



Diseño gráfico orientado a la Web

Modulo: Diseño de Página Web

Tema 5: Formularios y APIs de HTML5

Ing. Jorge Centeno Borge
Correo-e: georgejzcb@yahoo.es



Universidad Nacional
Autónoma de
Nicaragua, León

TEMA 5: FORMULARIOS Y APIs DE HTML5

CONTENIDO

CAPÍTULO 1: FORMULARIOS EN HTML5

- Las limitaciones de los formularios de HTML4
- Números, rangos, fechas y horas
- Validación
- Email y URLs
- Elementos para la retroalimentación de los usuarios
- Controles de formulario menos comunes
- Nuevos atributos para los elementos INPUT
- La protección de la información privada con el atributo autocomplete
- Soporte del navegador y detección de características de HTML5

CAPÍTULO 2: AUDIO Y VIDEO

- Audio y video en webs modernas
- El elemento <video>
- El elemento <audio>
- Controlando audio y video con JavaScript
- Integrando elementos multimedia con otro contenido

CAPÍTULO 3: APIs DE HTML5

- Canvas
- Drag and Drop
- Geolocation
- Storage
- File
- Communication

Web Workers

History

Offline

BIBLIOGRAFÍA

Niederst Robbins, Jennifer. **Learning Web Design**. Fourth Edition.

Gauchat, Juan Diego. **El gran libro de HTML5, CSS3, y Javascript**. Marcombo.

Clark, R, Studholme, O, Murphy, C y Manian, D. **Beginning HTML5 and CSS3**. 2012

Paganotti, S. **Designing Next Generation Web Projects with CSS3**. 2013

Crowther, R. **Hello! HTML5 & CSS3**. ©2013 by Manning Publications

CONTENIDO

Contenido	2
CAPÍTULO 1: FORMULARIOS EN HTML5.....	6
1.1 Las limitaciones de los formularios de HTML4.....	7
1.2 Números, rangos, fechas y horas	7
1.3 Validación.....	10
1.4 Email y URLs.....	12
1.5 Elementos para la retroalimentación de los usuarios.....	12
1.6 Controles de formulario menos comunes	14
1.7 Nuevos atributos para los elementos INPUT	15
1.8 La protección de la información privada con el atributo autocomplete	16
1.9 Soporte del navegador y detección de características de HTML5.....	16
CAPÍTULO 2: AUDIO Y VIDEO	18
2.1 Audio y video en webs modernas	19
2.2 El elemento <video>	19
2.3 El elemento <audio>	23
2.4 Controlando audio y video con JavaScript.....	25
2.5 Integrando elementos multimedia con otro contenido.....	26
CAPÍTULO 3: APIs DE HTML5.....	28
3.1 Api Canvas	29
El elemento <canvas>	29
Dibujando en el lienzo.....	30
Creando trazados	33
Texto	39
Sombras	41
Transformaciones	41
Restaurando el estado.....	43
3.2 Drag and Drop	44
3.3 Geolocation.....	44
3.4 Storage.....	44
3.5 File	45
3.6 Comunicación.....	45
3.7 Web Workers	45
3.8 History.....	45
3.9 Offline	45



CAPÍTULO 1: FORMULARIOS EN HTML5

VEREMOS:

- 1.1 Las limitaciones de los formularios de HTML4
- 1.2 Números, rangos, fechas y horas
- 1.3 Validación
- 1.4 Email y URLs
- 1.5 Elementos para la retroalimentación de los usuarios
- 1.6 Controles de formulario menos comunes
- 1.7 Nuevos atributos para los elementos INPUT
- 1.8 La protección de la información privada con el atributo autocomplete
- 1.9 Soporte del navegador y detección de características de HTML5

1.1 LAS LIMITACIONES DE LOS FORMULARIOS DE HTML4

HTML4 tiene una variedad pobre en cuanto a tipos de elementos de entrada: tres modos de introducir texto y tres formas de seleccionar de una lista predefinida de opciones. Repasemos lo que está disponible en HTML4 antes de aprender acerca de las nuevas características disponibles en HTML5:

La entrada de texto es el caballo de batalla de los formularios HTML4:

```
<input type="text" value="abc">
```

Por lo general, cuando no se puede predecir lo que el usuario va a querer entrar, pero se sabe que va a ser bastante corto, se tiene que utilizar una entrada de tipo texto. Esto incluye nombres de usuario, fechas y horarios, términos de búsqueda, direcciones de email, direcciones URL, números de teléfono, moneda, números de tarjetas de crédito, y los valores numéricos simples.

Si el usuario tiene que elegir entre un número limitado de valores posibles, se puede utilizar un elemento SELECT. Un elemento <textarea> es para grandes cantidades de texto libre, cuando se espera que los párrafos sean más que unas pocas palabras:

```
<select>
<option selected> Opción 1</option>
<option>Opción 2</option>
<option>Opción 3</option>
</select>
```

Una alternativa para los elementos <select> son los radio buttons y los checkbox.

Por último, se muestra el elemento <fieldset> con su <legend>, una entrada fileupload, y una entrada submit:

```
<fieldset title="Other form elements">
<legend>Example</legend>
<label>
Upload file
<input type="file" name="name">
</label>
</fieldset>
<input type="submit">
```

Los elementos <fieldset> y <legend> son útiles para agrupar conjuntos de controles juntos en formularios grandes.

Pero la solución para los formularios de HTML4 requiere una serie de compromisos. Utiliza entradas de texto para fines tales como números y direcciones de correo electrónico. A continuación se mostrarán los nuevos controles de formulario proporcionados por HTML5, que son más adecuados para dichas entradas.

1.2 NÚMEROS, RANGOS, FECHAS Y HORAS

INPUT TYPE: NUMBER



El tipo **number** se utiliza para campos de entrada que deben contener un valor numérico.

También puede establecer restricciones acerca de qué números son aceptados.

```
<form action="demo_form.asp">
  Cantidad (entre 1 y 5): <input type="number" name="quantity" min="1" max="5">
  <input type="submit">
```

```
</form>
```

Result:

Cantidad (entre 1 y 5):

Utilice los siguientes atributos para especificar restricciones:

- max - especifica el valor máximo permitido
- min - especifica el valor mínimo permitido
- paso - especifica los intervalos de números legales
- valor - Especifica el valor predeterminado

INPUT TYPE: RANGE



El tipo **range** se usa para los campos de entrada que deben contener un valor en un rango de números.

También puede establecer restricciones acerca de qué números son aceptados.

```
<form action="demo_form.asp">
  Puntos: <input type="range" name="points" min="1" max="10">
</form>
```

Result:

Puntos:

Utilice los siguientes atributos para especificar restricciones:

- max - especifica el valor máximo permitido
- min - especifica el valor mínimo permitido
- paso - especifica los intervalos de números legales
- valor - Especifica el valor predeterminado

INPUT TYPE: DATE



El tipo **date** permite al usuario elegir una fecha.

```
<form action="demo_form.asp">
  Cumpleaños: <input type="date" name="bday">
  <input type="submit">
</form>
```


Result:

Cumpleaños:

<< < junio 2013 > >>

lun	mar	mié	jue	vie	sáb	dom
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

INPUT TYPE: TIME



El tipo **time** permite al usuario elegir una hora.

```
<form action="demo_form.asp">
Selecciona una hora: <input type="time" name="usr_time">
  <input type="submit">
</form>
```

Result:

Selecciona una hora:

INPUT TYPE: DATETIME-LOCAL



El tipo **datetime-local** permite al usuario seleccionar una fecha y hora (sin zona horaria).

```
<form action="demo_form.asp">
Cumpleaños (fecha y hora): <input type="datetime-local" name="bdaytime">
</form>
```

INPUT TYPE: DATETIME



El tipo **datetime** permite al usuario seleccionar una fecha y hora (con zona horaria).

```
<form action="demo_form.asp">
Cumpleaños (fecha y hora): <input type="datetime" name="bdaytime">
</form>
```

INPUT TYPE: MONTH



El tipo **month** permite al usuario seleccionar un año y mes.

```
<form action="demo_form.asp">
```

```
Cumpleaños (mes y año): <input type="month" name="bdaymonth">
<input type="submit">
</form>
```

INPUT TYPE: WEEK



El tipo **week** permite al usuario seleccionar una semana y año.

```
<form action="demo_form.asp">
Seleccciona una semana: <input type="week" name="week_year">
</form>
```

1.3 VALIDACIÓN

La validación es a menudo un tema crucial en la web, tanto para la seguridad y para el buen funcionamiento general de las aplicaciones, pero es algo en que los creadores de páginas web con frecuencia se equivocan. HTML5 se ha incorporado en funciones la validación de formularios.

