,120,80,80);
}
```

9.5. GRADIENTES

Gradientes son una herramienta esencial en cualquier programa de dibujo estos días, y esta API no es la excepción. Así como en CSS3, los gradientes en la API Canvas pueden ser lineales o radiales, y pueden incluir puntos de terminación para combinar colores.

9.5.1. createLinearGradient(x1, y1, x2, y2)

Este método crea un objeto que luego será usado para aplicar un gradiente lineal al lienzo.

9.5.2. createRadialGradient(x1, y1, r1, x2, y2, r2)

Este método crea un objeto que luego será usado para aplicar un gradiente circular o radial al lienzo usando dos círculos. Los valores representan la posición del centro de cada círculo y sus radios.

9.5.3. addColorStop(posición, color)

Este método especifica los colores a ser usados por el gradiente. El atributo posición es un valor entre 0.0 y 1.0 que determina dónde la degradación comenzará para ese color en particular.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    var gradiente=lienzo.createLinearGradient(0,0,10,100);
    gradiente.addColorStop(0.5, '#0000FF');
    gradiente.addColorStop(1, '#000000');
    lienzo.fillStyle=gradiente;
    lienzo.fillRect(10,10,100,100);
    lienzo.fillRect(150,10,200,100);
}
```

9.6. CREANDO TRAZADOS

Los métodos estudiados hasta el momento dibujan directamente en el lienzo, pero ese no es siempre el caso. Normalmente tendremos que procesar figuras en segundo plano y una vez que el trabajo esté hecho enviar el resultado al contexto para que sea dibujado. Con este propósito, API Canvas introduce varios métodos con los que podremos generar trazados.

Un trazado es como un mapa a ser seguido por el lápiz. Una vez declarado, el trazado será enviado al contexto y dibujado de forma permanente en el lienzo. El trazado puede incluir diferentes tipos de líneas, como líneas rectas, arcos, rectángulos, entre otros, para crear figuras complejas.

Existen dos métodos para comenzar y cerrar el trazado:

9.6.1. beginPath()

Este método comienza la descripción de una nueva figura. Es llamado en primer lugar, antes de comenzar a crear el trazado.

9.6.2. closePath()

Este método cierra el trazado generando una línea recta desde el último punto hasta el punto de origen. Puede ser ignorado cuando utilizamos el método fill() para dibujar el trazado en el lienzo.

También contamos con tres métodos para dibujar el trazado en el lienzo:

9.6.3. stroke()

Este método dibuja el trazado como una figura vacía (solo el contorno).

9.6.4. fill()

Este método dibuja el trazado como una figura sólida. Cuando usamos este método no necesitamos cerrar el trazado con closePath(), el trazado es automáticamente cerrado con una línea recta trazada desde el punto final hasta el origen.

9.6.5. clip()

Este método declara una nueva área de corte para el contexto. Cuando el contexto es inicializado, el área de corte es el área completa ocupada por el lienzo. El método clip() cambiará el área de corte a una nueva forma creando de este modo una máscara. Todo lo que caiga fuera de esa máscara no será dibujado.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.beginPath();
    // aquí va el trazado
    lienzo.stroke();
}
```

Para crear el trazado y la figura real que será enviada al contexto y dibujada en el lienzo, contamos con varios métodos disponibles:

9.6.6. moveTo(x, y)

Este método mueve el lápiz a una posición específica para continuar con el trazado. Nos permite comenzar o continuar el trazado desde diferentes puntos, evitando líneas continuas.

9.6.7. lineTo(x, y)

Este método genera una línea recta desde la posición actual del lápiz hasta la nueva declarada por los atributos x e y.

9.6.8. rect(x, y, ancho, alto)

Este método genera un rectángulo. A diferencia de los métodos estudiados anteriormente, éste generará un rectángulo que formará parte del trazado (no directamente dibujado en el lienzo). Los atributos tienen la misma función.

9.6.9. arc(x, y, radio, ángulo inicio, ángulo final, dirección)

Este método genera un arco o un círculo en la posición x e y, con un radio y desde un ángulo declarado por sus atributos. El último valor es un valor booleano (falso o verdadero) para indicar la dirección a favor o en contra de las agujas del reloj.

9.6.10. quadraticCurveTo(cpx, cpy, x, y)

Este método genera una curva Bézier cuadrática desde la posición actual del lápiz hasta la posición declarada por los atributos x e y. Los atributos cpx y cpy indican el punto que dará forma a la curva.

9.6.11. bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y)

Este método es similar al anterior pero agrega dos atributos más para generar una curva Bézier cúbica. Ahora disponemos de dos puntos para moldear la curva, declarados por los atributos cp1x, cp1y, cp2x y cp2y.

Veamos un trazado sencillo para entender cómo funcionan:

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.beginPath();
    lienzo.moveTo(100,100);
    lienzo.lineTo(200,200);
    lienzo.lineTo(100,200);
    lienzo.stroke();
}
```

9.7. TEXTO

Escribir texto en el lienzo es tan simple como definir unas pocas propiedades y llamar al método apropiado. Tres propiedades son ofrecidas para configurar texto:

9.7.1. font

Esta propiedad tiene una sintaxis similar a la propiedad font de CSS, y acepta los mismos valores.

9.7.2. textAlign

Esta propiedad alinea el texto. Existen varios valores posibles: start (comienzo), end (final), left (izquierda), right (derecha) y center (centro).

9.7.3. textBaseline

Esta propiedad es para alineamiento vertical. Establece diferentes posiciones para el texto (incluyendo texto Unicode). Los posibles valores son: top, hanging, middle, alphabetic, ideographic y bottom.

Dos métodos están disponibles para dibujar texto en el lienzo:

9.7.4. strokeText(texto, x, y)

Del mismo modo que el método stroke() para el trazado, este método dibujará el texto especificado en la posición x,y como una figura vacía (sólo los contornos). Puede también incluir un cuarto valor para declarar el tamaño máximo. Si el texto es más extenso que este último valor, será encogido para caber dentro del espacio establecido.

9.7.5. fillText(texto, x, y)

Este método es similar al método anterior excepto que esta vez el texto dibujado será sólido (igual que la función para el trazado).

9.7.6. measureText()

Este método retorna información sobre el tamaño de un texto específico. Puede ser útil para combinar texto con otras formas en el lienzo y calcular posiciones o incluso colisiones en animaciones.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.font="bold 24px verdana, sans-serif";
    lienzo.textAlign="start";
    lienzo.fillText("Mi mensaje", 100,100);
}
```

Como se puede ver en ejemplo, la propiedad **font** puede tomar varios valores a la vez, usando exactamente la misma sintaxis que CSS. La propiedad textAling hace que el texto sea dibujado desde la posición 100,100 (si el valor de esta propiedad fuera end, por ejemplo, el texto terminaría en la posición 100,100). Finalmente, el método fillText dibuja un texto sólido en el lienzo.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.font="bold 24px verdana, sans-serif";
    lienzo.textAlign="start";
    lienzo.textBaseline="bottom";
    lienzo.fillText("Mi mensaje", 100,124);
    var tamano=lienzo.measureText("Mi mensaje");
    lienzo.strokeRect(100,100,tamano.width,24);
}
```

En este ejemplo agregamos un alineamiento vertical. La propiedad textBaseline se establece a bottom (inferior), lo que significa que la base o parte inferior del texto estará ubicada en la posición 124. Esto nos ayudará a conocer la posición vertical exacta del texto en el lienzo.

Usando el método measureText() y la propiedad width (ancho) obtenemos el tamaño horizontal del texto. Con esta medida estamos listos para dibujar un rectángulo que rodeará al texto.

9.8. SOMBRAS

Por supuesto, sombras son también una parte importante de Canvas API. Podemos generar sombras para cada trazado e incluso textos. La API provee cuatro propiedades para hacerlo:

9.8.1. shadowColor

Esta propiedad declara el color de la sombra usando sintaxis CSS.

9.8.2. shadowOffsetX

Esta propiedad recibe un número para determinar qué tan lejos la sombra estará ubicada del objeto (dirección horizontal).

9.8.3. shadowOffsetY

Esta propiedad recibe un número para determinar qué tan lejos la sombra estará ubicada del objeto (dirección vertical).

9.8.4. shadowBlur

Esta propiedad produce un efecto de difuminación para la sombra.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.shadowColor="rgba(0,0,0,0.5)";
    lienzo.shadowOffsetX=4;
    lienzo.shadowOffsetY=4;
    lienzo.shadowBlur=5;
    lienzo.font="bold 50px verdana, sans-serif";
    lienzo.fillText("Mi mensaje ", 100,100);
}
```

La sombra creada en este ejemplo usa la función rgba() para obtener un color negro semitransparente. Es desplazada 4 pixeles del objeto y tiene un valor de difuminación de 5.

9.9. TRANSFORMACIONES

LA API Canvas ofrece operaciones complejas que es posible aplicar sobre el lienzo para afectar los gráficos que luego son dibujados en él. Estas operaciones son realizadas utilizando cinco métodos de transformación diferentes, cada uno para un propósito específico.

9.9.1. translate(x, y)

Este método de transformación es usado para mover el origen del lienzo. Cada lienzo comienza en el punto 0,0 localizado en la esquina superior izquierda, y los valores se incrementan en cualquier dirección dentro del lienzo. Valores negativos caen fuera del lienzo. A veces es bueno poder usar valores negativos para crear figuras complejas. El método translate() nos permite mover el punto 0,0 a una posición específica para usar el origen como referencia para nuestros dibujos o para aplicar otras transformaciones.

9.9.2. rotate(ángulo)

Este método de transformación rotará el lienzo alrededor del origen tantos ángulos como sean especificados.

9.9.3. scale(x, y)

Este método de transformación incrementa o disminuye las unidades de la grilla para reducir o ampliar todo lo que esté dibujado en el lienzo. La escala puede ser cambiada independientemente para el valor horizontal o vertical usando los atributos x e y. Los valores pueden ser negativos, produciendo un efecto de espejo. Por defecto los valores son iguales a 1.0.

9.9.4. transform(m1, m2, m3, m4, dx, dy)

El lienzo contiene una matriz de valores que especifican sus propiedades. El método transform() aplica una nueva matriz sobre la actual para modificar el lienzo.

9.9.5. setTransform(m1, m2, m3, m4, dx, dy)

Este método reinicializa la actual matriz de transformación y establece una nueva desde los valores provistos en sus atributos.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.font="bold 20px verdana, sans-serif";
    lienzo.fillText("PRUEBA",50,20);
    lienzo.translate(50,70);
    lienzo.rotate(Math.PI/180*45);
    lienzo.fillText("PRUEBA",0,0);
    lienzo.rotate(-Math.PI/180*45);
    lienzo.translate(0,100);
    lienzo.scale(2,2);
    lienzo.fillText("PRUEBA",0,0);
}
```

No hay mejor forma de entender cómo funcionan las transformaciones que usarlas en nuestro código. En ejemplo anterior, aplicamos los métodos **translate()**, **rotate()** y **scale()** al mismo texto. Primero dibujamos un texto en el lienzo con la configuración por defecto. El texto aparecerá en la posición 50,20 con un tamaño de 20 pixeles. Luego de esto, usando translate(), el origen del lienzo es movido a la posición 50,70 y el lienzo completo es rotado 45 grados con el método rotate(). Otro texto se dibuja en el nuevo origen, con una inclinación de 45 grados. Las transformaciones aplicadas se vuelven los valores por defecto, por lo tanto antes de aplicar el siguiente método scale() rotamos el lienzo 45 grados negativos para ubicarlo en su posición original. Realizamos una transformación más moviendo el origen otros 100 pixeles hacia abajo. Finalmente, la escala del lienzo es duplicada y un nuevo texto es dibujado al doble del tamaño de los anteriores.

Cada transformación es acumulativa. Si realizamos dos transformaciones usando scale(), por ejemplo, el segundo método realizará el escalado considerando el estado actual del lienzo. Una orden scale(2,2) luego de otra scale(2,2) cuadruplicará la escala del lienzo. Y para los métodos de transformación de la matriz, esta no es una excepción. Es por esto que contamos con dos métodos para realizar esta clase de transformaciones: transform() y setTransform().

