Desarrollo Web en entorno Cliente: JAVASCRIPT

UNIDAD 10. HTML DINÁMICO

UNIDAD 10. HTML DINÁMICO

Si uno conoce HTML y CSS puede tener un control completo sobre los elementos que aparecen en la página y el aspecto que tienen, es decir hemos generado la parte estática de la página web.

El objetivo del HTML dinámico (DHTML) es dar un control programático sobre todos los elementos de un documento de una página web, tanto antes como después de que se cargue la página. Y por todos los elementos entendemos: el texto, los estilos, las etiquetas e, incluso, los atributos de las etiquetas. En un mundo DHTML ideal, todos estos elementos pueden verse como objetos o propiedades de objetos. Y como hemos visto, una vez que algo puede ser referenciado como un objeto o una propiedad, puede manipularse usando Javascript.

Esto, como se dijo, es lo ideal. En el mundo real DHTML es un lío de estándares confusos, soporte impropio o deficiente de esos estándares y extensiones propietarias creadas por algunas compañías.

DHTML es la combinación de HTML, CSS y JS.

El modelo de objetos de documento.- DOM

"El modelo de objetos de W3C DOM (Document) es una plataforma e interfaz de lenguaje neutro que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento".

"El DOM HTML es un estándar de cómo obtener, cambiar, añadir o eliminar elementos HTML".

El estándar W3C DOM se divide en 3 partes:

- Núcleo DOM (DOM Core)- modelo estándar para todos los tipos de documentos
- DOM XML - modelo estándar para los documentos XML
- HTML DOM - modelo estándar para los documentos HTML

El DOM se divide en diferentes niveles: DOM nivel 1, DOM nivel 2 y DOM nivel 3.

Niveles del DOM

A diferencia de otras recomendaciones, el DOM no se recoge en un único documento, sino que consiste en **tres niveles** —a día de hoy—, **y cada uno de ellos está a su vez compuesto por varias recomendaciones referidas a distintos aspectos de la interfaz**:

- **Modelo de objeto de documento de nivel 1 DOM 1**:
  - DOM Core : Define el conjunto mínimo de objetos e interfaces con los que manipular la estructura de un documento, ya se trate de un documento HTML o XML, o cualquier lenguaje basado en éste.
  - DOM HTML : Define una serie objetos y métodos específicos de un documento HTML, y que por extensión también pueden aplicarse a un XHTML. Es importante conocerlo bien porque en muchas ocasiones hay que emplearlo en un script como alternativa para Internet Explorer de métodos del DOM Core 1, como por ejemplo *getAttribute* y *setAttribute*.

Sobre la inconsistencia del soporte de Explorer para estos dos métodos, ver «Una nota sobre el soporte de getAttribute y setAttribute en Internet Explorer 6»

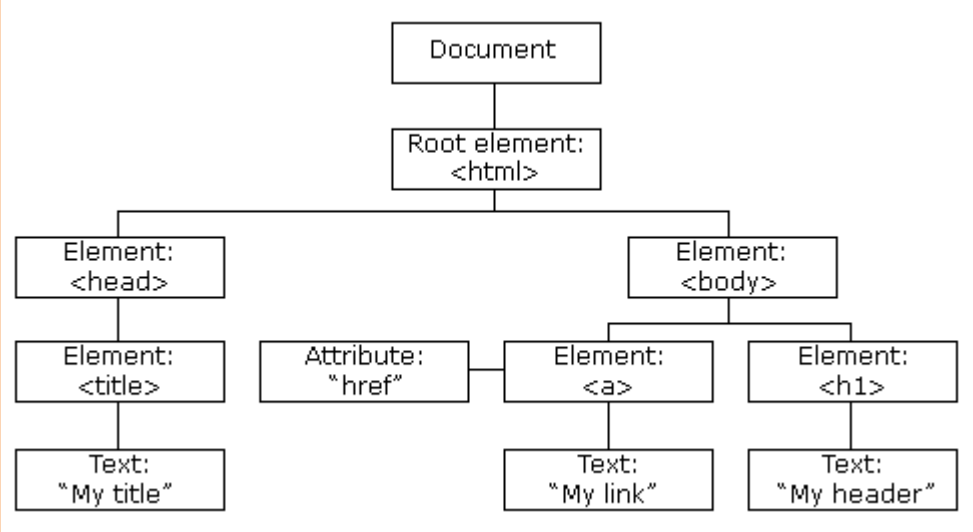
- **Modelo de objeto de documento de nivel 2 DOM 2**:
  - DOM Core : Extiende las interfaces definidas en el DOM Core 1, por ejemplo añadiendo soporte para espacios de nombre, y permitiendo así poder manipular secciones del documento asociadas a uno de ellos.
  - DOM Views : Define lo que es una «vista» de un documento, es decir, el documento cuando es representado en un agente de usuario —por ejemplo cuando se le aplica una hoja de estilo en un navegador—, en contraposición a la «vista abstracta» del documento en sí. Sí, es casi metafísica.
  - DOM Events : Especifica qué es un evento, sus tipos, y qué son los procesos de *burbuja*, *captura* y *cancelación*. Además, entre otras muchas cosas, define un método maravilloso: `addEventListener` .
  - DOM Style : Define una interfaz para poder extraer información y manipular la/s hojas/s de estilo de un documento.
  - DOM Traversal and Range : Define la interfaz avanzada y los métodos con los que desplazarse por el árbol del documento, y para seleccionar partes del mismo.
  - DOM HTML : Amplía el DOM HTML 1, e indica explícitamente que vuelve el anterior obsoleto.