EL ATRIBUTO: REQUIRED

La forma más simple de validar es marcar un campo como **required**. En HTML5 esto se hace mediante la adición del atributo **required**. Si una entrada se marca como necesaria, el navegador no debe permitir que el formulario se envíe hasta que el usuario haya proporcionado un valor.

El atributo **required** funciona con los siguientes tipos de entrada: text, search, url, tel, email, password, date pickers, number, checkbox, radio, and file.

Ejemplo:

```
Usuario: <input type="text" name="username" required>
```

EL ATRIBUTO: PATTERN

El atributo **pattern** especifica una expresión regular contra la cual el valor del elemento `<input>` se compara.

Nota: El atributo **pattern** funciona con los siguientes tipos de entrada: texto, búsqueda, url, teléfono, correo electrónico y contraseña.

Use el **title** del input para indicar al usuario el patrón correcto.

```
<form action="demo_form.asp">
  Código de país: <input type="text" name="country_code" pattern="[A-Za-z]{3}" title="El
código de país consta de tres letras">
  <input type="submit">
</form>
```

Result:

Código de país:

El código de país consta de tres letras

VALIDACIÓN Y CSS

Además del soporte visible en el navegador para la validación, HTML5 ofrece elementos para CSS y JavaScript. Estos permiten proporcionar retroalimentación visual inmediata. CSS tiene dos pseudo-clases que le permiten ofrecer diferentes estilos en función de si los elementos son válidos o no válidos.

He aquí un simple par de reglas CSS para poner un contorno verde alrededor de los controles válidos y un esquema de puntos rojos alrededor de los controles no válidos:

```
<style>
  input:valid {
    outline: 5px solid green;
  }
  input:invalid {
    outline: 5px dashed red;
  }
</style>
```

```
<form action="demo_form.asp">
  <input type="number" required><br><br>
  <input type="number" min="4" value="4"><br><br>
  <input type="number" min="4" value="3"><br><br>
</form>
```

Result:

También hay soporte CSS para estilizar controles necesarios de manera diferente a través de pseudo-clases:

```
<style>
  input:required {
    outline: 5px dashed blue;
  }
  input:optional {
    outline: 5px solid green;
  }
</style>
```

```
<form action="demo_form.asp">
  <input type="number" required><br><br>
  <input type="number"><br><br>
</form>
```

Result:

DESACTIVAR LA VALIDACIÓN

A veces desea que el usuario pueda enviar el formulario sin desencadenar validation. Para hacer esto, HTML5 ofrece dos nuevos atributos: `novalidate` y `formnovalidate`.

El atributo **`novalidate`** puede aplicarse al elemento `<form>` sí mismo, mientras que el atributo `formnovalidate` afecta a la entrada del formulario, pero sólo debe aplicarse a un botón de tipo **Submit**:

```
<form action="demo_form.asp" novalidate>
  E-mail: <input type="email" name="user_email">
  <input type="submit">
</form>
```

```
<form action="demo_form.asp">
  E-mail: <input type="email" name="userid"><br>
  <input type="submit" value="Submit"><br>
  <input type="submit" formnovalidate value="Submit sin validación">
</form>
```

1.4 EMAIL Y URLS

INPUT TYPE: EMAIL



El tipo **email** se utiliza para campos de entrada que deben contener una dirección de correo electrónico.

```
<form action="demo_form.asp">
  E-mail: <input type="email" name="email">
  <input type="submit">
</form>
```

INPUT TYPE: URL



El tipo **url** se utiliza para campos de entrada que deben contener una dirección URL.

```
<form action="demo_form.asp">
  Escriba una URL: <input type="url" name="homepage"><br>
  <input type="submit">
</form>
```

1.5 ELEMENTOS PARA LA RETROALIMENTACIÓN DE LOS USUARIOS

INPUT TYPE: OUTPUT

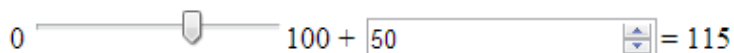


El elemento `<output>` representa el resultado de un cálculo (como uno realizado por un script).

```
<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">0
<input type="range" id="a" value="50">100 +
<input type="number" id="b" value="50">=>
<output name="x" for="a b"></output>
```

```
</form>
```

Result:



EL ELEMENTO <PROGRESS>



Consejo: Utilice la etiqueta <progress> conjuntamente con JavaScript para mostrar el progreso de una tarea.

Nota: La etiqueta <progress> no es adecuada para la representación de un indicador (por ejemplo, uso de espacio en disco o relevancia de un resultado de consulta). Para representar un indicador, utilice la etiqueta <meter> en su lugar.

```
0<progress value="5" min="0" max="9">5 de 9</progress>
```

Result:



EL ELEMENTO <METER>



La etiqueta <meter> define una medida escalar dentro de un rango conocido, o un valor fraccionario. Esto también se conoce como indicador.

Ejemplos: uso de disco, la importancia de un resultado de la consulta, etc

Nota: La etiqueta <meter> no se debe utilizar para indicar el progreso (como en una barra de progreso). Para las barras de progreso, utilice la etiqueta <progress>.

```
<p>Muestre un indicador:</p>
<meter value="2" min="0" max="10">2 de 10</meter><br>
<meter value="0.6">60%</meter>
```

Result:



```
<label for="exmeter">Meter</label>
<meter id="exmeter" value="3" min="1" max="10" high="8" low="2" optimum="3">
3 de 10
</meter>
```

Result:



Note el uso de los atributos **high** y **low**-si valor invade uno de ellos, tiene un efecto visible:

```
<label for="exmeter">Meter</label>
<meter id="exmeter" value="9" min="1" max="10" high="7" low="3" optimum="5">
9 de 10
</meter>
```

Result:

Meter 

```
<label for="exmeter">Meter</label>
<meter id="exmeter" value="2" min="1" max="10" high="7" low="3" optimum="5">
2 de 10
</meter>
```

Result:

Meter 

Al igual que con el elemento **<progress>**, los navegadores que no soportan el elemento **<meter>** visualizan el texto contenido entre las etiquetas de apertura y cierre. Aunque utilizamos el texto aquí, puede incluir una imagen o incluso un poco de SVG que se asemeja más a la prestación de los navegadores que admiten **<meter>**.

1.6 CONTROLES DE FORMULARIO MENOS COMUNES

INPUT TYPE: TEL



Definir un campo para introducir un número de teléfono:

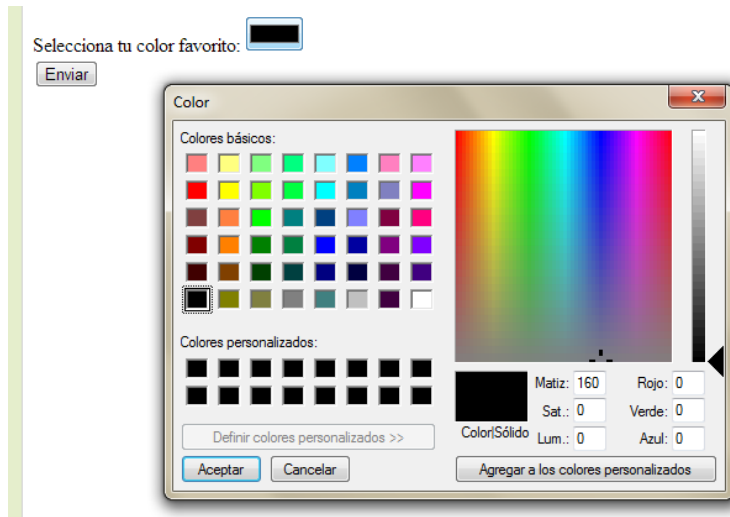
```
Teléfono: <input type="tel" name="usrtel">
```

INPUT TYPE: COLOR



Permite seleccionar un color

```
<form action="demo_form.asp">
  Selecciona tu color favorito: <input type="color" name="favcolor"><br>
  <input type="submit">
</form>
```



EL ELEMENTO <KEYGEN>



El propósito del elemento <keygen> es proporcionar una forma segura para autenticar a los usuarios.