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.transform(3,0,0,1,0,0);
    lienzo.font="bold 20px verdana, sans-serif";
    lienzo.fillText("PRUEBA",20,20);
    lienzo.transform(1,0,0,10,0,0);
    lienzo.font="bold 20px verdana, sans-serif";
```

```
    lienzo.fillText("PRUEBA",100,20);
}
```

En el código anterior aplicamos varios métodos de transformación sobre el mismo texto para comparar efectos. Los valores por defecto de la matriz del lienzo son 1,0,0,1,0,0. Cambiando el primer valor a 3, en la primera transformación de nuestro ejemplo arriba, estiramos el lienzo horizontalmente. El texto dibujado luego de esta transformación será más ancho que en condiciones por defecto.

Con la siguiente transformación, el lienzo se estira verticalmente cambiando el cuarto valor a 10 y preservando los anteriores.

Un detalle importante a recordar es que las transformaciones son aplicadas sobre la matriz declarada en previas transformaciones, por lo que el segundo texto será igual de ancho que el anterior (será reajustado tanto horizontal como verticalmente). Para reinicializar la matriz y declarar nuevos valores de transformación, podemos usar el método setTransform().

9.10. RESTAURANDO EL ESTADO

La acumulación de transformaciones hace realmente difícil volver a anteriores estados. En el código del Listado 7-18, por ejemplo, tuvimos que recordar el valor de rotación usado previamente para poder realizar una nueva rotación y volver el lienzo al estado original.

Considerando situaciones como ésta, Canvas API provee dos métodos para grabar y recuperar el estado del lienzo.

9.10.1. save()

Este método graba el estado del lienzo, incluyendo transformaciones ya aplicadas, valores de propiedades de estilo y la actual máscara (el área creada por el método clip(), si existe).

9.10.2. restore()

Este método recupera el último estado grabado.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.save();
    lienzo.translate(50,70);
    lienzo.font="bold 20px verdana, sans-serif";
    lienzo.fillText("PRUEBA1",0,30);
    lienzo.restore();
    lienzo.fillText("PRUEBA2",0,30);
}
```

9.11. GLOBALCOMPOSITEOPERATION

Cuando hablamos de trazados se dijo que existía una propiedad para determinar cómo una figura es posicionada y combinada con otras figuras dibujadas previamente en el lienzo. La propiedad es globalCompositeOperation y su valor por defecto es source-over, lo que significa que la nueva figura será dibujada sobre las que ya existen en el lienzo pero, esta propiedad ofrece 11 valores más:

9.11.1. source-in

Solo la parte de la nueva figura que se sobrepone a las figuras previas es dibujada. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.

9.11.2. source-out

Solo la parte de la nueva figura que no se sobrepone a las figuras previas es dibujada. El resto de la figura, e incluso el resto de las figuras previas, se vuelven transparentes.

9.11.3. source-atop

Solo la parte de la nueva figura que se superpone con las figuras previas es dibujada. Las figuras previas son preservadas, pero el resto de la nueva figura se vuelve transparente.

9.11.4. lighter

Ambas figuras son dibujadas (nueva y vieja), pero el color de las partes que se superponen es obtenido adicionando los valores de los colores de cada figura.

9.11.5. xor

Ambas figuras son dibujadas (nueva y vieja), pero las partes que se superponen se vuelven transparentes.

9.11.6. destination-over

Este es el opuesto del valor por defecto. Las nuevas figuras son dibujadas detrás de las viejas que ya se encuentran en el lienzo.

9.11.7. destination-in

Las partes de las figuras existentes en el lienzo que se superponen con la nueva figura son preservadas. El resto, incluyendo la nueva figura, se vuelven transparentes.

9.11.8. destination-out

Las partes de las figuras existentes en el lienzo que no se superponen con la nueva figura son preservadas. El resto, incluyendo la nueva figura, se vuelven transparentes.

9.11.9. destination-atop

Las figuras existentes y la nueva son preservadas solo en la parte en la que se superponen.

9.11.10. darker

Ambas figuras son dibujadas, pero el color de las partes que se superponen es determinado substrayendo los valores de los colores de cada figura.

9.11.11. copy

Solo la nueva figura es dibujada. Las ya existentes se vuelven transparentes.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    lienzo.fillStyle="#990000";
    lienzo.fillRect(100,100,300,100);
    lienzo.globalCompositeOperation="destination-atop";
    lienzo.fillStyle="#AAAAFF";
    lienzo.font="bold 80px verdana, sans-serif";
    lienzo.textAlign="center";
    lienzo.textBaseline="middle";
    lienzo.fillText("PRUEBA",250,110);
}
```

9.12. PROCESANDO IMÁGENES

API Canvas no sería nada sin la capacidad de procesar imágenes. Pero incluso cuando las imágenes son un elemento sumamente importante para una aplicación gráfica, sólo un método nativo se ha provisto para trabajar con ellas.

9.12.1. drawImage()

El método drawImage() es el único a cargo de dibujar una imagen en el lienzo. Sin embargo, este método puede recibir un número variable de parámetros que producen diferentes resultados.

drawImage(objeto_image, x, y)

Esta sintaxis es para dibujar una imagen en el lienzo en la posición declarada por x e y. El primer valor es una referencia a la imagen que será dibujada.

drawImage(objeto_image, x, y, ancho, alto)

Esta sintaxis nos permite escalar la imagen antes de dibujarla en el lienzo, cambiando su tamaño con los valores de los atributos ancho y alto.

drawImage(objeto_image, x1, y1, ancho1, alto1, x2, y2, ancho2, alto2)

Esta es la sintaxis más compleja. Hay dos valores para cada parámetro. El propósito es cortar partes de la imagen y luego dibujarlas en el lienzo con un tamaño y una posición específicos. Los valores x1 e y1 declaran la esquina superior izquierda de la parte de la imagen que será cortada. Los valores ancho1 y alto1 indican el tamaño de esta pieza. El resto de los valores (x2, y2, ancho2 y alto2) declaran el lugar donde la pieza será dibujada en el lienzo y su nuevo tamaño (el cual puede ser igual o diferente al original).

En cada caso, el primer atributo puede ser una referencia a una imagen en el mismo documento generada por métodos como getElementById(), o creando un nuevo objeto imagen usando métodos regulares de Javascript. No es posible usar una URL o cargar un archivo desde una fuente externa directamente con este método.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    var imagen = new Image();
    image.src="http://www.images.com/image01.png";
    image.addEventListener("load", function(){ lienzo.drawImage(imagen, 20, 20) }, false);
}
window.addEventListener("load", iniciar, false);
```

Comencemos con un simple ejemplo. El código anterior lo único que hace es cargar la imagen y dibujarla en el lienzo. Debido a que el lienzo sólo puede dibujar imágenes que ya están completamente cargadas, necesitamos controlar esta situación escuchando el evento load. Agregamos un listener para este evento y declaramos una función anónima para responder al mismo. El método drawImage(), dentro de esta función, dibujará la imagen cuando esté completamente cargada.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    var imagen = new Image();
    image.src="http://www.images.com/image01.png";
    image.addEventListener("load", function(){
          lienzo.drawImage(imagen, 0, 0, elemento.width, element.height)
    }, false);
}
window.addEventListener("load", iniciar, false);
```

En el ejemplo anterior se están utilizando los parámetros para cambiar el tamaño de la imagen. Las propiedades **width** y **height** retornan las medidas del lienzo, por lo que la imagen será estirada por este código hasta cubrir el lienzo por completo.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    var imagen = new Image();
    image.src="http://www.images.com/image01.png";
    image.addEventListener("load", function(){
          lienzo.drawImage(imagen, 135, 30, 50, 50, 0, 0, 200, 200)
    }, false);
}
window.addEventListener("load", iniciar, false);
```

El código anterior presenta la sintaxis más compleja del método drawImage(). Provee los nueve parámetros para obtener una parte de la imagen original, cambiar su tamaño y luego dibujarla en el lienzo. Tomamos un cuadrado de la imagen original desde la posición 135,50, con un tamaño de 50x50px. Este bloque, posteriormente, será redimensionado a 200x200px y finalmente será dibujado en el lienzo en la posición 0,0.

9.12.2. Datos de imágenes

Cuando se afirmó previamente que **drawImage()** era el único método disponible para dibujar imágenes en el lienzo, se "mintió suavemente". Existen poderosos métodos para procesar imágenes que, además, permiten dibujarlas en el lienzo. Debido a que estos métodos no trabajan con imágenes sino con datos, la declaración previa sigue siendo legítima.

Toda imagen puede ser representada por una sucesión de números enteros representados a través de valores RGBA (cuatro valores hexadecimales para cada pixel en formato Rojo/Verde/Azul/Transparencia). El grupo de valores con esta información se insertará en un array unidimensional que puede ser usado luego para generar una imagen. La API Canvas ofrece tres métodos para manipular datos y procesar imágenes de este modo:

9.12.3. getImageData(x, y, ancho, alto)

Este método toma un rectángulo del lienzo del tamaño declarado por sus atributos y lo convierte en datos. Retorna un objeto que puede ser luego accedido por sus propiedades width, height y data.

9.12.4. putImageData(datos, x, y)

Este método convierte los datos proporcionados en el parámetro **datos** en una imagen y la dibuja en el lienzo en la posición especificada por los parámetros **x** e **y**. Este es el opuesto a getImageData().

9.12.5. createImageData(ancho, alto)

Este método crea datos para representar una imagen vacía. Todos sus pixeles serán RGBA (0, 0, 0, 0), es decir, de color negro transparente. Puede también recibir datos como atributo, las dimensiones tomadas de los datos provistos para crear la imagen.

La posición de cada valor en el array se calcula con la fórmulas:

- Pixel Rojo: **(ancho × 4 × y) + (x × 4)**.
- Pixel Verde: **(ancho × 4 × y) + (x × 4) + 1**.
- Pixel Azul: **(ancho × 4 × y) + (x × 4) + 2**
- Transparencia: **(ancho × 4 × y) + (x × 4) + 3**

IMPORTANTE: Debido a restricciones de seguridad, no se puede extraer información del elemento CANVAS después de que una imagen tomada desde una fuente externa sea dibujada en el lienzo. Sólo cuando el documento y la imagen pertenezcan al mismo origen de datos o dominio y las directrices CORS lo permitan el método getImageData() trabajará adecuadamente.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    var imagen=new Image();
    imagen.src="logo.jpg";
    imagen.addEventListener("load", modificarimagen, false);
}
function modificarimagen(e){
    imagen=e.target;
    lienzo.drawImage(imagen,0,0);
    var info=lienzo.getImageData(0,0,175,262);
    var pos;
    for(x=0;x<=175;x++){
        for(y=0;y<=262;y++){
            pos=(info.width*4*y)+(x*4);
            info.data[pos]=255-info.data[pos];
            info.data[pos+1]=255-info.data[pos+1];
            info.data[pos+2]=255-info.data[pos+2];
        }
    }
    lienzo.putImageData(info,0,0);
}
window.addEventListener("load", iniciar, false);
```

Esta vez se ha creado una nueva función no anónima para procesar la imagen a posteriori. Primero, la función modificarimagen() genera una referencia a la imagen aprovechando la propiedad **target** extraída del evento generado por evento **load**. En el siguiente paso, utilizando esa referencia y el método drawImage(), la imagen se dibuja en el lienzo sobre la posición 0,0.

La imagen utilizada en nuestro ejemplo tiene un tamaño de 350x262px, por lo que usando el método getImageData() con los valores 0,0 para la esquina superior izquierda y 175x262 para el valor horizontal y

vertical, estamos extrayendo solo la mitad izquierda de la imagen original. Estos datos son grabados dentro de la variable **info**.

Una vez que esta información sea recolectada, es el momento de manipular cada pixel para obtener el resultado que queremos (en nuestro ejemplo esto será el efecto de negativo de este trozo de la imagen).

Debido a que cada color es declarado por un valor entre 0 y 255, el valor negativo debe ser obtenido restando el valor real a su máximo valor posible a través de la fórmula **color=255-color**. Para hacerlo con cada pixel de la imagen, se deberán crear dos bucles for (uno para las columnas y otro para las filas) para obtener cada color original y calcular el valor del negativo correspondiente. El bucle for para los valores x irá desde 0 a 175 (el ancho de la parte de la imagen que extrajimos del lienzo) y el for para los valores y irá desde 0 a 262 (el tamaño vertical de la imagen y también el tamaño vertical del trozo de imagen que estamos procesando).

Después de que cada pixel esté procesado, la variable **info** tendrá los datos de la imagen que debe ser enviada al lienzo como una imagen diferente usando el método **putImageData()**. La imagen será ubicada en la misma posición que la original, reemplazando la mitad izquierda de la imagen original por el negativo que se acaba de crear.

El método getImageData() retorna un objeto que puede ser procesado a través de sus propiedades (width, height y data) o puede ser usado íntegro por el método putImageData().

9.12.6. createPattern(imágen, tipo)

Los patrones son simples adiciones que pueden mejorar nuestros trazados. Con esta herramienta podemos agregar textura a nuestras figuras utilizando una imagen. El procedimiento es similar a la creación de gradientes; los patrones son creados por el método createPattern() y luego aplicados al trazado como si fuesen un color.

El atributo imágen es una referencia a la imagen que vamos a usar como patrón, y tipo configura el patrón por medio de cuatro valores: repeat, repeat-x, repeat-y y no-repeat.