- **Modelo de objeto de documento de nivel 3 DOM 3**:
  - DOM Core : Amplía aún más el contenido de DOM Core 2.
  - DOM Load and Save : Define una interfaz con la que cargar documentos XML que puedan integrarse en el árbol de otro documento.
  - DOM Validation : Define una interfaz que permite asegurar que la estructura de un documento modificado dinámicamente sigue siendo válida.

El w3c está trabajando en el DOM 4 http://www.w3.org/TR/2013/WD-dom-20131107/

En la actualidad el DOM 4 se encuentra como indica el siguiente enlace: (18-junio-2015)
https://www.w3.org/TR/2015/WD-dom-20150618/
Ver también: https://www.w3.org/DOM/DOMTR

Cuando se carga una página web, el navegador crea un Document Object Model, el DOM de la página.
El código modelo HTML DOM está construido como un árbol de objetos:



10.1 EL OBJETO STYLE

El objeto style se utiliza para modificar los atributos CSS de cualquier elemento HTML.

Lista de atributos CSS

Vamos a estudiar los siguientes apartados:

- Propiedades de texto y fuentes
- Propiedades de color y fondos
- Propiedades de composición relativas a los márgenes
- Propiedades de composición relativas a los rellenos
- Propiedades de composición relativas a los bordes
- Propiedades de clasificación
- Propiedades de posicionamiento
- Propiedades de impresión
- Propiedades de filtrado
- Otras propiedades

PROPIEDADES DE TEXTO Y FUENTES		
Atributo CSS	**Javascript**	**Significado**
font-family	fontFamily	Establece el tipo de fuente. Se permiten varios tipos de fuente en cada definición, de manera que en caso de no poderse aplicar uno, se intenta con el siguiente, y así sucesivamente hasta el final de la lista.
font-style	fontStyle	Establece el estilo (**normal**,**italic**, **oblique**) de las fuentes.
font-variant	fontVariant	Establece la **variación** de la fuente (**normal** y mayúsculas de pequeño tamaño **small-caps**).
font-weight	fontWeight	Establece el nivel de grosor de las letras del texto. A veces, los cambios numéricos de grososr no se corresponden con un cambio real, puesto que el sistema escoge el valor posible más cercano al indicado, que puede coincidir con el actual. Los valores admitidos son (**normal**, **bold**, **bolder**, **lighter**, **100**, **200**, **300**, **400**, **500**, **600**, **700**, **800**,). Como referencia el valor **400** coincide con b>normal y **700** con **bold**.
font-size	fontSize	Especifica el tamaño de la fuente.
@font-face		Especifica una fuente que se puede empotrar en el documento. Esto permite a un usuario utilizar fuentes inexistentes en su sistema.
font	font	Permite definir los atributos explicados anteriormente: (**font-style**, **font-variant**, **font-weight**, **font-size** y **font-family**). Los elementos **font-size** y **font-family** son obligatorios.
letter-spacing	letterSpacing	Especifica espacios adicionales entre letras. Añade una distancia entre letras a la previamente existente.
line-height	lineHeight	Especifica la distancia entre las líneas de un párrafo.
text-decoration	textDecoration	Describe tipos de decoración que se le puede aplicar al texto. (**none**, **underline**, **overline**, **line-through**, **blink**)
text-transform	textTransform	Modifica la visualización de un texto. (**capitalize**, **uppercase**, **lowercase**, **none**)

text-align	textAlign	Indica la alineación del texto. (**left**, **right**, **center**, **justify**))
text-indent	textIndent	Especifica la indentación de la primera línea
vertical-align	verticalAlign	Afecta al alineamiento vertical del objeto. **sub** alinea inferiormente y **super** superiormente.

