Dibujar con el elemento canvas (Parte 2)

1. ESTILOS DE LÍNEA

En esta lección continuamos viendo características del elemento **canvas** de HTML5 y, en especial, de la API de su contexto **2d**.

Ya sabemos que podemos utilizar colores para el trazo de las líneas o para el relleno de las figuras cerradas.

Bien, también podemos cambiar el **estilo de línea**, es decir, el borde o contorno de las líneas y del resto de figuras que dibujamos en el canvas.

Para ello, se incluyen estas tres propiedades del contexto 2d:

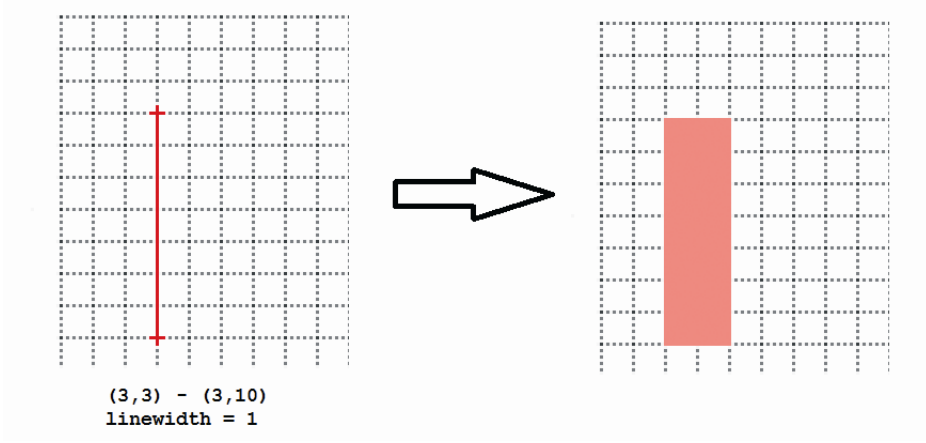
- **lineWidth**,
- **lineCap** y
- **lineJoin**

La propiedad **lineWidth** establece la anchura o grosor del trazo de las líneas, tomando un valor numérico entero o con decimales.

Recordando que la superficie del canvas está compuesta por un conjunto de píxeles, debemos entender que el grosor de las líneas se establece respecto del centro del trazado.

Esto puede ocasionar un problema a la hora de representar las líneas, ya que siempre se tienen que dibujar píxeles enteros.

Con esta figura lo entenderás mejor. El propósito es dibujar una línea que viene definida por los puntos inicial **(3, 3)** y final **(3, 10)**. El grosor será de **1 píxel**.

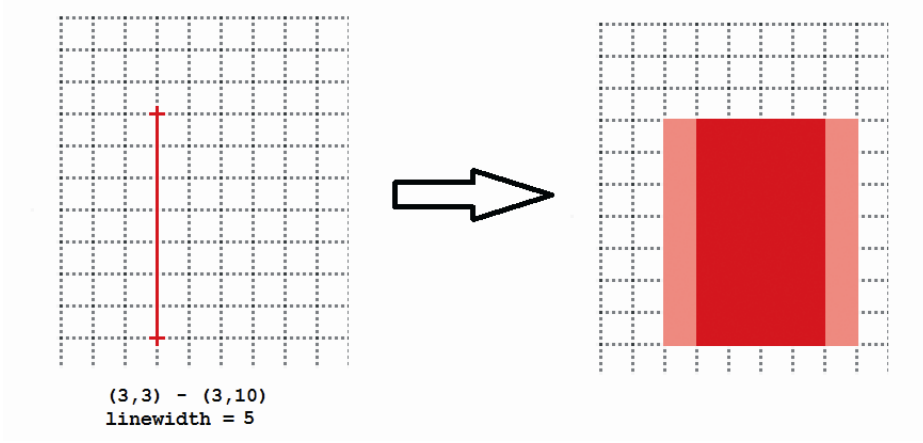


Si ampliamos la cuadrícula de píxeles de la superficie del canvas, tendremos que esa línea, con la anchura de un píxel, realmente se situaría en el centro del trazado.