```
function iniciar(){
    var lienzo=document.getElementById('lienzo').getContext('2d');
    var imagen=new Image();
    imagen.src="logo.jpg";
    imagen.addEventListener("load", modificarimagen, false);
}
function modificarimagen(e){
    var patron=lienzo.createPattern(e.target,'repeat');
    lienzo.fillStyle=patron;
    lienzo.fillRect(0,0,500,300);
}
window.addEventListener("load", iniciar, false);
```

9.13. TODATAURL(TIPO)

Este método es otra manera de extraer datos del lienzo que retorna el contenido en una cadena de texto codificada en BASE64. Esta cadena puede ser usada luego como fuente para otro lienzo, como fuente de un elemento HTML (por ejemplo, IMG), o incluso ser enviado al servidor o grabado en un archivo.

El método toDataURL() devuelve un String en formato data:url el cual contiene una representación de la imagen en el formato especificado por el parámetro **tipo** (el valor por defecto es PNG) y con una resolución de 96 dpi.

- Si la altura o el ancho del canvas es 0, se devolverá un string con "data:0".
- Si el parámetro **tipo** tiene un MIME diferente al de retorno se producirá un error de formato no soportado.
- Chrome también acepta el formato image/webp.

9.13.1. Crossbrowsing de toDataURL()

Chrome	IE	Firefox	Safari	Opera
4.0	9.0	3.7	4.0	9.0

9.13.2. Ejemplos

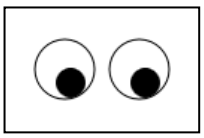
```
var canvas = document.getElementById("lienzo");
var dataURL = canvas.toDataURL();
console.log(dataURL);
// "
// blAAAADElEQVQImWNgoBMAAABpAAFEI8ARAAAAAElFTkSuQmCC"

var fullQuality = canvas.toDataURL("image/jpeg", 1.0);
// ...9oADAMBAAIRAxEAPwD/AD/6AP/Z"
var mediumQuality = canvas.toDataURL("image/jpeg", 0.5);
var lowQuality = canvas.toDataURL("image/jpeg", 0.1);
```

9.14. ANIMACIONES EN EL LIENZO

En este contexto, las animaciones son creadas a través de código Javascript convencional. No existen métodos para ayudarnos a animar figuras en el lienzo, ni tampoco existen procedimientos predeterminados para hacerlo. Básicamente, debemos borrar el área del lienzo que queremos animar, dibujar las figuras y repetir el proceso una y otra vez. Una vez que las figuras son dibujadas no se pueden mover. Sólo borrando el área y dibujando las figuras nuevamente podremos construir una animación. Por esta razón, en juegos o aplicaciones que requieren gran cantidad de objetos animados, es mejor usar imágenes en lugar de figuras construidas con trazados complejos.

Existen múltiples técnicas para lograr animaciones en el mundo de la programación. Algunas son simples y otras tan complejas como las aplicaciones para las que fueron creadas. Vamos a ver un ejemplo sencillo que usará el método **clearRect()** para limpiar el lienzo y dibujarlo nuevamente, generando una animación con sólo una función.



```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');
    window.addEventListener('mousemove', animacion, false);
}
function animacion(e){
    lienzo.clearRect(0,0,300,500);
    var xraton=e.clientX;
    var yraton=e.clientY;
    var xcentro=220;

    var ycentro=150;
    var angulo=Math.atan2(xraton-xcentro,yraton-ycentro);
    var x=xcentro+Math.round(Math.sin(angulo)*10);
    var y=ycentro+Math.round(Math.cos(angulo)*10);
    lienzo.beginPath();
    lienzo.arc(xcentro,ycentro,20,0,Math.PI*2, false);
    lienzo.moveTo(xcentro+70,150);
    lienzo.arc(xcentro+50,150,20,0,Math.PI*2, false);
    lienzo.stroke();
    lienzo.beginPath();
    lienzo.moveTo(x+10,y);
    lienzo.arc(x,y,10,0,Math.PI*2, false);
    lienzo.moveTo(x+60,y);
    lienzo.arc(x+50,y,10,0,Math.PI*2, false);
    lienzo.fill();
}
window.addEventListener("load", iniciar, false);
```

El ejemplo anterior mostrará dos ojos en pantalla que miran al puntero del ratón todo el tiempo. Para mover los ojos, se debe actualizar su posición cada vez que el ratón se desplaza. Por este motivo agregamos un listener para el evento mousemove en la función iniciar(). Cada vez que el puntero del ratón cambia de posición, el evento se dispara y la función animacion() se llama.

La función animacion() comienza limpiando el lienzo con la instrucción **clearRect(0,0,300,500)**. Luego, la posición del puntero del ratón se captura (usando las viejas propiedades clientX y clientY) y la posición del primer ojo se graba en las variables **xcentro** e **ycentro**. Después de que estas variables sean inicializadas, se comienza con las operaciones secundarias. Usando los valores de la posición del ratón y

el centro del ojo izquierdo, calculamos el ángulo de la línea invisible que va desde un punto al otro utilizando el método predefinido **atan2**. Este ángulo es usado en el siguiente paso para calcular el punto exacto del centro del iris del ojo izquierdo con la fórmula **xcentro + Math.round(Math.sin(angulo) × 10)** donde el número 10 representa la distancia desde el centro del ojo al centro del iris (porque el iris no está en el centro del ojo, está siempre sobre el borde).

Con todos estos valores podemos finalmente comenzar a dibujar nuestros ojos en el lienzo. El primer trazado es para los dos círculos representando los ojos. El primer método **arc()** para el primer ojo es posicionado en los valores **xcentro** y **ycentro**, y el círculo para el segundo ojo es generado 50 pixeles hacia la derecha usando la instrucción arc(xcentro+50, 150, 20, 0, Math.PI*2, false). La parte animada del gráfico se crea a continuación con el segundo trazado. Este trazado usa las variables x e y con la posición calculada previamente a partir del ángulo. Ambos iris se dibujan como un círculo negro sólido utilizando el método fill().

Este proceso se repetirá una y otra vez cada vez que el evento mousemove sea disparado.

9.15. PROCESANDO VIDEO

Al igual que para animaciones, no hay ningún método especial para mostrar video en el elemento CANVAS. La única manera de hacerlo es tomando cada cuadro del video desde el tag VIDEO y dibujarlo como una imagen en el lienzo usando drawImage(). Así que básicamente, el procesamiento de video en el lienzo es hecho con la combinación de técnicas ya comentadas anteriormente.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Video en el Lienzo</title>
        <style>
            .cajas{ display: inline-block; margin: 10px; padding: 5px; border: 1px solid #999999; }
        </style>
        <script>
            function iniciar(){
                var elemento=document.getElementById('lienzo');
                lienzo=elemento.getContext('2d');
                video=document.getElementById('medio');
                video.addEventListener('click', presionar, false);
            }
```

```
            function presionar(){
                if(!video.paused && !video.ended){
                    video.pause();
                    window.clearInterval(bucle);
                } else {
                    video.play();
                    bucle=setInterval(procesarCuadros, 33);
                }
            }
            function procesarCuadros(){
                lienzo.drawImage(video,0,0);
                var info=lienzo.getImageData(0,0,483,272);
                var pos, gris;
                for(x=0;x<=483;x++){
                    for(y=0;y<=272;y++){
                        pos=(info.width*4*y)+(x*4);
                        gris=parseInt(info.data[pos]*0.2989 +
                        info.data[pos+1]*0.5870 + info.data[pos+2]*0.1140);
                        info.data[pos]=gris;
                        info.data[pos+1]=gris;
                        info.data[pos+2]=gris;
                    }
                }
                lienzo.putImageData(info,0,0);
            }
            window.addEventListener("load", iniciar, false);
        </script>
    </head>
    <body>
        <section class="cajas">
            <video id="medio" width="360" height="240">
                <source src="http://techslides.com/demos/sample-videos/small.webm" type="video/webm" />
                <source src="http://techslides.com/demos/sample-videos/small.ogv" type="video/ogg" />
                <source src="http://techslides.com/demos/sample-videos/small.mp4" type="video/mp4" />
                <source src="http://techslides.com/demos/sample-videos/small.3gp" type="video/3gp" />
            </video>
        </section>
        <section class="cajas">
            <canvas id="lienzo" width="483" height="272">Su navegador no soporta el tag CANVAS</canvas>
        </section>
    </body>
</html>
```

Este ejemplo usa los métodos **getImageData()** y **putImageData()** para procesar datos de imagen. Como se ha comentado anteriormente, estos métodos extraen información del lienzo. Debido a restricciones de seguridad, no se puede extraer información del elemento CANVAS después de que una imagen tomada desde una fuente externa sea dibujada en el lienzo. Sólo cuando el documento y la imagen pertenezcan al mismo origen de datos (dominio) o las directrices CORS lo permitan estos métodos trabajarán adecuadamente.

Como se comentó previamente, para procesar video en el lienzo, simplemente debemos recurrir a códigos y técnicas ya vistas. En este código se ha utilizado la función presionar() para comenzar y detener la reproducción del video haciendo clic sobre el mismo. También utilizamos una función llamada procesarCuadros() vista anteriormente, excepto que esta vez en lugar de invertir la imagen estamos utilizando una fórmula que transforma todos los colores de cada cuadro del video en el correspondiente gris para reproducir el video en blanco y negro.

La función presionar() cumple con dos propósitos: comenzar o detener la reproducción del video e iniciar un intervalo que ejecutará la función procesarCuadros() cada 33 milisegundos. Esta función toma un cuadro del elemento VIDEO y lo dibuja en el lienzo con la instrucción **drawImage(video,0,0).** Luego los datos son extraídos del lienzo con el método **getImageData()** y cada pixel de ese cuadro se procesa por medio de dos bucles for (como se hizo en otro ejemplo anterior).

El proceso utilizado para convertir cada uno de los colores que integran cada pixel en su correspondiente gris es uno de los más populares y fáciles de encontrar en Internet.

La fórmula es la siguiente: **rojo × 0.2989 + verde × 0.5870 + azul × 0.1140**.

Cuando la fórmula se calcula, el resultado debe ser asignado a cada color del pixel (rojo, verde y azul), como se hacía en el ejemplo usando la variable gris. El proceso terminará cuando se dibuje nuevamente el cuadro modificado en el lienzo utilizando el método **putImageData()**.

IMPORTANTE: Procesar video en tiempo real no es una buena práctica. Este ejemplo tiene un carácter didáctico.

9.16. EVITAR PERDIDAS EN EL PROCESADO DE IMÁGENES

Cada vez que se procesa una imagen pueden producirse pérdidas de calidad en el lienzo. Para evitar estas pérdidas debemos utilizar una propiedad del objeto CANVAS que es **imageSmoothingEnabled**.

9.16.1. La propiedad imageSmoothingEnabled

Esta propiedad indica si los patrones de relleno y el método drawImage() deben suavizar las imágenes al cambiar la escala o procesar la imagen. El contexto 2D por defecto establece esta propiedad a **true**, que suaviza los píxeles dentro del CANVAS. Por ello, en el desarrollo web empresarial, lo normal es establecerlo siempre a **false**.

El problema de esta propiedad se presenta en el Cross Browsing ya que cada navegador tiene su propia interpretación de la propiedad. Por ello debemos añadir todas sus posibles variaciones.

```
document.getElementById("lienzo").lienzo.getContext("2d").imageSmoothingEnabled       = false;
document.getElementById("lienzo").lienzo.getContext("2d").webkitImageSmoothingEnabled = false;
document.getElementById("lienzo").lienzo.getContext("2d").mozImageSmoothingEnabled    = false;
document.getElementById("lienzo").lienzo.getContext("2d").msImageSmoothingEnabled     = false;
document.getElementById("lienzo").lienzo.getContext("2d").oImageSmoothingEnabled      = false;
```

10. La API Drag and Drop

Arrastrar un elemento desde un lugar y luego soltarlo en otro es algo que hacemos todo el tiempo en aplicaciones de escritorio, pero ni siquiera imaginamos hacerlo en la web. Esto no es debido a que las aplicaciones web son diferentes sino porque desarrolladores nunca contaron con una tecnología estándar disponible para ofrecer esta herramienta. Ahora, gracias a la API Drag and Drop, introducida por la especificación HTML5, finalmente tenemos la oportunidad de crear software para la web que se comportará exactamente como las aplicaciones de escritorio que usamos desde siempre.

10.1. NUEVOS EVENTOS

Uno de los más importantes aspectos de esta API es un conjunto de siete nuevos eventos introducidos para informar sobre cada una de las situaciones involucradas en el proceso. Algunos de estos eventos son disparados por la fuente (el elemento que es arrastrado) y otros son disparados por el destino (el elemento en el cual el elemento arrastrado será soltado).

Eventos asociados al origen (elemento arrastrado)

10.1.1. Evento dragstart

Este evento es disparado en el momento en el que el arrastre comienza. Los datos asociados con el elemento origen son definidos en este momento en el sistema.

10.1.2. Evento drag

Este evento es similar al evento mousemove, excepto que será disparado durante una operación de arrastre por el elemento origen.

10.1.3. Evento dragend

Cuando la operación de arrastrar y soltar finaliza (sea la operación exitosa o no) este evento es disparado por el elemento origen.

Eventos asociados al destino (dónde es soltado)

10.1.4. Evento dragenter

Cuando el puntero del ratón entra dentro del área ocupada por los posibles elementos destino durante una operación de arrastrar y soltar, este evento es disparado.

10.1.5. Evento dragover

Este evento es similar al evento mousemove, excepto que es disparado durante una operación de arrastre por posibles elementos destino.

10.1.6. Evento drop

Cuando el elemento origen es soltado durante una operación de arrastrar y soltar, este evento es disparado por el elemento destino.

10.1.7. Evento dragleave

Este evento es disparado cuando el ratón sale del área ocupada por un elemento durante una operación de arrastrar y soltar. Este evento es generalmente usado junto con dragenter para mostrar una ayuda visual al usuario que le permita identificar el elemento destino (donde soltar).