La etiqueta <keygen> especifica un campo generador de par de claves en un formulario.

Cuando se envía el formulario, se generan dos claves, una pública y otra privada.

La clave privada se almacena localmente, y la clave pública se envía al servidor. La clave pública se podría utilizar para generar un certificado de cliente para autenticar el usuario en el futuro.

```
<form action="demo_keygen.asp" method="get">
Username: <input type="text" name="usr_name">
Encryption: <keygen name="security">
<input type="submit">
</form>
```

1.7 NUEVOS ATRIBUTOS PARA LOS ELEMENTOS INPUT

Además de los nuevos controles de formularios en HTML5, los controles de formulario de HTML4 existentes se han ampliado con nuevos atributos.

Ya has visto algunos de ellos en la sección de validación, "El atributo required", donde cubrimos atributos como **required** y **pattern**, pero muchos otros se pueden aplicar a la mayoría de los elementos <input>: placeholder, autofocus, y autocomplete.

ATRIBUTO <INPUT> PLACEHOLDER



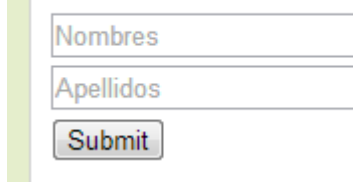
El atributo **placeholder** especifica una pista corta que describe el valor esperado de un campo de entrada (por ejemplo, un valor de la muestra o una breve descripción del formato esperado).

La pista se muestra en el campo de entrada antes de que el usuario introduzca un valor.

Nota: El atributo **placeholder** trabaja con los siguientes tipos de entrada: text, search, url, tel, email, y password.

```
<form action="demo_form.asp">
  <input type="text" name="fname" placeholder="Nombres"><br>
  <input type="text" name="lname" placeholder="Apellidos"><br>
  <input type="submit" value="Submit">
</form>
```

Result:



ATRIBUTO <INPUT> AUTOFOCUS



El atributo **autofocus** es un atributo booleano.

Cuando está presente, especifica que un elemento <input> debe recibir automáticamente el enfoque cuando se carga la página.

```
Nombre:<input type="text" name="fname" autofocus>
```

1.8 LA PROTECCIÓN DE LA INFORMACIÓN PRIVADA CON EL ATRIBUTO AUTOCOMPLETE

El atributo **autocomplete** especifica si un formulario o campo de entrada debe tener la función **autocomplete** encendido o apagado.

Cuando **autocomplete** está activada, el navegador automáticamente completa los valores basados en los valores que el usuario ha escrito antes.

Consejo: Es posible tener autocompletar encendido "on" en el formulario, y "off" para campos de entrada específicos, o viceversa. Nota: El atributo **autocomplete** trabaja con <form> y los siguientes tipos <input>: text, search, url, tel, email, password, datepickers, range, y color.

```
<form action="demo_form.asp" autocomplete="on">
  Nombres:<input type="text" name="fname"><br>
  Apellidos: <input type="text" name="lname"><br>
  E-mail: <input type="email" name="email" autocomplete="off"><br>
  <input type="submit">
</form>
```

1.9 SOPORTE DEL NAVEGADOR Y DETECCIÓN DE CARACTERÍSTICAS DE HTML5

A diferencia de los elementos estructurales que vimos en la clase anterior, los nuevos elementos de formulario tienen comportamientos asociados más complejos y APIs. Con estos requisitos más complejos, no es sorprendente que el soporte no sea tan avanzado como debería ser. La siguiente tabla muestra el nivel de soporte que se conoce, en las versiones más actuales y, de todos los principales navegadores.






											
	12	14	4	6	8	9	10	11.1	11.5	5	5.1
Input types	•	•	•	•			•	•	•	•	•
Validation API	•	•	•	•			•	•	•	•	•
Placeholder	•	•	•	•			•	•	•	•	•
Autofocus	•	•	•	•			•	•	•	•	•
Input UI								•	•		
Range	•	•						•	•	•	•
Meter	•	•						•	•		
Progress	•	•		•				•	•		
Output		•	•	•			•	•	•		•

IMAGEN 1: SOPORTE DE NAVEGADORES. TABLA TOMADA DE LEARNING WEB DESIGN. FOURTH EDITION

CAPÍTULO 2: AUDIO Y VIDEO

VEREMOS:

Audio y video en webs modernas

El elemento <video>

El elemento <audio>

Controlando audio y video con JavaScript

Integrando elementos multimedia con otro contenido

2.1 AUDIO Y VIDEO EN WEBS MODERNAS

El soporte de multimedia nativo es una de las características más conocidas de HTML5, así como una de las más controvertidos.

Los elementos multimedia son partes clave de la web moderna. Para muchos sitios, el vídeo y el audio son parte del contenido integral al igual que el texto y las imágenes, y en algunos casos, es más importante.

A pesar de su creciente importancia, HTML4 no ofrece un método propio para agregar audio o vídeo a una página web. Esto hace la incrustación de audio y video relativamente compleja. Compare el código necesario para añadir una imagen a una página web con el que por lo general requiere añadir un video.

Imagen	Video
<pre></pre>	<pre><object classid="clsid:d27cdb6e-ae6d-11cf-96b8- 444553540000" codebase="http://download.macromedia.com/ pub/shockwave/cabs/flash/ swflash.cab#version=6,0,40,0" width="320" height="240" id="myvideoname"> <param name="movie" value="myvideo.swf"> <param name="quality" value="high"> <param name="bgcolor" value="#ffffff"> <embed href="myvideo.swf" quality="high" bgcolor="#ffffff" width="320" height="240" name="myvideoname" type="application/x-shockwave-flash" pluginspage="http://www.macromedia.com/ go/getflashplayer"> </embed> </object></pre>

Debido a que no hay soporte nativo para audio y video, los desarrolladores web han tenido que recurrir a los plugins (complementos) del navegador. La web se ha asentado principalmente en Adobe Flash como un estándar de facto, pero como el código anterior muestra, esto es todavía bastante más complejo que poner una imagen en una página.

Y eso no es todo el código que se necesita: para agregar controles como Reproducir y Pausa, debe haber código escrito dentro de Flash, e incluso más código si el reproductor tiene que estar integrado en otro contenido de la página.

Una de las razones de la popularidad de YouTube es que reduce la complejidad de la visualización de vídeo en la web, en lugar de hacer todo el trabajo usted mismo, se sube el video a YouTube y luego se copia y pega un código. Pero HTML debería hacer que sea sencillo y sin tener la necesidad de un sitio de terceros. Este es un problema solucionado en HTML5 con la introducción de los elementos <audio> y <video>.

2.2 EL ELEMENTO <VIDEO>

► Show options = Supported = Not supported = Partially supported = Support unknown

Video element - Working Draft

Method of playing videos on webpages (without requiring a plug-in)

Usage stats: Global

	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	BlackBerry Browser
Support:								84.61%	
Partial support:								0.28%	
Total:								84.89%	
Sub-features:									
WebM/VP8 video format									
MPEG-4/H.264 video format									
Ogg/Theora video format									

Una de las características más mencionadas de HTML5 fue la capacidad de procesar video. El entusiasmo nada tenía que ver con las nuevas herramientas provistas por HTML5 para este propósito, sino más bien con el hecho de que desde los videos se volvieron una pieza esencial de Internet, todos esperaban soporte nativo por parte de los navegadores. Era como que todos conocían la importancia de los videos excepto aquellos encargados de desarrollar las tecnologías para la web.