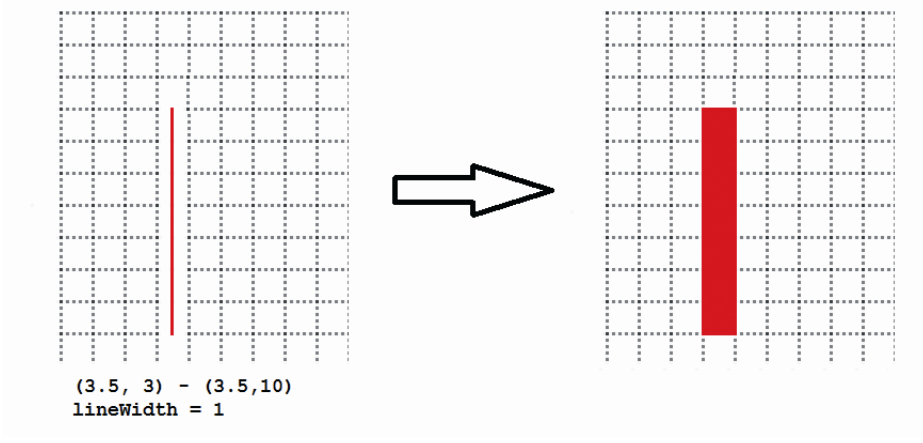
Como esto no se puede representar, ya que siempre se iluminarán píxeles completos, tenemos que lo que hace el navegador es una simulación dibujando una línea de 2 píxeles y difuminando el color de la misma (obsérvalo en la parte derecha de la figura anterior).

Por lo tanto, no conseguimos una línea bien definida.

Lo mismo podemos decir si **lineWidth** tiene el valor 5. En este caso, se dibujará una línea realmente ocupando 6 píxeles y los dos píxeles externos de un color distinto para difuminar la línea.

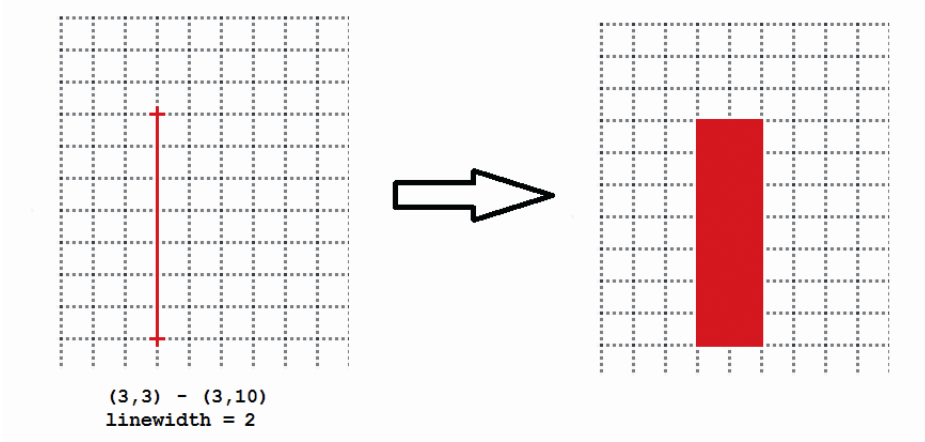


Podemos corregir este problema modificando las coordenadas del punto inicial y final de la línea. Así, si se establecen como **(3.5, 3)** y **(3.5, 10)**, entonces, como la línea va en el centro del trazado, se conseguirá una línea de un píxel y del color deseado.



Por lo tanto, si la anchura es un valor impar, tenemos que tener cuidado si deseamos conseguir líneas bien definidas.

Por su parte, si la anchura de la línea es de 2 píxeles u otro valor par, entonces sí que se puede dibujar utilizando píxeles completos, por lo que la línea siempre estará bien definida con el color real de la misma.