10.2. CROSSBROWSING DE LA API DRAG & DROP

La compatibilidad de esta API viene expresada en la siguiente tabla

API	Chrome	IE	Firefox	Safari	Opera
Drag and Drop	4.0	9.0	3.5	6.0	12.0

10.3. PREVENCIÓN DE COMPORTAMIENTOS POR DEFECTO

Para obtener el resultado que queremos, necesitamos prevenir en algunas ocasiones este comportamiento por defecto y personalizar las reacciones del navegador. Para algunos eventos, como **dragenter**, **dragover** y **drop**, la prevención es necesaria, incluso cuando una acción personalizada ya fue especificada.

Veamos cómo debemos proceder usando un ejemplo simple.

```
<!DOCTYPE HTML>
<html>
    <head>
        <style>
            #div1 {width:350px;height:70px;padding:10px;border:1px solid #aaaaaa;}
        </style>
        <script>
            function allowDrop(ev) { ev.preventDefault(); }
            function drag(ev)      { ev.dataTransfer.setData("text", ev.target.id); }
            function drop(ev)      {
                ev.preventDefault();
                var data = ev.dataTransfer.getData("text");
                ev.target.appendChild(document.getElementById(data));
            }
        </script>
    </head>
    <body>
        <p>Drag the W3Schools image into the rectangle:</p>
        <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
        <br>
        <img id="drag1" src="img_logo.gif" draggable="true" ondragstart="drag(event)" width="336"
height="69">
    </body>
</html>
```

10.4. DATATRANSFER

El objeto DataTransfer contiene la información en una operación arrastrar y soltar. El objeto dataTransfer tiene varios métodos y propiedades asociados. Los métodos de este objeto son setData(), getData() y clearData() y están a cargo de la información que es transferida.

10.4.1. Método setData(tipo, dato)

Este método es usado para declarar los datos a ser enviados y su tipo. El método puede recibir tipos de datos regulares (como text/plain, text/html o text/uri-list), tipos de datos especiales (como URL o Text) o incluso tipos de datos personalizados. Un método setData() debe ser llamado por cada tipo de datos que queremos enviar en la misma operación.

10.4.2. Método getData(tipo)

Este método retorna los datos enviados por el origen, pero solo del tipo especificado.

10.4.3. Método clearData()

Este método remueve los datos del tipo especificado.

10.4.4. Método setDragImage(elemento, x, y)

Algunos navegadores muestran una imagen en miniatura junto al puntero del ratón que representa al elemento que está siendo arrastrado. Este método es usado para personalizar esa imagen y seleccionar la posición la posición en la que será mostrada relativa al puntero del ratón. Esta posición es determinada por los atributos x e y.

10.4.5. Propiedad types

Esta propiedad retorna un array conteniendo los tipos de datos que fueron declarados durante el evento dragstart (por el código o el navegador). Podemos grabar este array en una variable (lista=dataTransfer.types) y luego leerlo con un bucle for.

10.4.6. Propiedad files

Esta propiedad retorna un array conteniendo información acerca de los archivos que están siendo arrastrados.

10.4.7. Propiedad dropEffect

Esta propiedad retorna el tipo de operación actualmente seleccionada. Los posibles valores son none, copy, link y move.

10.4.8. Propiedad effectAllowed

Esta propiedad retorna los tipos de operaciones que están permitidas. Puede ser usada para cambiar las operaciones permitidas. Los posibles valores son: none, copy, copyLink, copyMove, link, linkMove, move, all y uninitialized.

10.5. PERSONALIZAR COMPORTAMIENTOS

De momento no hemos hecho nada con el evento **dragenter**. Sólo hemos cancelado el comportamiento por defecto de los navegadores cuando este evento es disparado para prevenir efectos no deseados. Tampoco hemos utilizado los eventos **dragleave** y **dragend**. Estos son eventos importantes que nos permitirán ayudar al usuario cuando se encuentra arrastrando objetos por la pantalla.

```
function iniciar(){
    source = document.getElementById('imagen');
    source.addEventListener('dragstart', arrastrado, false);
    source.addEventListener('dragend', finalizado, false);

    target = document.getElementById('cajasoltar');
    target.addEventListener('dragenter', entrando, false);
    target.addEventListener('dragleave', saliendo, false);
    target.addEventListener('dragover', function(e){ e.preventDefault(); }, false);
    target.addEventListener('drop', soltado, false);
}

function entrando(e){ e.preventDefault(); soltar.style.background='rgba(0,150,0,.2)'; }
function saliendo(e){ e.preventDefault(); soltar.style.background='#FFFFFF'; }
function finalizado(e){ elemento=e.target; elemento.style.visibility='hidden'; }

function arrastrado(e){
    var codigo='<img src="'+source.getAttribute('src')+'">';
    e.dataTransfer.setData('Text', codigo);
}
function soltado(e){
    e.preventDefault();
    target.style.background='#FFFFFF';
    target.innerHTML=e.dataTransfer.getData('Text');
}
window.addEventListener('load', iniciar, false);
```

En el código anterior se puede ver como se asignan comportamientos a los eventos de la API de Drag and Drop para ayudar al usuario.

10.6. SETDRAGIMAGE()

Cambiar la imagen en miniatura que es mostrada junto al puntero del ratón en una operación arrastrar y soltar puede parecer inútil, pero en ocasiones nos evitará dolores de cabeza. El método **setDragImage()** no sólo nos permite cambiar la imagen sino también recibe dos atributos, x e y, para especificar la posición de esta imagen relativa al puntero. Algunos navegadores generan una imagen en miniatura por defecto a partir del objeto original que es arrastrado, pero su posición relativa al puntero del ratón es determinada por la posición del puntero cuando el proceso comienza. El método **setDragImage()** nos permite declarar una posición específica que será la misma para cada operación arrastrar y soltar.

```
function arrastrado(e){
    elemento=e.target;
    e.dataTransfer.setData('Text', elemento.getAttribute('id'));
    e.dataTransfer.setDragImage(e.target, 0, 0);
}
```

Probablemente, con este ejemplo, nos estemos acercando a lo que sería una aplicación de la vida real. Cuando la imagen es arrastrada, la función arrastrado() es llamada y en su interior una imagen miniatura es generada con el método setDragImage(). El código también crea el contexto para trabajar con el lienzo y dibuja la imagen soltada usando el método drawImage() estudiado en el capítulo anterior. Al final de todo el proceso la imagen original es ocultada usando la función finalizado().

Para la imagen miniatura personalizada usamos el mismo elemento que está siendo arrastrado, pero declaramos la posición relativa al puntero del ratón como 0,0. Gracias a esto ahora sabremos siempre cual es exactamente la ubicación de la imagen miniatura. Aprovechamos este dato importante dentro de la función soltado(). Usando la misma técnica introducida en el capítulo anterior, calculamos dónde el objeto es soltado dentro del lienzo y dibujamos la imagen en ese lugar preciso. Si prueba este ejemplo en navegadores que ya aceptan el método setDragImage() (por ejemplo, Firefox 4+), verá que la imagen es dibujada en el lienzo exactamente en la posición de la imagen miniatura que acompaña al puntero del ratón, haciendo fácil para el usuario seleccionar el lugar adecuado donde soltarla.

10.7. ARCHIVOS

Posiblemente la característica más interesante de la API Drag and Drop es la habilidad de trabajar con archivos. La API no está sólo disponible dentro del documento, sino también integrada con el sistema, permitiendo a los usuarios arrastrar elementos desde el navegador hacia otras aplicaciones y viceversa. Y normalmente los elementos más requeridos desde aplicaciones externas son archivos.

Como vimos anteriormente, existe una propiedad especial en el objeto dataTransfer que retorna un array que contiene la lista de archivos que están siendo arrastrados. Podemos usar esta información para construir complejos códigos que trabajan con archivos o subirlos a un servidor.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Drag and Drop</title>
            <link rel="stylesheet" href="dragdrop.css">
            <script>
                function iniciar(){
                    target=document.getElementById('cajatarget');
                    target.addEventListener('dragenter', function(e){
                    e.preventDefault(); }, false);
                    target.addEventListener('dragover', function(e){
                    e.preventDefault(); }, false);
                    target.addEventListener('drop', soltado, false);
                }
                function soltado(e){
                    e.preventDefault();
                    var archivos=e.dataTransfer.files;
                    var lista='';
                    for(var f=0;f<archivos.length;f++){
                        lista+='Archivo: '+archivos[f].name+' '+archivos[f].size+'<br>';
                    }
                    target.innerHTML=lista;
                }
                window.addEventListener('load', iniciar, false);
            </script>
        </head>
        <body>
        <section id="cajatarget">
            Arrastre y suelte archivos en este espacio
        </section>
    </body>
</html>
```

La información retornada por la propiedad files del objeto dataTransfer puede ser grabada en una variable y luego leída por un bucle for. En el código anterior, sólo se muestran el nombre y el tamaño del archivo en el elemento destino usando las propiedades name y size. Para aprovechar esta información y construir aplicaciones más complejas, necesitaremos recurrir a otras APIs y técnicas de programación comentadas en otros documentos de Internet.

11. La API Geolocation

11.1. ENCONTRANDO SU LUGAR

La API Geolocation fue diseñada para que los navegadores puedan proveer un mecanismo de detección por defecto que permita a los desarrolladores determinar la ubicación física real del usuario. Previamente solo contábamos con la opción de construir una gran base de datos con información sobre direcciones IP y programar códigos exigentes dentro del servidor que nos darían una idea aproximada de la ubicación del usuario (generalmente tan imprecisa como su país).

Esta API aprovecha nuevos sistemas, como triangulación de red y GPS, para retornar una ubicación precisa del dispositivo que está accediendo a la aplicación. La valiosa información retornada nos permite construir aplicaciones que se adaptarán a las particulares necesidades del usuario o proveerán información localizada de forma automática. Tres métodos específicos son provistos para usar la API:

11.2. GETCURRENTPOSITION(UBICACION, ERROR, CONFIGURACION)

Este es el método utilizado para consultas individuales. Puede recibir hasta tres atributos: una función para procesar la ubicación retornada, una función para procesar los errores retornados, y un objeto para configurar cómo la información será adquirida. Solo el primer atributo es obligatorio para que el método trabaje correctamente.

Este atributo es una función que recibirá un objeto llamado **Position**, el cual contiene toda la información retornada por los sistemas de ubicación.

El objeto **Position** tiene dos atributos:

Atributo	Descripción
coords	Este atributo contiene un grupo de valores que establecen la ubicación del dispositivo y otros datos importantes. Los valores son accesibles a través de siete atributos internos: latitude (latitud), longitude (longitud), altitude (altitud en metros), accuracy (exactitud en metros), altitudeAccuracy (exactitud de la altitud en metros), heading (dirección en grados) y speed (velocidad en metros por segundo).
timestamp	Este atributo indica el momento en el que la información fue adquirida (en formato timestamp). Este objeto, como dijimos, es pasado a la función que definimos como atributo del método getCurrentPosition() y luego sus datos son accedidos y procesados en esta función. Veamos un ejemplo práctico de cómo usar este método:

```
<!DOCTYPE html>
    <html lang="es">
    <head>
        <title>Geolocation</title>
        <script type="text/javascript">
            function iniciar(){
                var boton=document.getElementById('obtener');
                boton.addEventListener('click', obtener, false);
            }
            function obtener(){
                navigator.geolocation.getCurrentPosition(mostrar);
            }
            function mostrar(posicion){
                var ubicacion=document.getElementById('ubicacion');
                var datos='';
                datos+='Latitud: '+posicion.coords.latitude+'<br>';
                datos+='Longitud: '+posicion.coords.longitude+'<br>';
                datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
                ubicacion.innerHTML=datos;
            }
            window.addEventListener('load', iniciar, false);
        </script>
    </head>
    <body>
        <section id="ubicacion">
            <button id="obtener">Obtener mi Ubicación</button>
        </section>
    </body>
</html>
```

Una implementación de la API Geolocation es sencilla: declaramos el método getCurrentPosition() y creamos una función que mostrará los valores retornados por el mismo. El método getCurrentPosition() es un método del objeto geolocation. Este es un nuevo objeto que es parte del objeto navigator, un objeto Javascript que fue anteriormente implementado para retornar información acerca del navegador y el sistema. Por lo tanto, para acceder al método getCurrentPosition() la sintaxis a usar es navigator.geolocation.getCurrentPosition(**función**), dónde **función** es una función personalizada que recibirá el objeto **Position** y procesará la información que contiene.

En el código anterior, llamamos a esta función mostrar(). Cuando el método getCurrentPosition() es llamado, un nuevo objeto Position es creado con la información de la ubicación actual del usuario y es enviado a la función mostrar(). Referenciamos este objeto dentro de la función con la variable posicion, y luego usamos esta variable para mostrar los datos.

El objeto **Position** tiene dos importantes atributos: **coords** y **timestamp**. En el ejemplo sólo usamos **coords** para acceder a la información que queremos mostrar (latitud, longitud y exactitud). Estos valores son grabados en la variable **datos** y luego mostrados en la pantalla como el nuevo contenido del elemento **ubicacion**.

Agregando un segundo atributo al método getCurrentPosition(), con otra función, podremos capturar los errores producidos en el proceso. Uno de esos errores ocurre cuando el usuario no acepta compartir sus datos.

Junto con el objeto Position, el método getCurrentPosition() retorna el objeto **PositionError** si un error es detectado. Este objeto es enviado al segundo atributo de getCurrentPosition(), y tiene dos atributos

internos, **error** y **message**, para proveer el valor y la descripción del error. Los tres posibles errores son representados por las siguientes constantes:

Error	Descripción
PERMISSION_DENIED	Este error ocurre cuando el usuario no acepta activar el sistema de ubicación para compartir su información.
POSITION_UNAVAILABLE	Este error ocurre cuando la ubicación del dispositivo no pudo determinarse por ninguno de los sistemas de ubicación disponibles.
TIMEOUT	Este error ocurre cuando la ubicación no pudo ser determinada en el período de tiempo máximo declarado en la configuración.