Pero ahora que ya disponemos de soporte nativo para videos e incluso un estándar que nos permitirá crear aplicaciones de procesamiento de video compatibles con múltiples navegadores, podemos comprender que la situación era mucho más complicada de lo que nos habíamos imaginado. Desde codificadores hasta consumo de recursos, las razones para no implementar video de forma nativa en los navegadores eran mucho más complejas que los códigos necesarios para hacerlo.

A pesar de estas complicaciones, HTML5 finalmente introdujo un elemento para insertar y reproducir video en un documento HTML. El elemento `<video>` usa etiquetas de apertura y cierre y solo unos pocos parámetros para lograr su función. La sintaxis es extremadamente sencilla y solo el atributo `src` es obligatorio:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Video</title>
</head>
<body>
<section id="reproductor">
<video src="http://minkbooks.com/content/trailer.mp4" controls>
</video>
</section>
</body>
</html>
```

En teoría, el código anterior debería ser más que suficiente. Primero debemos proveer al menos dos archivos diferentes con formatos de video diferentes: OGG y MP4. Esto es debido a que a pesar de que el elemento `<video>` y sus atributos son estándar, no existe un formato estándar de video. Primero, algunos navegadores soportan un codificador de video que otros no, y segundo el codificador utilizado en el formato MP4 (el único soportado por importantes navegadores como Safari e Internet Explorer) se encuentra bajo licencia comercial.

Los formatos OGG y MP4 son contenedores de video y audio. OGG contiene codificadores de video Theora y de audio Vorbis, y los disponibles para el contenedor MP4 son H.264 para video y AAC para audio. En este momento OGG es reconocido por Firefox, Google Chrome y Opera, mientras que MP4 trabaja en Safari, Internet Explorer y también Google Chrome.

EL ELEMENTO <VIDEO>

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

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

En el código, el elemento **<video>** fue expandido. Ahora, dentro de las etiquetas del elemento hay dos elementos **<source>**. Estos nuevos elementos proveen diferentes fuentes de video para que los navegadores puedan elegir. El navegador leerá la etiqueta **<source>** y decidirá cual archivo reproducir de acuerdo a los formatos soportados (MP4 u OGG).

ATRIBUTOS PARA <VIDEO>

El atributo **controls** es uno de varios atributos disponibles para este elemento. Éste, en particular, muestra controles de video provistos por el navegador por defecto. Cuando el atributo está presente cada navegador activará su propia interface, permitiendo al usuario comenzar a reproducir el video, pausarlo o saltar hacia un cuadro específico, entre otras funciones.

Junto con **controls**, también podemos usar los siguientes:

autoplay Cuando este atributo está presente, el navegador comenzará a reproducir el video automáticamente tan pronto como pueda.

loop Si este atributo es especificado, el navegador comenzará a reproducir el video nuevamente cuando llega al final.

poster Este atributo es utilizado para proveer una imagen que será mostrada mientras esperamos que el video comience a ser reproducido.

preload Este atributo puede recibir tres valores distintos: **none**, **metadata** o **auto**. El primero indica que el video no debería ser cacheado, por lo general con el propósito de minimizar tráfico innecesario. El segundo valor, **metadata**, recomendará al navegador que trate de capturar información acerca de la fuente (por ejemplo, dimensiones, duración, primer cuadro, etc...). El tercer valor, **auto**, es el valor configurado por defecto que le sugerirá al navegador descargar el archivo tan pronto como sea posible.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Video</title>
```

```

</head>
<body>
<section id="reproductor">
<video id="medio" width="720" height="400" preload controls
loop poster="http://minkbooks.com/content/poster.jpg">
<source src="http://minkbooks.com/content/trailer.mp4">
<source src="http://minkbooks.com/content/trailer.ogg">
</video>
</section>
</body>
</html>

```

Debido a las diferencias en comportamiento entre un navegador y otro, algunos atributos estarán habilitados o deshabilitados por defecto, y algunos de ellos incluso no trabajarán en algunos navegadores o bajo determinadas circunstancias. Para obtener un control absoluto sobre el elemento **<video>** y el medio reproducido, deberemos programar nuestro propio reproductor de video en Javascript aprovechando los nuevos métodos, propiedades y eventos incorporados en HTML5.

EVENTOS PARA <VIDEO>

HTML5 incorpora nuevos eventos que son específicos de cada API. Para el procesamiento de video y audio, por ejemplo, los eventos fueron incorporados con el objetivo de informar sobre la situación del medio (el progreso de la descarga, si la reproducción del medio finalizó, o si la reproducción del medio es comenzada o pausada, entre otras). Estos son los más relevantes:

progress Este evento es disparado periódicamente para informar acerca del progreso de la descarga del medio. La información estará disponible a través del atributo **buffered**, como veremos más adelante.

canplaythrough Este evento es disparado cuando el medio completo puede ser reproducido sin interrupción. El estado es establecido considerando la actual tasa de descarga y asumiendo que seguirá siendo la misma durante el resto del proceso. Existe otro evento más para este propósito, **canplay**, pero no considera toda la situación y es disparado tan pronto como algunas partes del medio se encuentran disponibles (luego de descargar los primeros cuadros de un video, por ejemplo).

ended Es disparado cuando el reproductor llega al final del medio.

pause Es disparado cuando el reproductor es pausado.

play Es disparado cuando el medio comienza a ser reproducido.

error Este evento es disparado cuando ocurre un error. Es relacionado con el elemento **<source>** correspondiente a la fuente del medio que produjo el error.

MÉTODOS PARA <VIDEO>

Los métodos **play()** y **pause()** son parte de una lista de métodos incorporados por HTML5 para procesamiento de medios. Los siguientes son los más relevantes:

play() Este método comienza a reproducir el medio desde el inicio, a menos que el medio haya sido pausado previamente.

pause() Este método pausa la reproducción.

load() Este método carga el archivo del medio. Es útil en aplicaciones dinámicas para cargar el medio anticipadamente.

canPlayType(formato) Con este método podemos saber si el formato del archivo es soportado por el navegador o no.

PROPIEDADES PARA <VIDEO>

Las siguientes son las más relevantes:

paused Esta propiedad retorna **true** (verdadero) si la reproducción del medio está actualmente pausada o no ha comenzado.

ended Esta propiedad retorna **true** (verdadero) si la reproducción del medio ha finalizado porque se llegó al final.

duration Esta propiedad retorna la duración del medio en segundos.

currentTime Esta es una propiedad que puede retornar o recibir un valor para informar sobre la posición en la cual el medio está siendo reproducido o especifica una nueva posición donde continuar reproduciendo.

error Esta propiedad retorna el valor del error ocurrido.

buffered Esta propiedad ofrece información sobre la parte del archivo que ya fue cargada en el buffer. Nos permite crear un indicador para mostrar el progreso de la descarga. La propiedad es usualmente leída cuando el evento **progress** es disparado. Debido a que los usuarios pueden forzar al navegador a cargar el medio desde diferentes posiciones en la línea de tiempo, la información retornada por **buffered** es un array conteniendo cada parte del medio que ya fue descargada, no sólo la que comienza desde el principio. Los elementos del array son accesibles por medio de los atributos **end()** y **start()**. Por ejemplo, el código **buffered.end(0)** retornará la duración en segundos de la primera porción del medio encontrada en el buffer. Esta propiedad y sus atributos están bajo desarrollo en este momento.

FORMATOS DE VIDEO SOPORTADOS

La etiqueta <video> especifica medios como un clip de película o de otras fuentes de vídeo.