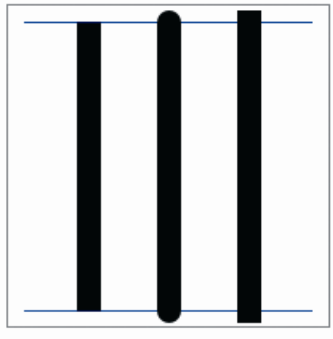


En el siguiente código se utiliza esta propiedad:

```
<script type="text/javascript">
window.onload = function() {
     var contexto =
 document.getElementById("lienzo1").getContext("2d");
     contexto.lineWidth = 10;
     contexto.strokeStyle = "red";
     contexto.beginPath();
     contexto.moveTo(30.5, 10);
     contexto.lineTo(30.5,189);
     contexto.stroke();
}
</script>
```

Recuerda que lo estamos viendo para una línea, pero sería igualmente válido para el borde o contorno de otras figuras, como los rectángulos o arcos.

Por su parte, la propiedad **lineCap** determina el aspecto de los puntos finales de las líneas. Aquí lo puedes ver. El ejemplo dibuja tres líneas paralelas pero que empiezan y terminan en los mismos puntos.



```
<script type="text/javascript">
window.onload = function() {
     var contexto =
 document.getElementById("lienzo1").getContext("2d");

     // Líneas horizontales
     contexto.strokeStyle = "blue";
     contexto.lineWidth = 1;
     contexto.beginPath();
     contexto.moveTo(10, 10.5);
     contexto.lineTo(190, 10.5);
     contexto.moveTo(10, 190.5);
     contexto.lineTo(190, 190.5);
     contexto.stroke();

     //Líneas verticales
     contexto.strokeStyle = "black";
     contexto.lineWidth = 15;

     contexto.lineCap = "butt";
     contexto.beginPath();
     contexto.moveTo(50.5, 10.5);
     contexto.lineTo(50.5, 190.5);
     contexto.stroke();

     contexto.lineCap = "round";
     contexto.beginPath();
     contexto.moveTo(100.5, 10.5);
     contexto.lineTo(100.5, 190.5);
     contexto.stroke();

     contexto.lineCap = "square";
     contexto.beginPath();
     contexto.moveTo(150.5, 10.5);
     contexto.lineTo(150.5, 190.5);
     contexto.stroke();


}
</script>
```

Sin embargo, utilizando la propiedad **lineCap** vemos que los puntos finales no son iguales. De hecho, en la segunda y tercera línea esos bordes finales van más allá de las coordenadas iniciales y finales (que quedan representadas aquí por las líneas azules).

Estos son los tres valores que puede tomar la propiedad **lineCap**:

- **butt**: en este caso la línea comienza y termina siguiendo exactamente el trazado establecido. Es el valor predeterminado.

Este es el valor aplicado en la línea de la izquierda.

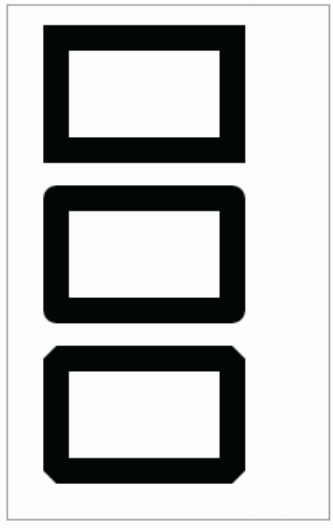
- **round**: hace que los extremos de la línea estén redondeados. Este efecto se nota mucho más si la línea es gruesa, ya que utiliza como diámetro del semicírculo la anchura de la línea.

Este es el valor aplicado en la línea central.

- **square**: similar al valor butt, pero la línea sobrepasa los extremos del trazado (exactamente en un 50% de la anchura de la línea).

Este es el valor aplicado en la línea de la derecha.