```
function errores(error){
    alert('Error: '+error.code+' '+error.message);
}
window.addEventListener('load', iniciar, false);
```

Los mensajes de error son ofrecidos para uso interno. El propósito es ofrecer un mecanismo para que la aplicación reconozca la situación y proceda de acuerdo al error recibido. En el código del Listado 9-3, agregamos un segundo atributo al método getCurrentPosition() (otra función) y creamos la función errores() para mostrar la información de los atributos code y message. El valor de code será un número entre 0 y 3 de acuerdo al número de error (listado anteriormente).

El objeto PositionError es enviado a la función errores() y representado en esta función por la variable error. Para propósitos didácticos, usamos un método alert() que muestra los datos recibidos, pero usted debería procesar esta información en silencio, si es posible, sin alertar al usuario de nada. Podemos también controlar por errores de forma individual (error.PERMISSION_DENIED, por ejemplo) y actuar solo si ese error en particular ocurrió.

El tercer atributo que podemos usar en el método getCurrentPosition() es un objeto que contiene tres posibles propiedades:

Propiedad	Descripción
enableHighAccuracy	Esta es una propiedad booleana para informar al sistema que requerimos de la información más exacta que nos pueda ofrecer. El navegador intentará obtener esta información a través de sistemas como GPS, por ejemplo, para retornar la ubicación exacta del dispositivo. Estos son sistemas que consumen muchos recursos, por lo que su uso debería estar limitado a circunstancias muy específicas. Para evitar consumos innecesarios, el valor por defecto de esta propiedad es false (falso).
timeout	Esta propiedad indica el tiempo máximo de espera para que la operación finalice. Si la información de la ubicación no es obtenida antes del tiempo indicado, el error TIMEOUT es retornado. Su valor es en milisegundos.
maximumAge	Las ubicaciones encontradas previamente son almacenadas en un caché en el sistema. Si consideramos apropiado recurrir a la información grabada en lugar de intentar obtenerla desde el sistema (para evitar consumo de recursos o para una

	respuesta rápida), esta propiedad puede ser declarada con un tiempo límite específico. Si la última ubicación almacenada es más vieja que el valor de este atributo entonces una nueva ubicación es solicitada al sistema. Su valor es en milisegundos.

```
function obtener(){
    var geoconfig={ enableHighAccuracy: true, timeout: 10000, maximumAge: 60000 };
    navigator.geolocation.getCurrentPosition(mostrar, errores, geoconfig);
}
```

El código anterior intentará obtener la ubicación más exacta posible del dispositivo en no más de 10 segundos, pero sólo si no hay una ubicación previa en el caché capturada menos de 60 segundos atrás (si existe una ubicación previa con menos de 60 segundos de antigüedad, éste será el objeto Position retornado).

El objeto conteniendo los valores de configuración fue creado primero y luego referenciado desde el método getCurrentPosition(). Nada cambió en el resto del código. La función mostrar() mostrará la información en la pantalla independientemente de su origen (si proviene del caché o es nueva).

11.3. WATCHPOSITION(UBICACION, ERROR, CONFIGURACION)

Este método es similar al anterior, excepto que comenzará un proceso de vigilancia para la detección de nuevas ubicaciones. Trabaja de forma similar que el conocido método **setInterval()** de Javascript, repitiendo el proceso automáticamente en determinados períodos de tiempo de acuerdo a la configuración por defecto o a los valores de sus atributos. Recibe tres atributos y realiza la misma tarea que getCurrentPosition salvo por la diferencia de que es el método getCurrentPosition() realiza una única operación y **watchPosition()** ofrece nuevos datos cada vez que la ubicación cambia. Este método vigilará todo el tiempo la ubicación y enviará información a la función correspondiente cuando se detecte una nueva ubicación, a menos que cancelemos el proceso con el método clearWatch().

Este es un ejemplo de cómo implementar el método watchPosition() basado en códigos previos:

```
function iniciar(){
    var boton=document.getElementById('obtener');
    boton.addEventListener('click', obtener, false);
}
function obtener(){
    var geoconfig={ enableHighAccuracy: true, maximumAge: 60000 };
    control=navigator.geolocation.watchPosition(mostrar, errores, geoconfig);
}
function mostrar(posicion){
    var ubicacion=document.getElementById('ubicacion');
    var datos='';
    datos+='Latitud: '+posicion.coords.latitude+'<br>';
    datos+='Longitud: '+posicion.coords.longitude+'<br>';
    datos+='Exactitud: '+posicion.coords.accuracy+'mts.<br>';
    ubicacion.innerHTML=datos;
}
function errores(error){
    alert('Error: '+error.code+' '+error.message);
}
```

```
window.addEventListener('load', iniciar, false);
```

Si este código se ejecuta desde un DesktopPC no se notará ningún cambio pero, en un dispositivo móvil, se mostrará información nueva (actualizada) cada vez que haya una modificación en la ubicación del dispositivo. El atributo maximumAge determina el tiempo máximo entre "mediciones". Si la nueva ubicación cambió en los últimos 60 segundos con respecto a la anterior, entonces se muestra, de lo contrario se ignorará.

El valor retornado por el método watchPosition() es almacenado en la variable control, que es como un identificador de la operación. Si más adelante se desea cancelar el proceso de vigilancia, sólo se debe ejecutar la línea **clearWatch(control)** y watchPosition() dejará de actualizar la información.

11.4. CLEARWATCH(ID)

El método **watchPosition()** retorna un valor que puede ser almacenado en una variable para luego ser usado como referencia pro el método **clearWatch()** y así detener la vigilancia. Este método es similar a clearInterval() usado para detener los procesos comenzados por **setInterval()**.

11.5. USOS PRÁCTICOS CON GOOGLE MAPS

Hasta el momento hemos mostrado la información sobre la ubicación exactamente como la recibimos. Sin embargo, estos valores normalmente no significan nada para la gente común. La mayoría de nosotros no podemos inmediatamente decir cuál es nuestra actual ubicación en valores de latitud y longitud, y mucho menos identificar a partir de estos valores una ubicación en el mundo. Para esto último, disponemos de dos alternativas: usar esta información internamente para calcular posiciones, distancias y otros valores que nos permitirán ofrecer resultados específicos a nuestros usuarios (como productos o servicios en el área), o podemos ofrecer la información obtenida por medio de la API Geolocation en un medio mucho más comprensible. ¿Y qué más comprensible que un mapa para representar una ubicación geográfica?

La API de Google Maps es una API Javascript externa, provista por Google y que nada tiene que ver con HTML5 pero se hace referencia a ella ya que es ampliamente utilizada en los sitios webs. Ofrece una variedad de alternativas para trabajar con mapas interactivos e incluso vistas reales de lugares muy específicos a través de la tecnología StreetView.

A continuación se muestra un ejemplo sencillo de utilización aprovechando una parte de la API llamada Static Maps API.

```
function mostrar(posicion){
  var ubicacion=document.getElementById('ubicacion');
  var mapurl='http://maps.google.com/maps/api/staticmap?center='+posicion.coords.latitude+
      ','+posicion.coords.longitude+'&zoom=12&size=400x400&sensor=false&markers='+posicion.coords.latitude+
      ','+posicion.coords.longitude;
  ubicacion.innerHTML='<img src="'+mapurl+'">';
}
```

El código es simple. Usamos el método getCurrentPosition() de ejemplos anteriores y enviamos la información a la función mostrar() para ser agregada a una URL de Google y luego la dirección obtenida es insertada como la fuente de un elemento <img> para mostrar el mapa en pantalla.

12. La API Web Storage

Esta característica de HTML5 no tiene nada que ver con el concepto de Cookie aunque, se puede confundir con cierta facilidad. Por ello, lo primero que debemos aclarar es cómo definimos los conceptos de LocalStorage, SessionStorage y en qué se diferencian a las Cookies.

12.1. TIPOS DE ALMACENAMIENTO WEB

Existen 3 tipos de almacenamiento Web hasta el momento que son localStorage, sessionStorage y Cookies. Como algunos ya sabrán, las Cookies se caracterizan porque tienen una limitación de espacio de 4KB, pueden ser activadas o desactivadas por los usuarios, tienen caducidad y aumentan el peso de las peticiones ya que, toda la información contenida en ellas, se envía al servidor para ser analizada y, seguidamente, traer de vuelta al navegador cliente.

12.2. LOCALSTORAGE

LocalStorage, como su propio nombre indica, es únicamente un almacenamiento local. Lo que sucede es que, este espacio de memoria reservado, es persistente entre las diferentes pestañas incluso si se cierra el navegador. Es por ello que hay que tener cuidado con la información que guarda en ella. El cómo podemos utilizar JavaScript para guardar información en los navegadores de los usuarios viene definido en las especificaciones de la W3C y en la WHATWG.

LocalStorage posee una capacidad de entre 2.5MB y 5.0MB dependiendo del navegador, no tiene caducidad alguna, es compatible en casi todos los navegadores y no aumenta el peso de cada petición ya que, toda la información contenida en ella, es únicamente "visible" desde del entorno local. Fuera del entorno local es como si no existiese.

12.2.1. CrossBrowsing de localStorage

Chrome	Firefox (Gecko)	IE	Opera	Safari (Webkit)
4+	3.5+	8+	10.50+	4

12.2.2. Uso de localStorage

LocalStorage tiene 3 métodos que son para establecer, recuperar y eliminar las variables almacenadas en este espacio de memoria:

```
/* Establecer un nuevo valor */
localStorage.setItem('variable', 'valor');

/* Recuperar el valor de una variable */
var var1 = localStorage.getItem('variable');

/* Eliminar variable de localStorage */
localStorage.removeItem('variable');
```

12.3. SESSIONSTORAGE

SessionStorage es exáctamente igual a localStorage en prácticamente todo con la salvedad de que no tiene un almacenamiento persistente, es decir, que en cuanto cerremos el navegador, automáticamente se eliminará.

12.3.1. CrossBrowsing de SessionStorage

Chrome	Firefox (Gecko)	IE	Opera	Safari (Webkit)
5+	2+	8+	10.50+	4

12.3.2. Uso de SessionStorage

SessionStorage tiene 3 métodos que son para establecer, recuperar y eliminar las variables almacenadas en este espacio de memoria:

```
/* Establecer un nuevo valor */
sessionStorage.setItem('variable', 'valor');

/* Recuperar el valor de una variable */
var var1 = sessionStorage.getItem('variable');

/* Eliminar variable de localStorage */
sessionStorage.removeItem('variable');
```

13. La API IndexedDB

IndexedDB es un sistema de base de datos destinado a almacenar información indexada en el ordenador del usuario. Fue desarrollada como una API de bajo nivel con la intención de permitir un amplio espectro de usos. Esto la convierte en una de las API más poderosas, pero también una de las más complejas. El objetivo fue proveer la estructura más básica posible para permitir a los desarrolladores construir librerías y crear interfaces de alto nivel para satisfacer necesidades específicas. En una API de bajo nivel como esta tenemos que hacernos cargo de cada aspecto y controlar las condiciones de cada proceso en toda operación realizada. El resultado es una API a la que la mayoría de los desarrolladores tardará en acostumbrarse y probablemente utilizará de forma indirecta a través de otras librerías populares construidas sobre ella que seguramente surgirán en un futuro cercano.

La estructura propuesta por IndexedDB es también diferente de SQL u otros sistemas de base de datos populares. La información es almacenada en la base de datos como objetos (registros) dentro de lo que es llamado Almacenes de Objetos (tablas). Los Almacenes de Objetos no tienen una estructura específica, solo un nombre e índices para poder encontrar los objetos en su interior. Estos objetos tampoco tienen una estructura definida, pueden ser diferentes unos de otros y tan complejos como necesitemos. La única condición para los objetos es que contengan al menos una propiedad declarada como índice para que sea posible encontrarlos en el Almacén de Objetos.

13.1. ¿POR QUÉ UTILIZAR INDEXEDDB?

La W3C ha anunciado que la base de datos SQL Web es una especificación almacenamiento local en desuso por lo desarrollador web no debe utilizar esta tecnología más. IndexedDB es una alternativa para la base de datos de SQL web y más eficaz que las tecnologías más antiguas.

13.2. CARACTERÍSTICAS

- Almacena los valores de par de claves.
- No es una base de datos relacional.
- Principalmente es asíncrona.
- No es un lenguaje de consulta estructurado.
- Se ha diseñado para acceder a los datos del mismo dominio.

13.3. CROSSBROWSING DE INDEXEDDB

Chrome	Firefox	IE	Edge	Opera	Safari
37+	38+	11+	13+	37+	9.1+

13.4. DETECTAR LA DISPONIBILIDAD DE INDEXEDDB

La detección de la disponibilidad de IndexedDB se realiza a través de unos diferentes prefijos según sea un navegador u otro.

El siguiente código permite detectar la disponibilidad de IndexedDB y guarda en la variable indexedDB del objeto window su resultado.