Actualmente, hay 3 formatos de video soportados para el elemento <video>: MP4, WebM y Ogg:

Browser	MP4	WebM	Ogg
Internet Explorer 9+	SI	NO	NO
Chrome 6+	SI	SI	SI
Firefox 3.6+	NO	SI	SI
Safari 5+	SI	NO	NO
Opera 10.6+	NO	SI	SI

Ver más información en:

http://www.w3schools.com/tags/ref_av_dom.asp

http://www.w3schools.com/tags/tag_video.asp

2.3 EL ELEMENTO <AUDIO>

# Audio element - Working Draft									
Method of playing sound on webpages (without requiring a plug-in)									
						*Usage stats:		Global	
						Support:		84.61%	
						Partial support:		0.03%	
						Total:		84.64%	
Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Blackberry Browser
								2.1	
								2.2	
						3.2		2.3	
						4.0-4.1		3.0	
						4.2-4.3		4.0	
	8.0	20.0				5.0-5.1		4.1	7.0
Current	10.0	22.0	26.0	5.1	12.1	6.0-6.1	5.0-7.0	4.2	10.0
Near future	11.0	23.0	28.0	7.0	15.0	7.0			
Farther future		24.0	29.0						

HTML5 provee un nuevo elemento para reproducir audio en un documento HTML. El elemento, por supuesto, es **<audio>** y comparte casi las mismas características del elemento **<video>**.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Audio</title>
</head>
<body>
<section id="reproductor">
<audio src="http://minkbooks.com/content/beach.mp3" controls>
</audio>
</section>
</body>
</html>
```

EL ELEMENTO <AUDIO>

El elemento **<audio>** trabaja del mismo modo y comparte varios atributos con el elemento **<video>**:

src Este atributo especifica la URL del archivo a ser reproducido. Al igual que en el elemento **<video>** normalmente será reemplazado por el elemento **<source>** para ofrecer diferentes formatos de audio entre los que el navegador pueda elegir.

controls Este atributo activa la interface que cada navegador provee por defecto para controlar la reproducción del audio.

autoplay Cuando este atributo está presente, el audio comenzará a reproducirse automáticamente tan pronto como sea posible.

loop Si este atributo es especificado, el navegador reproducirá el audio una y otra vez de forma automática.

preload Este atributo puede tomar tres valores diferentes: **none**, **metadata** o **auto**. El primero indica que el audio no debería ser cacheado, normalmente con el propósito de minimizar tráfico innecesario. El segundo valor, **metadata**, recomendará al navegador obtener información sobre el medio (por ejemplo, la duración). El tercer valor, **auto**, es el valor configurado por defecto y le aconseja al navegador descargar el archivo tan pronto como sea posible.

Una vez más debemos hablar acerca de codificadores, y otra vez debemos decir que el código siguiente debería ser más que suficiente para reproducir audio en nuestro documento, pero no lo es. MP3 está bajo licencia comercial, por lo que no es soportado por navegadores como Firefox u Opera. Vorbis (el codificador de audio del contenedor OGG) es soportado por esos navegadores, pero no por Safari e Internet Explorer.

Por esta razón, nuevamente debemos aprovechar el elemento **<source>** para proveer al menos dos formatos entre los cuales el navegador pueda elegir:

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Reproductor de Audio</title>
</head>
<body>
<section id="reproductor">
<audio id="medio" controls>
<source src="http://minkbooks.com/content/beach.mp3">
<source src="http://minkbooks.com/content/beach.ogg">
</audio>
</section>
</body>
</html>
```

2.4 CONTROLANDO AUDIO Y VIDEO CON JAVASCRIPT

En primer lugar, echemos un vistazo a reproducir y pausar un vídeo. Esto es sencillo, el elemento **<video>** proporciona métodos `play()` y `pause()`:

```
<button
onclick="document.getElementById('myvideo').play();">
Reproducir
</button>
<button
onclick="document.getElementById('myvideo').pause();">
Pausar
</button>
```

Para reproducir un elemento desde una posición específica:

```
function reproducirEn(secs) {
var v = document.getElementById('myvideo');
v.currentTime = secs;
v.play();
}
```

Esta función inicia la reproducción de vídeo desde cualquier punto en el medio, se pasa el punto desde donde se quiere reproducir como parámetro.

A continuación, puede proporcionar botones para iniciar la reproducción desde un punto significativo:

```
<button onclick="reproducirEn(4);">Reproducir desde el segundo 4</button>
<button onclick="reproducirEn(8);">Reproducir desde el segundo 10</button>
```

En el ejemplo, aunque los botones pretenden iniciar la reproducción del vídeo en el cuarto y décimo segundo, el usuario no tiene forma de ver si realmente funciona. Cuando se ocultan los controles, se pierde no sólo los botones de reproducción y pausa, sino también la línea de tiempo. Afortunadamente, HTML5 proporciona un nuevo elemento **<meter>** que es excelente para la medición de la cuanto lleva reproducido vídeo o clip de audio:

```
<meter id="mymeter" min="0"></meter>
```

El valor del **<meter>** debe actualizarse continuamente a medida que se reproduce el video. Para esto se utiliza el evento `timeupdate`, y agregándolo al elemento **<video>** junto al evento `loadeddata`, que se utiliza para capturar el nombre del archivo de video que se está reproduciendo:

```
<video id="myvideo" ontimeupdate="updateTime(this);"
onloadeddata="dataLoaded(this);">
```

Aquí está la función `dataLoaded`. Se ha actualizado para establecer el valor máximo del elemento `<meter>` para que coincida exactamente con la duración del vídeo cargado:

```
function dataLoaded(v) {
  document.getElementById('source').innerHTML = 'Playing file ' +
  v.currentSrc.slice(v.currentSrc.lastIndexOf('/')+1);
  m = document.getElementById('mymeter');
  m.max = v.duration;
  m.value = 0;
}
```

El código para actualizar el elemento `<meter>` es incluso más simple, sólo establece el valor del elemento de `<meter>` a la `currentTime` del elemento `<video>`:

```
function updateTime(v) {
  m = document.getElementById('mymeter');
  m.value = v.currentTime;
}
```

2.5 INTEGRANDO ELEMENTOS MULTIMEDIA CON OTRO CONTENIDO

Los elementos `<video>` y `<audio>` son como cualquier otro elemento de la página web. Pueden tener de estilo con CSS y ser manipulados con JavaScript. Para empezar, echemos un vistazo a la aplicación de transformaciones y transiciones CSS. Las tres imágenes siguientes muestran la misma página web en el transcurso de 10 segundos.



```
div video {
  transition-duration: 10s;
}
div:hover video:nth-child(1) {
  transform-origin: bottom
  right;
  transform: rotate(16.5deg);
}
div:hover video:nth-child(2) {
  transform-origin: top right;
  transform: rotate(33deg);
}
div:hover video:nth-child(3) {
  transform-origin: top left;
  transform: rotate(66deg);
}
```

```
<div>
<video id="myvideo1" width="160" autoplay loop>
<source src="00092.webm" type="video/webm">
<source src="00092.mp4" type="video/mp4">
<source src="00092.low.mp4" type="video/mp4">
```

```
<source src="00092.ogv" type="video/ogg">
No video!
</video>
</div>
```

SOPORTE A NAVEGADORES ANTIGUOS CON VÍDEO FLASH

Es posible tener lo mejor de ambos mundos: el vídeo HTML5 para el navegador que lo soporte y flash para los navegadores que no lo hacen. En su forma más simple, esto es un asunto de envolver el código para Flash en el interior del elemento de <video>:

```
<video id="myvideo" controls>
  <source src="myvideo.webm" type="video/webm">
  <source src="myvideo.mp4" type="video/mp4">
  <source src="myvideo.low.mp4" type="video/mp4">
  <source src="myvideo.ogv" type="video/ogg">
  <object
    classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
    codebase="http://download.macromedia.com/
      pub/shockwave/cabs/flash/
      swflash.cab#version=6,0,40,0"
    width="320" height="240"
    id="myvideoname">
    <param name="movie" value="player.swf">
    <param name="quality" value="high">
    <param name="bgcolor" value="#ffffff">
    <param name="flashvars" value="file=myvideo.mp4">
  </object>
  No video support, <a href="myvideo.webm">try downloading</a>!
</video>
```

Diagram annotations:

- SOURCES FOR HTML5 VIDEO AS NORMAL (points to the <source> tags)
- ADD FLASH CODE IN <video> ELEMENT (points to the <object> tag)
- FLASH MOVIE IS A PLAYER (points to the <param name="movie"> tag)
- VIDEO FILE TO PLAY (points to the <param name="flashvars" value="file=myvideo.mp4"> tag)

Los navegadores que soportan el elemento <video> ignorarán el contenido de reserva, mientras que los navegadores que no soportan el elemento <video>, lo ignorarán y sólo verán el elemento <object> que incorpora el plugin de Flash. Los navegadores que no soportan ni el elemento <video> ni el flash player verán el enlace para descargar el vídeo.