Finalmente, la propiedad **lineJoin** define la forma con la que se unirán las líneas pertenecientes a un trazado o rectángulo.



En este caso se dibujan tres rectángulos del mismo tamaño y con el mismo grosor de línea, pero aplicando distintos valores para la propiedad **lineJoin**.

Estos son los valores que puede tomar esta propiedad:

- **miter**: las uniones forman esquinas perfectas. Es el valor predeterminado y el que se ha utilizado en el rectángulo superior.

- **round**: las uniones forman esquinas redondeadas. Es el valor utilizado en el rectángulo central.

- **bevel**: las uniones no aparecen redondeadas sino formando una especie de biselado. Es el valor utilizado en el rectángulo inferior.

Todas estas propiedades pueden utilizarse tanto con trazados compuestos de distintas operaciones de dibujo o con rectángulos individuales.

2. GRADIENTES

Otra posibilidad que nos proporciona el contexto 2d de canvas es crear **gradientes** y aplicarlos a la hora de dibujar el trazado de los objetos o su relleno.

Los gradientes se utilizan habitualmente en el diseño de las páginas web, sobre todo como fondo de las páginas o de alguno de sus elementos.

Es corriente utilizar alguna aplicación de diseño, como Photoshop, para crear el gradiente y después utilizarlo como una imagen más en la página web.

Sin embargo, utilizando canvas (o CSS3) esto ya no es necesario.

Un gradiente no es más que una forma de conseguir un degradado o transición suave entre dos o más colores.

Para utilizar un gradiente a la hora de trazar o de rellenar una figura, primero debes crearlo y después establecerlo como el valor de la propiedad **strokeStyle** o **fillStyle**.

En este ejemplo puedes verlo. Así pues, en la línea resaltada se crea un gradiente lineal con el método **createLinearGradient**.

```
<script type="text/javascript">
window.onload = function() {
  var lienzo = document.getElementById('lienzo1');
  var contexto = lienzo.getContext('2d');

  var gradiente = contexto.createLinearGradient(0, 0,
                  lienzo.width, lienzo.height);
  gradiente.addColorStop(0, "#ffffff");
  gradiente.addColorStop(1, "#000000");
  contexto.fillStyle = gradiente;
  contexto.fillRect(0, 0, lienzo.width, lienzo.height);
}
</script>
```

Este método requiere de cuatro parámetros:

- Los dos primeros parámetros representan las coordenadas **x** e **y** del primer punto que define el gradiente.
- Los dos segundos parámetros representan las coordenadas **x** e **y** del segundo punto que define el gradiente.

En este caso se ha utilizado como punto inicial el origen del elemento canvas **(0, 0)** y como punto final su esquina inferior derecha **(lienzo.width, lienzo.height)**.

Fíjate que se han utilizado las propiedades **width** y **height** del objeto **canvas** para obtener su anchura y altura.

De esta forma, el gradiente crea una transición de colores desde una esquina hasta la otra, consiguiendo un efecto como de degradado en diagonal.

Adicionalmente, debemos establecer los colores del gradiente. Esto se consigue mediante el método **addColorStop**.

El método **addColorStop** requiere de dos parámetros:

- El primer parámetro es un valor numérico que va desde **0.0** hasta **1.0** y representa la posición en que se aplica el color.
- El segundo parámetro es el propio color.

Por lo tanto, en esta línea estamos indicando con el valor **0** del primer parámetro que el color **#ffffff** se aplica al principio del gradiente:

```
gradiente.addColorStop(0, "#ffffff");
```

Mientras que en esta otra línea indicamos con el valor **1** del primer parámetro que el color **#000000** se aplica al final del gradiente:

```
gradiente.addColorStop(1, "#000000");
```

Es decir, que el gradiente se definirá como una transición entre el color blanco (al principio) y negro (al final).

Podemos incorporar más colores al gradiente, simplemente indicando la posición y el color.