```
window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitIndexedDB || window.msIndexedDB;

window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction || window.msIDBTransaction;
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.msIDBKeyRange

if (!window.indexedDB){ window.alert("Your browser doesn't support a stable version of IndexedDB."); }
```

13.5. ABRIR UNA BASE DE DATOS INDEXADA

Antes de crear una base de datos, tenemos que preparar algunos datos para el inicio de la base de datos con detalles de los empleados de la empresa.

```
// Datos de nuestros empleados
const employeeData = [
   { id: "01", name: "Fulano", age: 35, email: "fulano@emails.com" },
   { id: "02", name: "Mengano", age: 24, email: "mengano@emails.com" }
];

var db;
// Estabecemos el nombre de la base de datos y versión (sólo números enteros)
var request = window.indexedDB.open("newDatabase", 1);

request.onerror = function(event) { console.log("error: "); };
request.onsuccess = function(event) { db = request.result; console.log("success: "+ db); };
request.onupgradeneeded = function(event) {
    var db = event.target.result;
    // Creamos la tabla empleados
    var objectStore = db.createObjectStore("employee", {keyPath: "id"});

    // Añadimos a la tabla empleados todos los registros definidos anteriormente
    for (var i in employeeData) {
        objectStore.add(employeeData[i]);
    }
}
```

13.6. LA ADICIÓN DE LOS DATOS

Hay tres modos de transacciones que son:

- Escritura y lectura (**readwrite**)
- Sólo lectura (**readonly**)
- Cambio de versión (**versionchange**).

La única manera de crear y eliminar almacenes de objetos (tablas) e índices es el uso de una transacción de cambio de versión.

```
function add() {
   var request = db.transaction(["employee"], "readwrite")
   .objectStore("employee")
   .add({ id: "01", name: "prasad", age: 24, email: "prasad@tutorialspoint.com" });

   request.onsuccess = function(event) {
      alert("Prasad has been added to your database.");
   };

   request.onerror = function(event) {
      alert("Unable to add data\r\nPrasad is already exist in your database! ");
   }
}
```

13.7. RECUPERANDO DATOS

Para la recuperación de datos es necesario proporcionar una clave para recuperar el valor.

Puesto que sólo hay un almacén de objetos, se podría evitar pasar una lista de almacenes de objetos que se utiliza en cada transacción y simplemente enviar el nombre como una cadena.

Llamando transaction() sin modo especificado se le asigna una transacción "readonly".

```
function read() {
   var transaction = db.transaction(["employee"]);
   var objectStore = transaction.objectStore("employee");
   var request = objectStore.get("00-03");

   request.onerror = function(event) {
      alert("Unable to retrieve data from database!");
   };

   request.onsuccess = function(event) {
      if(request.result) {
```

```
            alert("Name: "+request.result.name+", Age: "+request.result.age+", Email: "+request.result.email);
        } else {
            alert("Kenny couldn't be found in your database!");
        }
    };
}
```

Usando con get(), podemos almacenar los datos en el objeto en lugar de almacenar los datos en el cursor y podemos recuperar los datos de cursor.

```
function readAll() {
    var objectStore = db.transaction("employee").objectStore("employee");

    objectStore.openCursor().onsuccess = function(event) {
        var cursor = event.target.result;

        if (cursor) {
            alert("Name for id " + cursor.key + " is " + cursor.value.name + ", Age: " + cursor.value.age + ",
Email: " + cursor.value.email);
            cursor.continue();
        }

        else {
            alert("No more entries!");
        }
    };
}
```

13.8. ELIMINADO DE DATOS

Podemos eliminar los datos de IndexedDB con la función remove().

```
function remove() {
    var request = db.transaction(["employee"], "readwrite")
    .objectStore("employee")
    .delete("02");

    request.onsuccess = function(event) {
        alert("Entry has been removed from your database.");
    };
}
```

14. La API File

La especificación de HTML5 fue desarrollada considerando cada aspecto necesario para la construcción y funcionalidad de aplicaciones web. Desde el diseño hasta la estructura elemental de los datos, todo fue cubierto. Y los archivos no podían ser ignorados, por supuesto. Por esta razón, la especificación incorpora la API File.

LA API File comparte algunas características con las API de almacenamiento comentadas en capítulos previos. Esta API posee una infraestructura de bajo nivel, aunque no tan compleja como IndexedDB, y al igual que otras puede trabajar de forma síncrona o asíncrona. La parte síncrona fue desarrollada para ser usada con la API Web Workers (del mismo modo que IndexedDB y otras APIs), y la parte asíncrona está destinada a aplicaciones web convencionales. Estas características nos obligan nuevamente a cuidar cada aspecto del proceso, controlar si la operación fue exitosa o devolvió errores, y probablemente adoptar (o desarrollar nosotros mismos) en el futuro APIs más simples construidas sobre la misma.

API File es una vieja API que ha sido mejorada y expandida. A día de hoy está compuesta por tres especificaciones: API File, API File: Directories & System, y API File: Writer, pero esta situación puede cambiar durante los próximos tiempos por nuevas especificaciones o incluso que se unan algunas de las ya existentes.

Básicamente, la API File nos permite interactuar con archivos locales y procesar su contenido en nuestra aplicación, la extensión la API File: Directories & System provee las herramientas para trabajar con un pequeño sistema de archivos creado específicamente para cada aplicación, y la extensión API File: Writer es para escribir contenido dentro de archivos que fueron creados o descargados por la aplicación.

14.1. DETECTANDO DISPONIBILIDAD DE API FILE

```
// Check for the various File API support.
if (window.File && window.FileReader && window.FileList && window.Blob) {
  // Great success! All the File APIs are supported.
} else {
  alert('The File APIs are not fully supported in this browser.');
}
document.getElementById('files').addEventListener('change', handleFileSelect, false);
```

14.2. LECTURA DE ARCHIVOS (FILEREADER)

Para leer archivos en el ordenador de los usuarios tenemos que usar la interface **FileReader**. Esta interface retorna un objeto con varios métodos para obtener el contenido de cada archivo:

Desde un input FILE el usuario puede seleccionar un archivo para ser procesado. Para detectar esta acción, se puede utilizar una función que escuche el evento change. De este modo, otra función será

ejecutada cada vez que algo cambie en el elemento, en otras palabras, un nuevo archivo sea seleccionado.

La propiedad **files** enviada por el elemento input, al igual que en la API Drag and Drop, es un array que contiene todos los archivos seleccionados. Cuando el atributo **multiple** no está presente en la etiqueta input no es posible seleccionar múltiples archivos, por lo que el primer elemento del array será el único disponible.

Lo primero que se debe hacer para comenzar a procesar el archivo es obtener un objeto FileReader usando el constructor **FileReader()**. En el siguiente paso, se debe registrar un manejador de eventos onload para el objeto lector con el objetivo de detectar cuando el archivo es cargado y podemos comenzar a procesarlo. Finalmente, el método readAsText() leerá el archivo y retornará su contenido en formato texto.

14.2.1. readAsText(archivo, codificación)

El método readAsText se utiliza para leer el contenido de un archivo especificado. Cuando la operación de lectura se ha completado, el readyState se cambia a DONE, el **loadend** se dispara, y el atributo **result** devuelve el contenido del archivo como una cadena de texto.

Para procesar el contenido como texto podemos usar este método. Un evento load será disparado desde el objeto FileReader cuando el archivo sea cargado. El contenido retornado será codificado en UTF-8 a menos que el atributo codificación sea especificado con un valor diferente. Este método intentará interpretar cada byte o una secuencia de múltiples bytes como caracteres de texto.

14.2.2. readAsDataURL(archivo)

El método readAsDataURL es usado para leer el contenido de archivos. Cuando la operación de lectura finaliza, el readyState se convierte en DONE, y el evento loadend es disparado. En ese momento, el atributo **result** contiene la información como una URL representando la información del archivo como una cadena de caracteres codificados en base64. El método readAsDataURL retorna el contenido del archivo en formato **data:url** que puede ser usado luego como fuente de un elemento <img> para mostrar la imagen seleccionada en la pantalla.

14.2.3. readAsBinaryString(archivo)

La información es leída por este método como una sucesión de enteros en el rango de 0 a 255. Este método se asegura que cada byte es leído de manera independiente, sin ningún intento de interpretación. Es útil para procesar contenido binario como imágenes o videos.

14.2.4. readAsArrayBuffer(archivo)

Este método retorna los datos del archivo como datos del tipo ArrayBuffer.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>File API</title>
        <link rel="stylesheet" href="file.css">
        <script type="text/javascript">
            function iniciar(){
                cajadatos=document.getElementById('cajadatos');
                var archivos=document.getElementById('archivos');
                archivos.addEventListener('change', procesar, false);
            }
            function procesar(e){
                var archivos=e.target.files, archivo=archivos[0];
                var lector=new FileReader();
                lector.onload=mostrar;
                lector.readAsText(archivo);
            }
            function mostrar(e){
                var resultado=e.target.result;
                cajadatos.innerHTML=resultado;
            }
            window.addEventListener('load', iniciar, false);
        </script>
        <style>
            #cajaformulario { float: left; padding: 20px; border: 1px solid #999999; }
            #cajadatos      { float: left; width: 500px; border: 1px solid #999999; }
            #cajadatos      { margin-left: 20px; padding: 20px; }
            .directorio     { color: #0000FF; font-weight: bold; cursor: pointer; }
        </style>
    </head>
    <body>
        <section id="cajaformulario">
            <form name="formulario">
                API File
                <p>Archivos:<br><input type="file" name="archivos" id="archivos"></p>
            </form>
        </section>
        <section id="cajadatos">
            No se seleccionaron archivos
        </section>
    </body>
</html>
```

14.3. PROPIEDADES DE ARCHIVOS

En una aplicación real, información como el nombre del archivo, su tamaño o tipo será necesaria para informar al usuario sobre los archivos que están siendo procesados o incluso controlar cuáles son o no son admitidos. El objeto enviado por el elemento input incluye varias propiedades para acceder a esta información:

Propiedad	Descripción
name	Esta propiedad retorna el nombre completo del archivo (nombre y extensión).
size	Esta propiedad retorna el tamaño del archivo en bytes.
type	Esta propiedad retorna el tipo de archivo, especificado en tipos MIME.

```
function procesar(e){
    var archivos=e.target.files;
    cajadatos.innerHTML='';
    var archivo=archivos[0];
    if(!archivo.type.match(/image.*/i)){
        alert('seleccione una imagen');
    } else {
        cajadatos.innerHTML+='Nombre: '+archivo.name+'<br>';
        cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';
        var lector=new FileReader();
        lector.onload=mostrar;
        lector.readAsDataURL(archivo);
    }
}

function mostrar(e){
    var resultado=e.target.result;
    cajadatos.innerHTML+='<img src="'+resultado+'">';
}
window.addEventListener('load', iniciar, false);
```

14.4. FICHEROS BLOB

Además de archivos, la API trabaja con otra clase de fuente de datos llamada blobs. Un blob es un objeto representando datos en crudo. Fueron creados con el propósito de superar limitaciones de Javascript para trabajar con datos binarios. Un blob es normalmente generado a partir de un archivo, pero no es necesariamente eso. Es una buena alternativa para trabajar con datos sin cargar archivos enteros en memoria, y provee la posibilidad de procesar información binaria en pequeñas piezas.

Un objeto blob puede tener múltiples propósitos, pero está enfocado a mejorar el procesado de grandes piezas de datos en crudo o archivos. Para generar blobs desde otros blobs o archivos, la API ofrece el método slice()

14.4.1. slice(comienzo, largo, tipo)

Este método retorna un nuevo blob generado desde otro blob o un archivo. El primer atributo indica el punto de comienzo, el segundo el largo del nuevo blob, y el último es un atributo opcional para especificar el tipo de datos usados.