PROPIEDADES DE COLOR Y FONDO		
Atributo CSS	**Javascript**	**Significado**
color	color	Determina el color del texto de un elemento.
background-color	backgroundColor	Determina el color de fondo de un elemento. (**color**, **transparent**).
background-image	backgroundImage	Determina la imagen de fondo de un elemento.
background-repeat	backgroundRepeat	Determina la forma en la que una imagen de fondo se repite en su contenedor (**repeat**, **repeat-x**, **repeat-y**, **no-repeat**).
background-attachment	backgroundAttachment	Determina si la imagen de fondo de un elemento se desplaza o se queda fija(**scroll**, **fixed**).
background-position	backgroundPosition	Determina la posición dónde se coloca la imagen de fondo. (**top**, **left**, **center**, **right**, **bottom**).
background	background	Permite definir todas las propiedades de fondo en un solo atributo

PROPIEDADES DE COMPOSICIÓN RELATIVAS A LOS MÁRGENES		
Atributo CSS	**Javascript**	**Significado**
margin-top	marginTop	Determina el margen superior del elemento
margin-bottom	marginBottom	Determina el margen inferior del elemento
margin-left	marginLeft	Determina el margen izquierdo del elemento
margin-right	marginRight	Determina el margen derecho del elemento
margin	margin	Determina la posición de los márgenes superior, inferior, derecho e izquierdo. Si se incluye un solo valor, los cuatro márgenes se adaptan al valor. Cuando se especifican dos valores, el primero representa los márgenes superior e inferior y el segundo los márgenes izquierdo y derecho. Con tres valores, el primero indica el margen superior, el segundo los márgenes izquierdo y derecho y el tercero el margen inferior. Cuando se incluyen cuatro el orden es: superior, derecho, inferior, izquierdo.

PROPIEDADES DE COMPOSICIÓN RELATIVAS A LOS RELLENOS		
Atributo CSS	**Javascript**	**Significado**
padding-left, padding-right, padding-top, padding-bottom	paddingLeft, paddingRight, paddingTop, paddingBottom	Determina el relleno en la parte izquierda, derecha, superior e inferior del elemento.
padding	padding	Determina el relleno. Su empleo es igual que el descrito en **margin**

PROPIEDADES DE COMPOSICIÓN RELATIVAS A LOS BORDES		
Atributo CSS	**Javascript**	**Significado**
border-xxx-color	borderTopColor, borderBottomColor, borderLeftColor, borderRightColor	Determina el color del borde de un elemento, **xxx** representa a las palabras top, bottom, left o right, por lo que existen las cuatro propiedades: border-top-color, border-bottom-color, border-left-color, y border-right-color.
border-xxx-style	borderTopStyle, borderBottomStyle, borderLeftStyle, borderRightStyle	Determina el estilo del borde de un elemento, **xxx** representa a las palabras top, bottom, left o right, por lo que existen las cuatro propiedades: border-top-style, border-bottom-style, border-left-style, y border-right-style. Puede tomar los siguientes valores: **none**, **dotted**, **dashed**, **solid**, **double**, **groove**, **ridge**, **inset**, **outset**).
border-xxx-width	borderTopWidth, borderBottomWidth, borderLeftWidth, borderRightWidth	Determina las características del borde de un elemento, **xxx** representa a las palabras top, bottom, left o right, por lo que existen las cuatro propiedades: border-top-width, border-bottom-width, border-left-width, y border-right-width. Puede tomar los siguientes valores: **thin**, **medium**, **thick** además de la longitud).
border-xxx	borderTop, borderBottom, borderLeft, borderRight	Determina la anchura del borde de un elemento, **xxx** representa a las palabras top, bottom, left o right, por lo que existen las cuatro propiedades: border-top, border-bottom, border-left y border-right.
border	border	Determina las características del borde de un elemento. Las características especificadas se aplican de igual manera a los bordes superior, inferior, izquierdo y derecho del elemento.
float	styleFloat	Determina la posición flotante en la que el elemento se sitúa respecto a otros elementos. (**left**, **right**, **none**).