Por ejemplo, con el valor **0.5** para el primer parámetro estableceríamos un cambio de color hacia la mitad del gradiente.

Finalmente, en las dos últimas líneas del *script* se utiliza el gradiente para establecer el valor de la propiedad **fillStyle** y se dibuja un rectángulo relleno, obteniendo el resultado que has visto anteriormente.

Este tipo de gradiente se llama **lineal** porque lo define la línea imaginaria entre el punto inicial y final.

Otra posibilidad es definir el gradiente mediante dos círculos en lugar de dos puntos. Para ello, se utiliza el método **createRadialGradient**:

```
<script type="text/javascript">
window.onload = function() {
  var lienzo = document.getElementById('lienzo1');
  var contexto = lienzo.getContext('2d');

  var gradiente =
     contexto.createRadialGradient(lienzo.width/2,
     lienzo.height/2, 0, lienzo.width/2,
     lienzo.height/2, 150);
  gradiente.addColorStop(0, "#ffffff");
  gradiente.addColorStop(1, "#000000");
  contexto.fillStyle = gradiente;
  contexto.fillRect(0, 0, lienzo.width, lienzo.height);
}
</script>
```

En este caso, los tres primeros parámetros definen el primer círculo y los tres segundos parámetros, el segundo círculo.

Para definir un círculo se indica la coordenada **x** e **y** de su centro y la **longitud** del radio.

Por lo tanto, el primer círculo tiene como centro el punto central del canvas y un radio de **0** píxeles; mientras que el segundo círculo tiene el mismo punto para el centro y un radio de **150** píxeles.

Observa el resultado obtenido.

Fíjate que los bordes del canvas no presentan el degradado, ya que el radio del círculo externo es de **150** píxeles. La transición va del blanco al negro pero ahora de forma radial.

Veremos que también es posible crear gradientes de forma muy parecida con las hojas de estilo en cascada **CSS3**.

3. PATRONES

Para rellenar figuras dibujadas en el canvas, también podemos utilizar un **patrón** en lugar de colores o gradientes.

Normalmente dicho patrón es una imagen que se repite formando el fondo deseado, de forma muy parecida a lo que podemos conseguir con la propiedad **background-image** de CSS.

Aquí lo puedes ver. El resultado dibujado en la superficie del canvas no es una única imagen sino la repetición de una más pequeña.

Veamos cómo se ha conseguido.

```
<script type="text/javascript">
window.onload = function() {
   var lienzo = document.getElementById("lienzo1");
   var contexto = lienzo.getContext("2d");
   var imagen = document.createElement("img");
   imagen.src = "images/stripes.jpg";
   imagen.onload = function() {
   var pattern = contexto.createPattern(this, "repeat");
   contexto.fillStyle = pattern;
   contexto.fillRect(0, 0, lienzo.width, lienzo.height);
   }
}
</script>
```

Primero creamos un elemento **img** con el método del DOM **createElement**. Es importante darse cuenta que este elemento solo existe en la memoria del navegador, ya que no lo enlazamos con ningún elemento del árbol del documento.

Después establecemos la propiedad **src** del nuevo elemento **img** para indicar el archivo gráfico que realmente se utiliza.

Seguidamente se añade el manejador del evento **load** para la imagen. Esto es necesario, ya que tenemos que esperar a que la imagen se haya cargado completamente para poder manipularla.

El código incluye la creación del patrón mediante el uso del método **createPattern** y su posterior empleo para dibujar un rectángulo relleno.

Fíjate cómo se crea el patrón. El comando **createPattern** requiere de dos parámetros:

- En el primero se indica la imagen a utilizar, por lo que hemos escrito **this** ya que es la misma imagen y
- en el segundo parámetro se indica cómo queremos que se repita.

Este segundo parámetro admite los mismos valores que la propiedad **background-repeat** de las hojas de estilo en cascada. Es decir: **no-repeat**, **repeat**, **repeat-x** y **repeat-y**.