CAPÍTULO 3: APIs DE HTML5

VEREMOS:

Canvas
Drag and Drop
Geolocation
Storage
File
Communication
Web Workers
History
Offline

3.1 API CANVAS

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

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.

EL ELEMENTO <CANVAS>

Este elemento genera un espacio rectangular vacío en la página web (lienzo) en el cual serán mostrados los resultados de ejecutar los métodos provistos por la API. Cuando es creado, 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="lienzo">
<canvas id="lienzo" width="500" height="300">
Su navegador no soporta el elemento canvas
</canvas>
</section>
</body>
</html>
```

Solo es necesario especificar unos pocos atributos para este elemento, como puede ver. 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.

Eso es básicamente todo lo que el elemento **<canvas>** hace. Simplemente crea una caja vacía en la pantalla. Es solo a través de Javascript y los nuevos métodos y propiedades introducidos por la API que esta superficie se transforma en algo práctico.

IMPORTANTE: Por razones de compatibilidad, en caso de que Canvas API no se encuentre disponible en el navegador, el contenido entre las etiquetas **<canvas>** será mostrado en pantalla.

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.

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

Una referencia al elemento **<canvas>** fue almacenada en la variable **elemento** y el contexto de dibujo fue creado por **getContext('2d')**. El método puede tomar dos valores: **2d** y **3d**. Esto es, por supuesto, para ambientes de 2 dimensiones y 3 dimensiones. Por el momento solo 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 malla de píxeles listados en filas y columnas de arriba a abajo e izquierda a derecha, con su origen (el píxel 0,0) ubicado en la esquina superior izquierda del lienzo.

DIBUJANDO EN EL LIENZO

Luego de que el elemento **<canvas>** y su contexto han sido inicializados podemos finalmente comenzar a crear y manipular gráficos. La lista de herramientas provista por la API para este propósito es extensa, desde la creación de simples formas y métodos de dibujo hasta texto, sombras o transformaciones complejas. Vamos a estudiarlas una por una.

DIBUJANDO RECTÁNGULOS

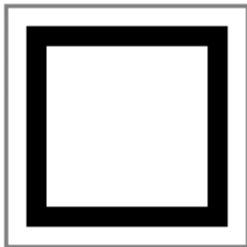
Normalmente el desarrollador deberá preparar la figura a ser dibujada en el contexto (como veremos pronto), pero existen algunos métodos que nos permiten dibujar directamente en el lienzo, sin preparación previa. Estos métodos son específicos para formas rectangulares y son los únicos que generan una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

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** y **y**. Los atributos **ancho** y **alto** declaran el tamaño.

strokeRect(x, y, ancho, alto) Similar al método anterior, éste dibujará un rectángulo vacío (solo su contorno).

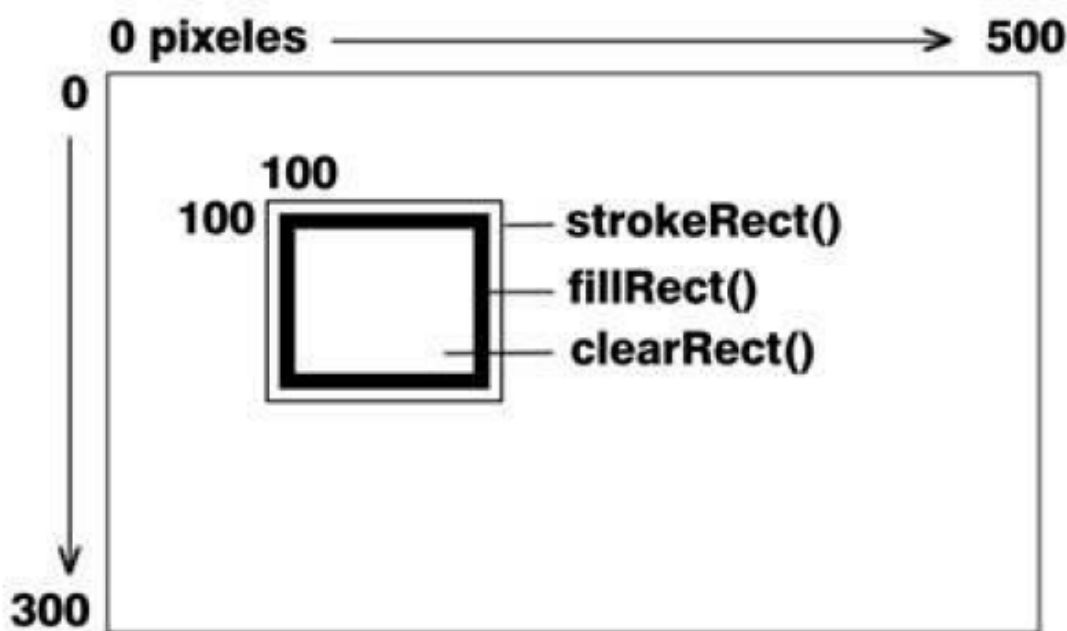
clearRect(x, y, ancho, alto) Este método es usado para substraer píxeles del área especificada por sus atributos. Es un borrador rectangular.

```
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 píxeles (este es un cuadrado de 120 píxeles). 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.



El elemento **<canvas>** es como una malla, con su origen en la esquina superior izquierda y el tamaño especificado en sus atributos. Los rectángulos son dibujados en el lienzo en la posición declarada por los atributos **x** y **y**, y uno sobre el otro de acuerdo al orden en el código (el primero en aparecer en el código será dibujado primero, el segundo será dibujado por encima del anterior, y así sucesivamente). Existe un método para personalizar cómo las figuras son dibujadas en pantalla, pero lo veremos más adelante.

COLORES

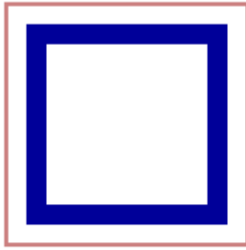
Hasta el momento hemos usado el color otorgado por defecto, negro sólido, pero podemos especificar el color que queremos aplicar mediante sintaxis CSS utilizando las siguientes propiedades:

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

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

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);
}
window.addEventListener("load", iniciar, false);
```



Los colores fueron declarados usando números hexadecimales. Podemos también usar funciones como **rgb()** o incluso especificar transparencia para la figura aprovechando la función **rgba()**. Estos métodos deben ser siempre escritos entre comillas (por ejemplo, **strokeStyle="rgba(255,165,0,1)"**).

Cuando un nuevo color es especificado se vuelve el color por defecto para el resto de los dibujos, a menos que volvamos a cambiarlo más adelante.

A pesar de que el uso de la función **rgba()** es posible, existe otra propiedad más específica para declarar el nivel de transparencia: **globalAlpha**. Su sintaxis es **globalAlpha=valor**, donde **valor** es un número entre 0.0 (totalmente opaco) y 1.0 (totalmente transparente).

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.

createLinearGradient(x1, y1, x2, y2) Este método crea un objeto que luego será usado para aplicar un gradiente lineal al lienzo.

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.

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);
}
window.addEventListener("load", iniciar, false);
```




Creamos el objeto gradiente desde la posición **0,0** a la **10,100**, otorgando una leve inclinación hacia la izquierda. Los colores fueron declarados por el método **addColorStop()** y el gradiente logrado fue finalmente aplicado a la propiedad **fillStyle**, como un color regular.