PROPIEDADES DE CLASIFICACIÓN		
Atributo CSS	**Javascript**	**Significado**
display	display	Determina si el elemento se verá o no. A diferencia del atributo **visibility**, con **display** no se reserva espacio para el elemento.
list-style-type	listStyleType	Determina la forma del marcador que se coloca a la izquierda de cada elemento de una lista. Puede valer:(**disc**, **circle**, **square**, **decimal**, **lower-roman**, **upper-roman**, **lower-alpha**, **upper-alpha**, **none**).
list-style-image	listStyleImage	determina la imagen que servirá como marcador que se coloca a la izquierda de cada elemento de una lista.
list-style-position	listStylePosition	Determina la forma en la que se visualizará el marcador de cada elemento de una lista en relación con su contenido. (**inside**, **outside**).
list-style	listStyle	Notación corta para determinar el estilo, posición e imagen de los marcadores de las listas.

PROPIEDADES DE POSICIONAMIENTO		
Atributo CSS	**Javascript**	**Significado**
clip	clip	Crea un área de recorte de un elemento posicionado (**shape**, **auto**, donde **shape: rect(top, right, bottom, left)).**
height	height	Determina la altura del elemento.
width	width	Determina la anchura del elemento.
left, top	left, top	Determinan la posición del elemento relativa al borde izquierdo **left** y al borde superior del documento **top**. Se deben emplear con el atributo **position**.
position	position	Determina la forma en la que el elemento se va a posicionar en el documento respecto a los demás objetos. Empleando el valor **static** se siguen las reglas habituales de colocación de elementos en HTML. Al usar **absolute** se consigue colocar el objeto en una posición absoluta independientemente de donde se encuentran los demás elementos. **relative** permite situar el objeto en una posición relatva respecto a la posición inicial que le correspondería al elemento en la página HTML. (**absolute**, **relative**, **static**).
visibility	visibility	Determina si el elemento es visible. A diferencia de **display:none**, el espacio del objeto se respeta cuando no está visible. (**hidden**, **inherit**).
overflow	overflow	Indica si el contenido existente en un contenedor se corta cuando no cabe **hidden** o bien se puede acceder al resto del contenido mediante barras de desplazamiento **scroll**. (**scroll**, **hidden**, **visible**, **auto**).
z-index	zIndex	Determina la posición en el eje **z** de los elementos. Los elementos cuyo atributo **z-index** sea mayor se verán por delante de los elementos cuyo atributo **z-index** sea menor.

PROPIEDADES DE IMPRESIÓN		
Atributo CSS	**Javascript**	**Significado**
page-break-before	pageBreakBefore	Fuerza la inserción de un salto de página antes del elemento. (**auto**, **always**, **left**, **right**).
page-break-after	pageBreakAfter	Fuerza la inserción de un salto de página después del elemento. (**auto**, **always**, **left**, **right**).

PROPIEDADES DE FILTRADO		
Atributo CSS	**Javascript**	**Significado**
filter	filter	Permite aplicar un conjunto de filtros a los elementos.

OTRAS PROPIEDADES		
Atributo CSS	**Javascript**	**Significado**
active		Determina el estilo de un elemento cuando está activo.
link		Determina el estilo de un elemento cuando un usuario se sitúa un tiempo suficiente sobre el mismo.
hover		Determina el estilo de un elemento cuando todavía no ha sido visitado.
visited		Determina el estilo de un elemento cuando ha sido visitado.
cursor	cursor	Determina el icono que se visualiza como puntero del ratón (**auto**, **crosshair**, **default**,

		hand, **move**, **e-resize**, **ne-resize**, **nw-resize**, **n-resize**, **se-resize**, **sw-resize**, **s-resize**, **w-resize**, **text**, **wait**, **help**).
!important		Indica que la regla marcada como **importante** prevalezca sobre las demás.
@import		Se usa para importar una hoja de estilos.

Objeto Style: http://www.w3schools.com/jsref/dom_obj_style.asp

El HTML DOM permite a JavaScript cambiar el estilo de los elementos HTML:

http://www.w3schools.com/js/js_htmldom_css.asp

document.getElementById(id).style.property=new style

Ej: document.getElementById("myH1").style.color = "red";

10.2 POSICIONAMIENTO

El **posicionamiento absoluto** es la capacidad de situar un objeto, o bien un elemento, o un control, en cualquier punto de una página web estableciendo la esquina superior izquierda en coordenadas x, y por medio de sus atributos de estilo **left y top** respectivamente.

El **posicionamiento relativo** sitúa el objeto con respecto a su posición original como ocurre de manera natural dentro del código HTML.