El patrón que hemos empleado en este caso es una **imagen**. Sin embargo, la especificación del HTML5 indica que también podríamos utilizar otro elemento **canvas** (con su correspondiente contenido) e incluso un **vídeo**, aunque esto último todavía no está implementado en ningún navegador.

4. DIBUJAR IMÁGENES

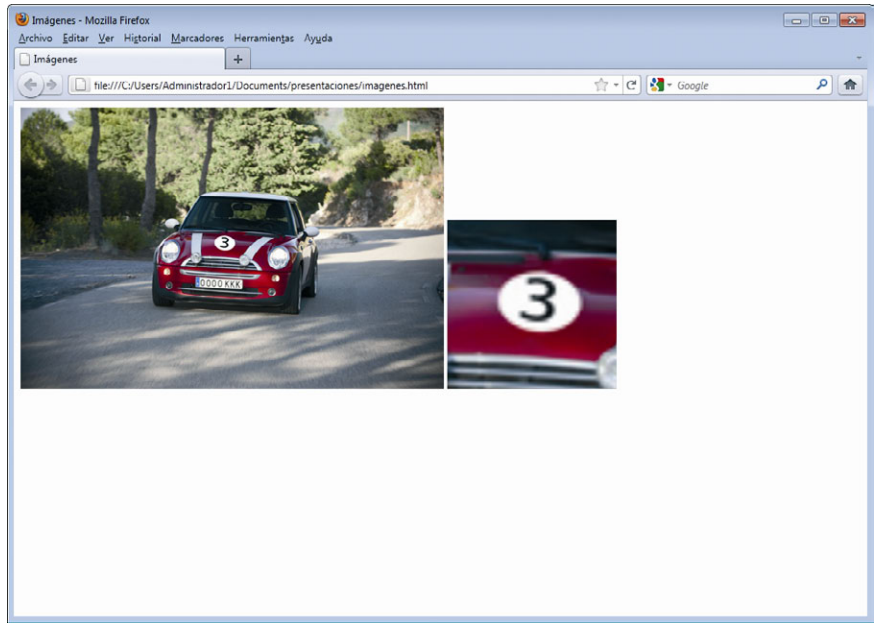
La superficie de dibujo del elemento canvas admite la visualización de **imágenes** a través del método **drawImage** del contexto 2d.

Este método tiene la misma entidad que cualquier otro método de dibujo que hemos visto, pero en lugar de dibujar líneas o figuras, dibuja una imagen o archivo gráfico.

Normalmente esto no tendrá demasiado sentido si no se busca conseguir algún efecto especial, ya que la etiqueta **img** del lenguaje HTML es más adecuada si lo único que deseamos es incluir una imagen en la página web.

La página de la figura siguiente muestra una imagen a través de la etiqueta estándar **img**. Al lado de esta se sitúa un elemento canvas de HTML5 que por ahora no es visible al tener el fondo transparente.

El código JavaScript de la página hace que cuando el usuario sitúe el puntero del ratón por encima de la imagen original, la parte donde está situado dicho puntero se muestre en el elemento canvas, pero de forma ampliada.



Lo que se ha hecho es dibujar una parte de la imagen completa en el elemento canvas, realizando, a la vez, un zoom de dicha zona de la imagen.

```
<!DOCTYPE HTML>
<html>
<head>
<meta    http-equiv="Content-Type"    content="text/html;
charset=utf-8">
<title>Imágenes</title>
</head>

<body>
<img    id="foto"    src="images/coche.jpg"    width="500"
height="333">
<canvas id="lienzo1" width="200" height="200"></canvas>
<script type="text/javascript">
window.onload = function() {
     var lienzo = document.getElementById("lienzo1");
     var contexto = lienzo.getContext("2d");
     var imagen = document.getElementById("foto");

     function zoom(e)
     {
          var sx = e.clientX - this.offsetLeft;
          var sy = e.clientY - this.offsetTop;
          contexto.drawImage(this, sx - 25, sy - 25,
                              50, 50, 0, 0, 200, 200);
     }
     imagen.onmouseover = zoom;
     imagen.onmousemove = zoom;
}
</script>
</body>
</html>
```