Note que las posiciones del gradiente son correspondientes al lienzo, no a las figuras que queremos afectar. El resultado es que si movemos los rectángulos dibujados al final de la función hacia una nueva posición, el gradiente para esos triángulos cambiará.

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:

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

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:

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

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.

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();
} window.addEventListener("load", iniciar, false);
```

El código anterior no crea absolutamente nada, solo incorpora los métodos necesarios para iniciar y luego dibujar el trazado en pantalla. Para crear el trazado y la figura real que será enviada al contexto y dibujada en el lienzo, contamos con varios métodos disponibles:

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.

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**.

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.

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.

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.

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:

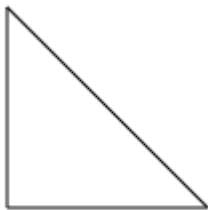
```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
lienzo.moveTo(100,100);
lienzo.lineTo(200,200);
lienzo.lineTo(100,200);
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```



Recomendamos siempre establecer la posición inicial del lápiz inmediatamente después de iniciar el trazado con **beginPath()**. En el código anterior el primer paso fue mover el lápiz a la posición **100,100** y luego generar una línea desde ese punto hasta el punto **200,200**. Ahora la posición del lápiz es **200,200** y la siguiente línea será generada desde aquí hasta el punto **100,200**. Finalmente, el trazado es dibujado en el lienzo como una forma vacía con el método **stroke()**.

Si prueba el código en su navegador, verá un triángulo abierto en la pantalla. Este triángulo puede ser cerrado o incluso rellenado y transformado en una figura sólida usando diferentes métodos, como vemos en el siguiente ejemplo:

```
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.closePath();
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```



El método **closePath()** simplemente agrega una línea recta al trazado, desde el último al primer punto, cerrando la figura.

Usando el método **stroke()** al final de nuestro trazado dibujamos un triángulo vacío en el lienzo. Para lograr una figura sólida, este método debe ser reemplazado por **fill()**:

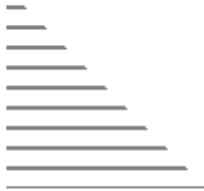
```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
lienzo.moveTo(100,100);
lienzo.lineTo(200,200);
lienzo.lineTo(100,200);
lienzo.fill();
} window.addEventListener("load", iniciar, false);
```



Ahora la figura en la pantalla será un triángulo sólido. El método **fill()** cierra el trazado automáticamente, por lo que ya no tenemos que usar **closePath()** para lograrlo.

Uno de los métodos mencionados anteriormente para dibujar un trazado en el lienzo fue **clip()**. Este método en realidad no dibuja nada, lo que hace es crear una máscara con la forma del trazado para seleccionar qué será dibujado y qué no. Todo lo que caiga fuera de la máscara no se dibujará en el lienzo. Veamos un ejemplo:

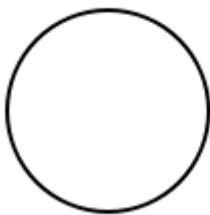
```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
lienzo.moveTo(100,100);
lienzo.lineTo(200,200);
lienzo.lineTo(100,200);
lienzo.clip();
lienzo.beginPath();
for(f=0; f<300; f=f+10){
lienzo.moveTo(0,f);
lienzo.lineTo(500,f);
}
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```



Para mostrar exactamente cómo funciona el método **clip()**, en el código utilizamos un bucle **for** para crear líneas horizontales cada 10 píxeles. Estas líneas van desde el lado izquierdo al lado derecho del lienzo, pero solo las partes de las líneas que caen dentro de la máscara (el triángulo) serán dibujadas.

Ahora que ya sabemos cómo dibujar trazados, es tiempo de ver el resto de las alternativas con las que contamos para crearlos. Hasta el momento hemos estudiado cómo generar líneas rectas y formas rectangulares. Para figuras circulares, la API provee tres métodos: **arc()**, **quadraticCurveTo()** y **bezierCurveTo()**. El primero es relativamente sencillo y puede generar círculos parciales o completos, como mostramos en el siguiente ejemplo:

```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
lienzo.arc(100,100,50,0,Math.PI*2, false);
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```



Lo primero que seguramente notará en el método **arc()** en nuestro ejemplo es el uso del valor **PI**. Este método usa radianes en lugar de grados para los valores del ángulo. En radianes, el valor **PI** representa 180 grados, por lo que la fórmula **PI*2** multiplica **PI** por **2** obteniendo un ángulo de 360 grados.

El código genera un arco con centro en el punto **100,100** y un radio de 50 pixeles, comenzando a **0** grados y terminando a **Math.PI*2** grados, lo que representa un círculo completo. El uso de la propiedad **PI** del objeto **Math** nos permite obtener el valor preciso de PI.

Si necesitamos calcular el valor en radianes de cualquier ángulo en grados usamos la fórmula: **Math.PI / 180 × grados**, como en el próximo ejemplo:

```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
var radianes=Math.PI/180*45;
lienzo.arc(100,100,50,0,radianes, false);
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```



Con el código anterior obtenemos un arco que cubre 45 grados de un círculo. Intente cambiar el valor de la dirección a **true** (verdadero). En este caso, el arco será generado desde 0 grados a 315, creando un círculo abierto. Una cosa importante a considerar es que si continuamos construyendo el trazado luego del arco, el actual punto de comienzo será el final del arco. Si no deseamos que esto pase tendremos que usar el método **moveTo()** para cambiar la posición del lápiz, como hicimos anteriormente. Sin embargo, si la próxima figura es otro arco (por ejemplo, un círculo completo) siempre recuerde que el método **moveTo()** mueve el lápiz virtual hacia el punto en el cual el círculo comenzará a ser dibujado, no el centro del círculo. Digamos que el centro del círculo que queremos dibujar se encuentra en el punto **300,150** y su radio es de **50**. El método **moveTo()** debería mover el lápiz a la posición **350,150** para comenzar a dibujar el círculo.

ESTILOS DE LÍNEA

Hasta esta parte del capítulo hemos usado siempre los mismos estilos de líneas. El ancho, la terminación y otros aspectos de la línea pueden ser modificados para obtener exactamente el tipo de línea que necesitamos para nuestros dibujos.

Existen cuatro propiedades específicas para este propósito:

lineWidth Esta propiedad determina el grosor de la línea. Por defecto el valor es 1.0 unidades.

lineCap Esta propiedad determina la forma de la terminación de la línea. Puede recibir uno de estos tres valores: **butt**, **round** y **square**.

lineJoin Esta propiedad determina la forma de la conexión entre dos líneas. Los valores posibles son: **round**, **bevel** y **miter**.

miterLimit Trabajando en conjunto con **lineJoin**, esta propiedad determina cuánto la conexión de dos líneas será extendida cuando la propiedad **lineJoin** es declarada con el valor **miter**.