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var archivos=document.getElementById('archivos');
    archivos.addEventListener('change', procesar, false);
}
function procesar(e){
    var archivos=e.target.files;
    cajadatos.innerHTML='';
    var archivo=archivos[0];
    var lector=new FileReader();
    lector.onload=function(e){ mostrar(e, archivo); };
    var blob=archivo.slice(0,1000);
    lector.readAsBinaryString(blob);
}
function mostrar(e, archivo){
    var resultado=e.target.result;
    cajadatos.innerHTML='Nombre: '+archivo.name+'<br>';
    cajadatos.innerHTML+='Tipo: '+archivo.type+'<br>';
    cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';
    cajadatos.innerHTML+='Tamaño Blob: '+resultado.length+'bytes<br>';
    cajadatos.innerHTML+='Blob: '+resultado;
}
window.addEventListener('load', iniciar, false);
```

En el código anterior, se crea un objeto blob con el método slice(). El blob tiene 1000 bytes de largo y comienza desde el byte 0 del archivo. Si el archivo cargado es más pequeño que 1000 bytes, el blob será del mismo largo del archivo (desde el comienzo hasta el EOF, o End Of File).

Para mostrar la información obtenida por este proceso, se debe registrar un manejador de eventos onload y llamar a la función mostrar() desde una función anónima con la que pasamos la referencia del objeto archivo. Este objeto es recibido por mostrar() y sus propiedades son mostradas en la pantalla.

Las ventajas ofrecidas por blobs son incontables. Podemos crear un bucle para dividir un archivo en varios blobs, por ejemplo, y luego procesar esta información parte por parte, creando programas para subir archivos al servidor o aplicaciones para procesar imágenes, entre otras. Los blobs ofrecen nuevas posibilidades a los códigos Javascript.

14.5. EVENTOS

El tiempo que toma a un archivo para ser cargado en memoria depende de su tamaño. Para archivos pequeños, el proceso se asemeja a una operación instantánea, pero grandes archivos pueden tomar varios segundos en cargar. Además del evento load ya estudiado, la API provee eventos especiales para informar sobre cada instancia del proceso:

14.5.1. loadstart

Este evento es disparado desde el objeto FileReader cuando la carga comienza.

14.5.2. progress

Este evento es disparado periódicamente mientras el archivo o blob está siendo leído.

14.5.3. abort

Este evento es disparado si el proceso es abortado.

14.5.4. error

Este evento es disparado cuando la carga ha fallado.

14.5.5. loadend

Este evento es similar a load, pero es disparado en caso de éxito o fracaso.

14.5.6. Ejemplo

```
function procesar(e){
    var archivos=e.target.files;
    cajadatos.innerHTML='';
    var archivo=archivos[0];
    var lector=new FileReader();
    lector.onloadstart=comenzar;
    lector.onprogress=estado;
    lector.onloadend=function(){ mostrar(archivo); };
    lector.readAsBinaryString(archivo);
}
```

14.6. CONTROL DEL PROGRESO DE UNA LECTURA

Una de las funciones que se pueden disfrutar gratuitamente al utilizar el control de eventos de tipo asíncrono es la de control del progreso de la lectura de un archivo. Esto resulta útil para leer archivos de gran tamaño, detectar errores y saber cuándo se ha completado una lectura.

Los eventos onloadstart y onprogress se pueden utilizar para controlar el progreso de una lectura.

En el ejemplo que aparece a continuación, se muestra una barra de progreso que permite controlar el estado de la lectura. Para ver cómo funciona el indicador de progreso, intenta utilizar un archivo grande o un archivo de una unidad remota.

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>File API</title>
        <style>
          #progress_bar {
            margin: 10px 0;
            padding: 3px;
            border: 1px solid #000;
            font-size: 14px;
            clear: both;
            opacity: 0;
            -moz-transition: opacity 1s linear;
            -o-transition: opacity 1s linear;
            -webkit-transition: opacity 1s linear;
          }
          #progress_bar.loading {
            opacity: 1.0;
          }
          #progress_bar .percent {
            background-color: #99ccff;
            height: auto;
            width: 0;
          }
        </style>
        <script>
          var reader;
          var progress = document.querySelector('.percent');

          function abortRead() {
            reader.abort();
          }

          function errorHandler(evt) {
            switch(evt.target.error.code) {
              case evt.target.error.NOT_FOUND_ERR:
                alert('File Not Found!');
```

```
      break;
    case evt.target.error.NOT_READABLE_ERR:
      alert('File is not readable');
      break;
    case evt.target.error.ABORT_ERR:
      break; // noop
    default:
      alert('An error occurred reading this file.');
  };
}

function updateProgress(evt) {
  // evt is an ProgressEvent.
  if (evt.lengthComputable) {
    var percentLoaded = Math.round((evt.loaded / evt.total) * 100);
    // Increase the progress bar length.
    if (percentLoaded < 100) {
      progress.style.width = percentLoaded + '%';
      progress.textContent = percentLoaded + '%';
    }
  }
}

function handleFileSelect(evt) {
  // Reset progress indicator on new file selection.
  progress.style.width = '0%';
  progress.textContent = '0%';

  reader = new FileReader();
  reader.onerror = errorHandler;
  reader.onprogress = updateProgress;
  reader.onabort = function(e) {
    alert('File read cancelled');
  };
  reader.onloadstart = function(e) {
    document.getElementById('progress_bar').className = 'loading';
  };
  reader.onload = function(e) {
    // Ensure that the progress bar displays 100% at the end.
    progress.style.width = '100%';
    progress.textContent = '100%';
    setTimeout("document.getElementById('progress_bar').className='';", 2000);
  }
```

```
            // Read in the image file as a binary string.
            reader.readAsBinaryString(evt.target.files[0]);
          }

          document.getElementById('files').addEventListener('change', handleFileSelect, false);
        </script>
    </head>
    <body>
        <input type="file" id="files" name="file" />
        <button onclick="abortRead();">Cancel read</button>
        <div id="progress_bar"><div class="percent">0%</div></div>
    </body>
</html>
```

14.7. OTRAS OPERACIONES QUE SE PODRÍAN HACER

Actualmente se puede realizar todo lo mencionado anteriormente aunque también se pueden crear archivos y directorios, mover, copiar, … esto todavía no está estandarizado y, casi ni definido ya que sólo funciona en Chrome.

15. Resumen de elementos nuevos en HTML5

Existen numerosas novedades dentro del HTML5 que se han representado por etiquetas o elementos de diversa índole. Algunas etiquetas son realmente nuevas y otras como EMBED ya existían pero se han incorporado al estándar.

15.1. ETIQUETAS PARA MULTIMEDIA

Algunos elementos nuevos servirán para integrar contenido multimedia, pues sabemos que cada día esos nuevos tipos de información están más presentes en la Web.

- AUDIO: Para insertar sonido dentro de una web.
- VIDEO: Para insertar clips de vídeo.
- EMBED: Para embeber contenido externo de otro tipo, como el traído de diversos plugins que se comercializan actualmente o se comercializarán en el futuro.
- SOURCE: Permite especificar varias fuentes diferentes cuando se insertan elementos en AUDIO y VIDEO.
- TRACK: Permite especificar varias pistas de sonido o vídeo para los elementos AUDIO y VIDEO.

15.2. NUEVOS ELEMENTOS DE FORMULARIO

En el caso del HTML5 y los formularios tenemos que destacar que no solamente se han creado nuevas etiquetas, sino que se ha añadido soporte a las existentes anteriormente. De momento estas son las nuevas etiquetas que nos ofrece.

- METER: Para trabajar con medidas y escalas.
- PROGRESS: Para crear barras de progreso.
- DATALIST: Extensión para crear campos con funcionalidad de autocompletar.
- KEYGEN: Genera claves pública y privada para encriptación.
- OUTPUT: Realizar y mostrar cálculos matemáticos.

15.3. DIBUJOS COMPLETOS EN HTML5, LIENZO DE CANVAS

Luego tenemos una utilidad nueva que merece la pena verla por separado, puesto que nos va a transformar la manera con la que se experimentará la web. Se trata de un lienzo en donde se puede dibujar cualquier cosa e incluso hacer animación compleja. Esto se realiza con la etiqueta CANVAS pero, cabe destacar que, esta etiqueta, simplemente delimita un área de la página dónde se puede dibujar, pero para realizar esos dibujos se tiene que utilizar el lenguaje Javascript, a través del API de Canvas. Para aprender a diseñar en un CANVAS se recomienda la lectura del Manual del Elemento Canvas del HTML5.

15.4. SECCIONES DENTRO DE UNA PÁGINA

Algunas de las nuevas etiquetas nos sirven para decir qué secciones contiene una página.

- ARTICLE: Especifica un artículo, es decir, una unidad de contenido.
- SECTION: Es una sección dentro de un documento.
- HEADER: La cabecera de una página.
- FOOTER: El pie de página o informaciones que formen el pie de una sección.
- ASIDE: Es una parte de la web que muestra contenido accesorio, generalmente colocado en un panel lateral.
- NAV: con el que colocar el navegador principal de una página web.

15.5. OTROS TIPOS DE INFORMACIONES

Hay otras muchas etiquetas que nos sirven para definir qué es el contenido que se escribe dentro.

- BDI: Define una parte del texto que debe ser entendido aparte de la línea de contenido que se esté escribiendo.
- MENU: una lista de opciones que formen parte de un menú.
- COMMAND: Uno de los elementos o botones de un menú de opciones.
- DETAILS: Detalles o información suplementaria que se puede ver u ocultar por el usuario.
- SUMMARY: Encabezamiento para detalles especificados en DETAILS.
- FIGURE: es un contenido que ilustre el artículo, como fotos, diagramas, ilustraciones, etc.
- FIGCAPTION: El pie o explicación de un FIGURE.
- HGROUP: Un grupo de encabezamientos, útil cuando se tiene diversos bloques de encabezamientos del mismo nivel H1, H2...
- MARK: Un texto o información que es remarcable.
- TIME: Para escribir una fecha, una hora o ambas.
- WBR: Define un posible salto de línea.

16. Modernizr

Modernizr es una librería JavaScript que nos permite conocer la compatibilidad del navegador con tecnologías **HTML5** y **CSS3**, lo que nos permitirá desarrollar sitios web que se adapten a las capacidades cada navegador.

Este *framework* es un paquete de detección de las capacidades de un navegador relativas a HTML5 y CSS3, esto es, una librería JavaScript que nos informará cuáles de las funcionalidades de estas tecnologías están disponibles en el navegador del usuario, para utilizarlas, o no, en cada caso.

Sabiendo que nuestro navegador soporta ciertas capacidades de CSS3 o de HTML5, podremos utilizarlas con libertad. De modo contrario, si sabemos que un navegador no es compatible con determinada funcionalidad, podremos implementar variantes que sí soporte y así crear sitios web que se adaptan perfectamente al cliente web de cada visitante.

Existen dos herramientas principales en Modernizr que se pueden utilizar para detectar las funcionalidades que están presentes en un navegador. Una la podemos utilizar a través de JavaScript y otra directamente sobre código CSS. En resumen, con Modernizr podemos detectar las funcionalidades disponibles de CSS3 y HTML5.

16.1. AÑADIR MODERNIZR A UNA PÁGINA

El primer paso consistirá en descargar el código fuente de Modernizr. Se trata de un archivo con código JavaScript que podemos encontrar en dos variantes:

- **Development**: contiene el código fuente completo, sin comprimir y con comentarios. Debemos utilizar esta variante únicamente en desarrollo o cuando queremos acceder a su código, para comprenderlo o ampliarlo.
- **Production**: es recomendable (o más bien obligatorio) utilizar esta variante cuando pasamos a un entornos de producción. Al descargarnos Modernizr, tenemos la posibilidad de generar una librería únicamente con las funcionalidades que queremos detectar, lo que nos permitirá ahorrarnos una importante cantidad de KB innecesarios.

Una vez que hemos descargado nuestra librería, debemos incluirla en el código HTML de la página, de la misma manera que incluimos scripts JavaScript.

```
<script src="/js/lib/vendor/modernizr-custom.min.js"></script>
```

Según podemos leer en la documentación de Modernizr, se aconseja colocar el script dentro del HEAD, porque debe cargarse antes del BODY de la página, debido a un componente que quizás utilicemos, para permitir HTML5 en Internet Explorer, llamado HTML5 Shiv. Además, se recomienda colocarlo después de los estilos CSS para evitar un comportamiento poco deseable llamado FOUC, por el cual puede mostrarse, por un pequeño espacio de tiempo, la página sin los estilos CSS aplicados.
A partir de este momento tendremos disponibles nuestros scripts de detección de funcionalidades así como una serie de clases CSS que nos ayudarán a aplicar estilos solo cuando los navegadores los soporten.

16.2. OBJETO MODERNIZR

Cuando tenemos Modernizr cargado en nuestra página, se crea automáticamente un objeto JavaScript que tiene una serie de propiedades que nos indican si están o no disponibles cada una de las funcionalidades presentes en CSS3 y HTML5. Las mencionadas propiedades contienen simplemente valores booleanos (*true* o *false*) que podemos consultar para saber si están o no disponibles las funcionalidades que deseamos utilizar.
El uso es tan sencillo como se indica a continuación:

```
if (Modernizr.boxshadow) {
   // Podemos aplicar sombras!
} else {
   // La propiedad box-shadow no está disponible
}
```

Aquí estamos consultando la propiedad `boxshadow` del objeto Modernizr. Esta propiedad nos indica si el navegador es compatible con el atributo `box-shadow` de CSS3, que sirve para crear cajas de contenido con sombreado.
Ahora veamos otro ejemplo similar que detectaría si está disponible el elemento `canvas` del HTML5.

```
if (Modernizr.canvas) {
   // Podemos utilizar canvas!
} else {
   // El elemento canvas no está disponible
}
```

Este es un simple ejemplo que comprueba si están disponibles ciertas propiedades y funcionalidades de CSS3 y HTML5. Lógicamente, tendremos que desarrollar las funcionalidades que correspondan en cada caso. El listado completo de propiedades del objeto para la detección de funcionalidades HTML5 y CSS3 se puede encontrar en la propia documentación de Modernizr.