El método **drawImage** admite tres versiones distintas, que se distinguen por el número de parámetros que utilizan:

- **drawImage(imagen, dx, dy)**: el primer parámetro es el objeto **imagen** que se dibujará en el canvas, mientras que los otros dos parámetros se corresponden con las coordenadas **x** e **y** del punto donde se mostrará en el canvas. Esas coordenadas representan la esquina superior izquierda del dibujo.

- **drawImage(imagen, dx, dy, dw, dh)**: es igual que la versión anterior, pero ahora los parámetros adicionales **dw** y **dh** representan las dimensiones con las que queremos que se muestre la imagen.

  Si esas dimensiones son distintas a las de la imagen original, entonces el navegador realizará un stretch o zoom.

- **drawImage(imagen, sx, sy, sw, sh, dx, dy, dw, dh)**: en este caso los cuatro primeros parámetros establecen el área de la imagen que queremos mostrar en el canvas (**sx** y **sy** representan las coordenadas iniciales, mientras que **sw** y **sh** las dimensiones de dicha área).

  Los cuatro últimos parámetros son los mismos que la versión anterior del método, es decir, que definen el punto inicial en la superficie del canvas y las dimensiones con las que se muestran.

  Esta última versión es la que hemos utilizado en el código anterior

Pero primero expliquemos cómo conseguimos la imagen que se dibujará en la superficie del canvas.

Como puedes comprobar se ha utilizado la etiqueta estándar **img** para añadir una imagen al contenido de la página web. Esta imagen tiene el identificador "**foto**", que se utiliza como parámetro del método **document.getElementById** para seleccionar el correspondiente objeto en el árbol del documento.

Se utilizan dos manejadores de evento. Se trata de los eventos de ratón **onmouseover** y **onmousemove**.

El evento **mouseover** ocurre cuando tenemos el puntero del ratón fuera de la imagen y entra en su superficie.

Por su parte, el evento **mousemove** ocurre cuando el puntero del ratón se mueve por encima de la superficie de la imagen.

En ambos casos se ejecuta el código de la función **zoom**.

```
function zoom(e)
{
     var sx = e.clientX - this.offsetLeft;
     var sy = e.clientY - this.offsetTop;
     contexto.drawImage(this, sx - 25, sy - 25,
                              50, 50, 0, 0, 200, 200);
}
```

La función **zoom** empieza con dos líneas para calcular la posición de la imagen en la ventana del navegador. Para ello, se han utilizado las propiedades **clientX**, **clientY**, **offsetLeft** y **offsetTop**.

Como puedes ver, la función **zoom** recibe un parámetro que se corresponde con un objeto que representa el evento que ha ocurrido. Este objeto dispone de estas cuatro propiedades:

- **clientX** y **clientY** representan las coordenadas del punto donde está situado el puntero del ratón, pero correspondientes a la zona de renderización de la ventana del navegador.

- **offsetLeft** y **offsetTop**, por su parte, permiten calcular dónde empieza la imagen en esa superficie.

De esta forma, la expresión **e.clientX - e.offsetLeft** proporciona la coordenada **x** donde empieza la imagen; mientras que **e.clientY - e.offsetTop**, la coordenada **y** correspondiente.

Sabiendo estas coordenadas, ya podemos utilizar el método **drawImage** para dibujar la zona de la imagen en el **canvas**:

```
contexto.drawImage(this, sx - 25, sy - 25,
                         50, 50, 0, 0, 200, 200);
```

- **this**: representa el objeto imagen sobre el que se está ejecutando el manejador del evento, es decir, la imagen de identificador "foto".