Las propiedades afectarán el trazado completo. Cada vez que tenemos que cambiar las características de las líneas debemos crear un nuevo trazado.

```
function iniciar(){
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.beginPath();
lienzo.arc(200,150,50,0,Math.PI*2, false);
lienzo.stroke();
lienzo.lineWidth=10;
lienzo.lineCap="round";
lienzo.beginPath();
lienzo.moveTo(230,150);
lienzo.arc(200,150,30,0,Math.PI, false);
lienzo.stroke();
lienzo.lineWidth=5;
lienzo.lineJoin="miter";
lienzo.beginPath();
lienzo.moveTo(195,135);
lienzo.lineTo(215,155);
lienzo.lineTo(195,155);
lienzo.stroke();
} window.addEventListener("load", iniciar, false);
```

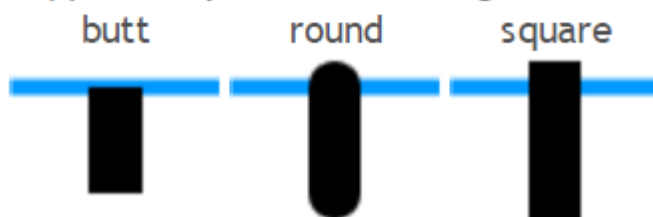


Line styles

Attributes

Name	Type	Default
lineWidth	<i>float</i>	1.0
lineCap	<i>string</i>	butt

Supports any of the following values:



lineJoin	<i>string</i>	miter
-----------------	---------------	-------

Supports any of the following values:



miterLimit	<i>float</i>	10
-------------------	--------------	----

Comenzamos el dibujo en el código creando un trazado para un círculo completo con propiedades por defecto. Luego, usando **lineWidth**, cambiamos el ancho de la línea a 10 y definimos la propiedad **lineCap** como **round**. Esto hará que el siguiente trazado sea más grueso y con terminaciones redondeadas. Para crear un trazado con estas características, primero movimos el lápiz a la posición **230,150** y luego generamos un semicírculo. Los extremos redondeados nos ayudarán a simular una boca sonriente.

Finalmente, agregamos un trazado creado con dos líneas para lograr una forma similar a una nariz. Las líneas para este trazado fueron configuradas con un ancho de **5** y serán unidas de acuerdo a la propiedad **lineJoin** y su valor **miter**. Esta propiedad hará a la nariz lucir puntiaguda, expandiendo las puntas de las líneas en la unión hasta que ambas alcancen un punto en común.

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:

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

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

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:

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 (solo 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.

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).

```
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);
} window.addEventListener("load", iniciar, false);
```

Mi mensaje

Como podemos ver en el código anterior, la propiedad **font** puede tomar varios valores a la vez, usando exactamente la misma sintaxis que CSS. La propiedad **textAlign** 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.

Además de los previamente mencionados, la API provee otro método importante para trabajar con texto:

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.textBaseline="bottom";
lienzo.fillText("Mi mensaje", 100,124);
var tamano=lienzo.measureText("Mi mensaje");
lienzo.strokeRect(100,100,tamano.width,24);
} window.addEventListener("load", iniciar, false);
```

Mi mensaje



En este ejemplo comenzamos con el mismo código anterior, pero agregamos un alineamiento vertical. La propiedad **textBaseline** fue establecida como **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.

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:

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

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

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

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);
} window.addEventListener("load", iniciar, false);
```

Mi mensaje

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

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.

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.

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

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** y **y**. Los valores pueden ser negativos, produciendo un efecto de espejo. Por defecto los valores son iguales a 1.0.

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.

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("PRUEBA2",0,0);
lienzo.rotate(-Math.PI/180*45);
lienzo.translate(0,100);
lienzo.scale(2,2);
lienzo.fillText("PRUEBA3",0,0);
} window.addEventListener("load", iniciar, false);
```

PRUEBA

PRUEBA2

PRUEBA3

No hay mejor forma de entender cómo funcionan las transformaciones que usarlas en nuestro código. En el código, 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 es dibujado 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);
```

```
} window.addEventListener("load", iniciar, false);
```

PRUEBA

PRUEA

Al igual que 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 fue estirado 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 mostrado por el código será igual de ancho que el anterior (es estirado horizontal y verticalmente). Para reinicializar la matriz y declarar nuevos valores de transformación, podemos usar el método **setTransform()**.

RESTAURANDO EL ESTADO

La acumulación de transformaciones hace realmente difícil volver a anteriores estados. En el código del 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.

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).

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);
} window.addEventListener("load", iniciar, false);
```

PRUEBA2

PRUEBA1

Si ejecuta el código en su navegador, verá el texto “PRUEBA1” en grandes letras al centro del lienzo, y el texto “PRUEBA2” en letras pequeñas, cercano al origen. Lo que hicimos fue grabar el estado por defecto del lienzo y luego establecer una nueva posición para el origen y estilos para el texto. El primer texto es dibujado con esta configuración, pero antes de dibujar el segundo texto el estado original es restaurado, por lo que este texto es mostrado con los estilos por defecto, no con los declarados para el primero.

No importa cuántas transformaciones hayamos realizado, luego de llamar al método **restore()** la configuración del lienzo será retornada exactamente a su estado anterior (el último grabado).

3.2 DRAG AND DROP

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

3.3 GEOLOCATION

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

3.4 STORAGE

Dos APIs fueron creadas con propósitos de almacenamiento de datos: **Web Storage** e **Indexed Database**. Básicamente, estas APIs transfieren la responsabilidad del almacenamiento de datos del servidor al ordenador del usuario, pero en el caso de Web Storage y su atributo `sessionStorage`, esta incorporación también incrementa el nivel de control y la eficiencia de las aplicaciones web.

Web Storage contiene dos importantes atributos que son a veces considerados APIs por sí mismos: `sessionStorage` y `localStorage`.

El atributo `sessionStorage` es responsable por mantener consistencia sobre la duración de la sesión de una página web y preservar información temporal como el contenido de un carro de compras, asegurando los datos en caso de accidente o mal uso (cuando la aplicación es abierta en una segunda ventana, por ejemplo).

Por el otro lado, el atributo `localStorage` nos permite grabar contenidos extensos de información en el ordenador del usuario. La información almacenada es persistente y no expira, excepto por razones de seguridad.

Ambos atributos, `sessionStorage` y `localStorage` reemplazan la anterior función del sistema de cookies y fueron creados para superar sus limitaciones.

La segunda API, agrupada dentro de las APIs de almacenamiento pero independiente del resto, es **Indexed Database**. La función elemental de un sistema de base de datos es la de almacenar información indexada.

Web Storage API trabaja sobre el almacenamiento de grandes o pequeñas cantidades de información, datos temporales o permanentes, pero no datos estructurados. Esta es una posibilidad solo disponible para sistemas de base de datos y la razón de la existencia de esta API.

3.5 FILE

Bajo el título de **File**, HTML5 ofrece varias APIs destinadas a operar con archivos. En este momento existen tres disponibles: File, File: Directories & System, y File: Writer.

Gracias a este grupo de APIs, ahora podemos crear y procesar archivos en el ordenador del usuario.

3.6 COMUNICACIÓN

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

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

El primer problema fue resuelto con el desarrollo de **XMLHttpRequest Level 2**. XMLHttpRequest fue la API usada por mucho tiempo para crear aplicaciones Ajax, códigos que acceden al servidor sin recargar la página web. El nivel 2 de esta API incorpora nuevos eventos, provee más funcionalidad (con eventos que permiten hacer un seguimiento del proceso), portabilidad (la API es ahora estándar), y accesibilidad (usando solicitudes cruzadas, desde un dominio a otro).

La solución para el segundo problema fue la creación de **Cross Document Messaging**. Esta API ayuda a los desarrolladores a superar las limitaciones existentes para comunicar diferentes cuadros y ventanas entre sí. Ahora una comunicación segura a través de diferentes aplicaciones es ofrecida por esta API utilizando mensajes.

La solución para el último de los problemas listados anteriormente es **Web Sockets**. Su propósito es proveer las herramientas necesarias para la creación de aplicaciones de red que trabajan en tiempo real (por ejemplo, salas de chat). La API les permite a las aplicaciones obtener y enviar información al servidor en períodos cortos de tiempo, volviendo posible las aplicaciones en tiempo real para la web.

3.7 WEB WORKERS

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

3.8 HISTORY

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

3.9 OFFLINE

Incluso hoy día, con acceso a Internet en cada lugar que vamos, quedar desconectado es aún posible.

Dispositivos portátiles se encuentran en todas partes, pero no la señal para establecer comunicación. Y los ordenadores de escritorio también pueden dejarnos desconectados en los momentos más críticos. Con la combinación de atributos HTML, eventos controlados por Javascript y archivos de texto, **Offline** permitirá a las aplicaciones trabajar en línea o desconectados, de acuerdo a la situación del usuario.