16.3. CLASES CSS EN MODERNIZR

Cuando tenemos Modernizr cargado en nuestra página, éste crea automáticamente una serie de clases que asigna al elemento `html` del documento. Cada una de estas clases hace referencia a las características que soporta el navegador, permitiendo desde CSS adaptar la interfaz según las funcionalidades. Un ejemplo de las clases que crea en un navegador de escritorio moderno:

```
<html lang="en" class=" js flexbox canvas canvastext webgl no-touch geolocation postmessage websqldatabase
indexeddb hashchange history draganddrop websockets rgba hsla multiplebgs backgroundsize borderimage
borderradius boxshadow textshadow opacity cssanimations csscolumns cssgradients cssreflections
csstransforms csstransforms3d csstransitions fontface generatedcontent video audio localstorage
sessionstorage webworkers applicationcache svg inlinesvg smil svgclippaths">
<head>
    ...
    <script src="/js/lib/vendor/modernizr-custom.min.js"></script>
</head>
<body>
    <div class="elemento"></div>
</body>
</html>
```

Todo el proceso es automático y la creación de cada una de esas clases se realizará únicamente en caso de que el navegador sea compatible con cada una de las características de CSS3 y HTML5.

La manera de utilizar estas clases es muy sencilla. Pensemos en que queremos aplicar sombra a un elemento con CSS3 en los navegadores que lo permitan y emular ese sombreado por medio de estilos CSS clásicos en los navegadores que no soporten el atributo `box-shadow`. Lo único que tenemos que hacer es aplicar los estilos *clásicos* al elemento que deseemos:

```
.elemento{
   border-left: 1px solid #ccc;
   border-top: 1px solid #ccc;
   border-bottom: 1px solid #666;
   border-right: 1px solid #666;
}
```

Posteriormente, si nuestro navegador es compatible con el atributo `box-shadow` de CSS3, Modernizr habrá incluido la clase "boxshadow" en el elemento `html`. Nosotros podemos utilizar dicha clase CSS para aplicar estilos que sabemos que solo acabarán afectando a los navegadores que soporten el atributo `box-shadow`.

```
.boxshadow .elemento{
   border: 1px solid #ccc;
```

```
   box-shadow: #999 3px 3px 3px;
}
```

Como se puede ver, hemos sobrescrito la regla CSS para el borde y además hemos aplicado la propiedad `box-shadow`. El efecto conseguido es que los navegadores modernos, que son capaces de procesar el atributo `box-shadow`, mostrarán una sombra CSS3 y los no compatibles con esta propiedad al menos mostrarán unos estilos para los que sí son compatibles.

16.4. EL METODO LOAD()

El método `Modernizr.load()` es una sencilla manera para cargar librerías sólo cuando los usuarios las necesitan, es decir, cuando una funcionalidad en concreto está (o no) disponible. Es una buena manera de ahorrar ancho de banda y mejorar un poco más el rendimiento de la aplicación.

Por ejemplo, pensemos en que estamos desarrollando una aplicación basada en el API de geolocalización de HTML5. Con Modernizr podemos saber si el navegador ofrece soporte a ese API, mediante la propiedad `Modernizr.geolocation`. Si deseamos cargar unos recursos u otros dependiendo de la disponibilidad de esta funcionalidad, el método `Modernizr.load()` nos podrá ahorrar algo de código fuente y de paso acelerar nuestra página en algunas ocasiones. Con éste método podemos indicar a los navegadores que no soporten ese API que carguen el *polyfill* correspondiente, de modo que se pueda utilizar esa característica de HTML5 en ellos también.

La sintaxis de `Modernizr.load()` es bastante sencilla de comprender. Un ejemplo sencillo:

```
Modernizr.load({
  test: Modernizr.geolocation,
  yep : 'geo.js',
  nope: 'geo-polyfill.js'
});
```

En este ejemplo, se decide que *script* se debe cargar, en función de si la geolocalización esta soportada por el navegador o no. De esta manera, nos ahorramos el tener que descargar código que el navegador no soporta y por lo tanto, no necesita.

`Modernizr.load()` es simple y eficaz, pero podemos utilizarlo de manera más compleja, como se muestra en el siguiente ejemplo:

```
// Give Modernizr.load a string, an object, or an array of strings and objects
Modernizr.load([
  // Presentational polyfills
  {
    // Logical list of things we would normally need
    test : Modernizr.fontface && Modernizr.canvas && Modernizr.cssgradients,
    // Modernizr.load loads css and javascript by default
```

```
    nope : ['presentational-polyfill.js', 'presentational.css']
  },
  // Functional polyfills
  {
    // This just has to be truthy
    test : Modernizr.websockets && window.JSON,
    // socket-io.js and json2.js
    nope : 'functional-polyfills.js',
    // You can also give arrays of resources to load.
    both : [ 'app.js', 'extra.js' ],
    complete : function () {
      // Run this after everything in this group has downloaded
      // and executed, as well everything in all previous groups
      myApp.init();
    }
  },
  // Run your analytics after you've already kicked off all the rest
  // of your app.
  'post-analytics.js'
]);
```

16.5. COMPATIBILIDAD CON ETIQUETAS SEMÁNTICAS DEL HTML5 CON HTML5SHIV

16.5.1. Qué es HTML5Shiv

HTML5Shiv es una solución JavaScript que permite dar estilo a los elementos de HTML5 en las versiones de Internet Explorer anteriores a la versión 9.

Se podría decir que es un *"hack"* necesario para utilizar los nuevos elementos semánticos de HTML5 (header, nav, hgroup, article,...) en Internet Explorer 8 y anteriores.

Esta solución se creó porque cuando se van a utilizar las nuevas etiquetas para estructurar una página web, la presentación final depende totalmente de los estilos que se apliquen a esas etiquetas. Si IE 8 u otra versión anterior no reconoce estas etiquetas no se les podrá aplicar los estilos.

Si por ejemplo creásemos un documento HTML con el código de la página siguiente podríamos observar que en versiones anteriores a IE9 se vería como si no tuviese estilos, mientras que lo mostramos en una versión IE9+ sí que se mostraría de manera correcta. Resultado sería:

IE9 y superiores	Este es el contenido de esta etiqueta HTML5
IE9 e inferiores	Este es el contenido de esta etiqueta HTML5

```
<header>Este es el contenido de esta etiqueta HTML5</header>
header {
    border: 1px solid black;
    background-color: #f0f0f0;
    font-size: 1.5em;
    padding: 20px 12px;
    color: red;
}
```

Modernizr, como ya sabrán muchos, nos ofrece soporte a etiquetas nuevas que han aparecido en la versión más reciente del lenguaje de la web, como son las etiquetas semánticas y etiquetas estructurales del HTML5.

16.5.2. Script html5shiv para habilitar HTML5 en Internet Explorer

El componente que queremos comentar en este caso, que forma parte de Modernizr, pero que también podemos obtener por separado se llama html5shiv. Incorpora, gracias a la inclusión de Javascript, soporte para los nuevos elementos o etiquetas de HTML5.

Lo podemos encontrar de manera predeterminada en las descargas que realizamos con Modernizr, pero también lo podemos descargar de manera independiente en https://github.com/aFarkas/html5shiv.

También lo encontraremos disponible comprimido y listo para usar en IE, en otros dos repositorios diferentes de Google Code y http://code.google.com/p/html5shim/.

El script no solo provoca que versiones antiguas de Internet Explorer entiendan las etiquetas nuevas del HTML5, sino que también incorpora unos estilos predeterminados para esos nuevos elementos, como los ARTICLE o SECTION.

Aunque el caso más relevante de incompatibilidad con HTML5 es el de Internet Explorer, otros clientes web también tienen bajo soporte a las nuevas etiquetas incorporadas en la última versión del lenguaje de la web. Es por ello que este proyecto también ofrece soporte a Safari 4.x, Safari basado en iOS 3.x, y Firefox 3.x.

16.5.3. Usar html5shiv de manera independiente

Si queremos utilizar esta librería por separado (o sea, sin formar parte del paquete de *scripts* Modernizr) basta con que insertemos el siguiente código en la cabecera de nuestro documento HTML:

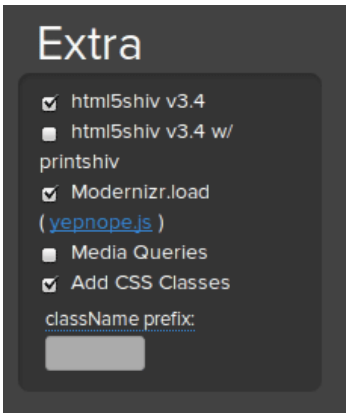
```
<!--[if lt IE 9]>
    <script src="dist/html5shiv.js"></script>
<![endif]-->
```

Esto hace que, mediante un condicional que sólo lee Internet Explorer, se cargue el script únicamente para los navegadores que lo necesitarían.

16.6. USAR HTML5SHIV COMO PARTE DE MODERNIZR

Si estamos pensando en usar diversas capacidades del HTML5 y CSS3, además de las etiquetas nuevas del lenguaje, podemos considerar descargarlo como uno de los paquetes incorporados a la super-librería Modernizr. Ello nos puede ayudar a compatibilizar muchas otras características de la nueva versión del lenguaje para la web.

En este caso tendríamos que descargar Modernizr asegurándonos que se incluye html5shiv. De manera predeterminada, estas librerías ya se encuentran activas en la descarga de Modernizr, por lo que realmente no tendremos que hacer mucho esfuerzo.



De todos modos, en la página de descarga podemos fijarnos que esté seleccionada html5shiv, que se encuentra en el recuadro "Extra".

16.7. PROBAR LAS CAPACIDADES DE MODERNIZR Y SU “MAGIA”

Cuando tenemos Modernizr cargado en nuestra página, se crea automáticamente un objeto Javascript que tiene una serie de propiedades que nos informan sobre si están o no disponibles cada una de las funcionalidades presentes en CSS3 y HTML5. Las mencionadas propiedades contienen simplemente valores boleanos (verdadero o falso) que podemos evaluar para saber si están o no cada una de las funcionalidades que deseemos detectar.

El uso es tan sencillo como esto:

```
if (Modernizr.boxshadow){
   alert("SI sombra caja");
} else {
   alert("NO sombra caja");
}
if (Modernizr.canvas){
   alert("SI canvas");
} else {
   alert("NO canvas");
}
```

La "magia" CSS de Modernizr se basa en la creación en línea de una serie de clases CSS. Modernizr se encargará de crear esas clases y colocarlas en la etiqueta BODY de la página. Todo el proceso es automático y la creación de cada una de esas clases se realizará únicamente en caso de que el navegador sea compatible con cada una de las características de CSS3.

16.8. REGLA @FONT-FACE Y COMPATIBILIDAD

La regla `@font-face` sirve para definir tipografías para textos aunque éstas no estén presentes en el sistema del usuario. Esta regla es propia de CSS3 y para utilizarla se debe subir el archivo de la tipografía al servidor, o bien utilizar algún CDN de fuentes como Google Fonts o TypeKit.

El siguiente código CSS accede a la fuente "Tulpen One" alojada en Google Fonts:

```
@import url(http://fonts.googleapis.com/css?family=Tulpen+One);
```

Si se accede a través del navegador directamente a la URL de la declaración anterior se podrá ver el código CSS real que se está importando y dónde encontramos la regla susodicha regla `@font-face`.

Una vez importada la fuente, ya sde puede utilizar en los elementos que deseemos, por ejemplo:

```
h1 {
    font-family: 'Tulpen One', serif;
    font-size: 1.5em;
    color: #296033;
}
```

La fuente 'Tulpen One' es más pequeña de lo normal, por lo que el tamaño 1.5em no sería muy bueno para un titular H1 (se vería demasiado pequeño para que visualmente se entienda como un encabezamiento de nivel 1). Es por ello que sería interesante conocer si nuestro navegador acepta o aceptó la fuente 'Tulpen One', para que en ese caso podamos aumentar el tamaño del texto o no.

Es justamente aquí donde podemos aprovechar las bondades de Modernizr, ya que nos permite saber si la regla de estilos @font-face está o no presente en nuestro navegador.

Si @font-face es compatible con nuestro navegador, entonces Modernizr creará, en la etiqueta BODY, una clase llamada "fontface" que se podrá utilizar para definir estilos para esos navegadores web que lo permitan.

```
.fontface h1{
   font-size: 2.5em;
}
```

Como se puede observar hemos definido un tamaño de letra mayor que nos servirá para que los encabezamientos, a los que habíamos definido la tipografía "Tulpen One" tengan un tamaño de letra más adecuado.

16.8.1. Fuentes de Iconos (FontAweSome)

FontAweSome es una colección de iconos vectoriales escalables que permiten, entre otras cosas, personalizar - el tamaño, color, sombra,… a través de CSS.

Las fuentes cargadas a través de servicios CDN pueden no funcionar en todos los navegadores, es por ello que, en muchas ocasiones es mejor descargarlas al servidor e importarlas en todos los formatos posibles como es el caso de FontAweSome. Veamos con un ejemplo:

```
@font-face { font-family:'FontAwesome';
   src:url('./fonts/fontawesome-webfont.eot?v=4.1.0');
   src:url('./fonts/fontawesome-webfont.eot?#iefix&v=4.1.0') format('embedded-opentype'),
           url('./fonts/fontawesome-webfont.woff?v=4.1.0') format('woff'),
           url('./fonts/fontawesome-webfont.ttf?v=4.1.0') format('truetype'),
           url('./fonts/fontawesome-webfont.svg?v=4.1.0#fontawesomeregular') format('svg');
   font-weight:normal;
   font-style:normal;
}
```

17. Referencias

http://librosweb.es/
http://www.w3schools.com
https://es.wikipedia.org/
El gran libro de HTML5 y CSS3