- **sx-25** y **sy-25**: representan las coordenadas iniciales de la zona de la imagen que queremos mostrar en el canvas. Se restan 25 píxeles para que el puntero del ratón realmente represente el punto central de la zona que se visualiza en el canvas.

- **50, 50**: son las dimensiones del área de la imagen que queremos capturar y dibujar en el canvas. Por eso anteriormente se ha restado 25 píxeles para establecer el punto medio de esta selección.

- **0, 0**: indica las coordenadas del punto inicial donde se dibuja en el canvas.

- **200, 200**: representa las dimensiones con las que se mostrará.

Fíjate que se está seleccionando un área de **50x50** píxeles y se muestra como **200x200** píxeles. De ahí que el canvas muestre el área seleccionada con un zoom de ampliación.

Aunque este efecto se podría haber conseguido de distintas maneras, la posibilidad de utilizar un elemento **canvas** hace que el navegador adquiera mayor protagonismo al no requerir de ningún otro complemento o *plugin*.

5. DIBUJAR TEXTO

Finalizaremos la lección comprobando que igual que podemos dibujar imágenes en la superficie del canvas, podemos dibujar **texto**.

Sin embargo, es importante entender que esto no es tampoco adecuado si queremos que dicho texto forme una parte importante del contenido de la página web. Todo lo que se visualice en el canvas se maneja siempre como un dibujo.

Esto quiere decir que realmente en el árbol del documento lo único que quedará representado es el propio canvas, por lo que su contenido no estará a nuestra disposición a través de JavaScript ni lo estará para otros dispositivos, como los lectores de páginas web para personas con alguna discapacidad. El elemento canvas es una superficie de dibujo y como tal tiene que ser entendido.

```
<script type="text/javascript">
window.onload = function() {
     var lienzo = document.getElementById("lienzo1");
     var contexto = lienzo.getContext("2d");
     var gradiente = contexto.createLinearGradient(0, 0,
                                        0, lienzo.height);
     gradiente.addColorStop(0, "#f0b7a1");
     gradiente.addColorStop(0.5, "#8c3310");
     gradiente.addColorStop(1, "#bf6e4e");
     contexto.strokeStyle = gradiente;
     contexto.font ="bold 35px Verdana, sans-serif";
     contexto.strokeText("Texto de prueba", 10, 50);
}
</script>
```

Para dibujar texto en la superficie del canvas puedes utilizar el método **strokeText** si solo quieres dibujar el contorno; o el método **fillText** para dibujar el texto tal como lo harías habitualmente.

Ambos métodos requieren la cadena de texto y las coordenadas donde se empezará el dibujo en el canvas.

Opcionalmente se puede utilizar un cuarto parámetro que representa la anchura máxima que puede ocupar el texto.

Estos dos métodos utilizan el valor de la propiedad **strokeStyle** y **fillStyle**, respectivamente.

En el ejemplo anterior se ha utilizado un gradiente lineal de tres colores como valor de la propiedad **strokeStyle**.

Además, es posible utilizar las propiedades **font**, **textAlign** y **textBaseLine** para modificar el texto.

La propiedad **font** utiliza la misma sintaxis que su propiedad equivalente CSS. Por ejemplo, en la línea **19** establecemos un tipo de fuente **Verdana** o **sans-serif** de tamaño **35** píxeles y en **negrita**.

Esta es la sintaxis que puedes utilizar para establecer la propiedad **font**:

```
contexto.font = "font-style font-variant font-weight font-size line-height font-family";
```

No tienes que incluir todos estas subpropiedades, pero sí respetar el orden en que aparecen.

Finalmente, se utiliza el método **strokeText** para dibujar el contorno del texto.

Si en lugar de la propiedad **strokeStyle**, utilizamos **fillStyle**; y en lugar del método **strokeText**, utilizamos **fillText** obtendremos este otro resultado.

Recuerda que siempre estarás dibujando, por lo que no utilices estos métodos si realmente lo que quieres hacer es introducir texto o imágenes que forman una parte significativa del contenido de la página web.