El siguiente listado combina el posicionamiento absoluto y relativo utilizando dos bloques diferentes, conteniendo cada uno una imagen y un poco de texto que describe la imagen.

```
<html>
<head>
  <style type="text/css">
    body { margin-left: 0.25in; margin-top: 0.5in }
    #bloque1 {position: relative; top: 50px;
              left: 20px; width: 300px;
              background-color: lemonchiffon }
    #bloque2 {position: absolute; top: 50px;
              left: 400px; width: 300px;
              background-color: lightcoral}
  </style>
```

```
  <body>
    <div id="bloque1">
      <img src="imagenes/fluorita3tn.JPG">
      <strong>Fluorita</strong><p>
      La fluorita es un miembro de los <em>hálidos</em>. Existe en
      varios colores, y es un mineral bastante común. Recogí este
      fragmento debido a su rara coloración, que puede comprobarse
      en la fotografía. Esta muestra es una de las más grandes que
      poseo, con 10 cm de largo. Ello la convierte en una muestra
      "de mano", y procede de la Mina Denton, en Hardin County, Illinois.
    </div>
    <div id="bloque2">
      <img src="imagenes/apofilita2tn.JPG">
      <strong>Apofilita</strong><p>
      La apofilita es un miembro de la clase de los <em>filosilicatos</em>.
      Es un cristal incoloro, aunque tiene un matiz grisáceo. La muestra
      está asociada a la estilbita, que no es visible en la fotografía.
      Esta muestra procede de Poone (Pune), India. Este cristal puede
      diferenciarse de otros cristales claros por su relativo peso y es
      una piedra ligera con un tacto notablemente graso.
    </div>
</body>
</html>
```

Ejemplo anterior (Posicionamiento absoluto y relativo): **PosAbsolutoRelativo.html**

10.3 CONTENIDO

Las propiedades de texto, innerHTML y outerHTML

Las propiedades **innerHTML, outerHTML, innerText** y **outerText** son usadas para reemplazar contenidos en páginas web.

Las propiedades **innerHTML** y **outerHTML** funcionan con el HTML que está entre las etiquetas de inicio y fin del elemento. La diferencia entre las dos es que **innerHTML** reemplaza el contenido entre las etiquetas del elemento y la propiedad **outerHTML** reemplaza las etiquetas de inicio y fin del elemento, así como el contenido del elemento.

Las propiedades **innerText** y **outerText** funcionan igual pero trabajan solo con texto y no con etiquetas HTML.

Como ejemplo del uso de estas propiedades, veremos una versión electrónica del juego **tres en raya.**

```
<html>
<head>
  <style type="text/css">
    body { margin: 0.5in; background-color: white }
    td.val { background-color: yellow;
             width: 30; height:35; text-align:center }
    td.noval { background-color: lime;
               width: 30; height: 35; text-align: center }
```

```
    div.juego { margin-top: 0.5in; margin-left: 1.5in }
  </style>
  <script language="javascript">
    <!--
    var nombre_imagen = "o.gif"
    function jugar()
      {
      // Obtener lugar del click y colocar imagen
      var obj=window.event.srcElement;
      if (obj.className != 'val')
        return
      obj.className = 'noval';
      nombre_imagen = nombre_imagen == "Imagenes/x.gif" ?
                           "Imagenes/o.gif" : "Imagenes/x.gif";
      var mi_nombre_imagen = "<img src='" + nombre_imagen + "'>";
      obj.innerHTML = mi_nombre_imagen;
      }
    -->
  </script>
</head>
<body onClick="jugar()">
  <h1 id="cabecera"> Tres en raya con HTML dinámico</h1>
  <p>
     Bueno, espero que sepa cómo jugar a este juego... es tan viejo como
     el tiempo. El jugador 1 es "X" y el jugador 2 es "O". El primer
     jugador en conseguir tres cuadros iguales en una fila, en diagonal,
     horizontal o vertical, gana. Si ningún jugador consigue tres cuadros
     en una fila, el juego queda empatado.
  </p>
  <div class="juego">
     <table border="5" cellpadding="0" cellspacing="5">
       <tr><td id="uno" class=val> </td><td
            id="dos" class=val> </td>
            <td id="tres"class=val> </td></tr>
       <tr><td id="cuatro" class=val> </td><td
            id="cinco" class=val> </td>
            <td id="seis" class=val> </td></tr>
       <tr><td id="siete" class=val> </td><td
            id="ocho" class=val> </td>
            <td id="nueve" class=val> </td></tr>
     </table>
  </div>
</body>
</html>
```

Ejemplo anterior (El juego del tres en raya): **TresRaya.html**
(Funciona en IE y Chrome)

10.4 COLECCIONES

Una colección es un término VBscript, y representa un conjunto de valores. También se puede acceder a los conjuntos / colecciones actuales utilizando Javascript.

Muchas de las colecciones como forms, images y scripts hacen referencia a elementos que ya existían en forma de arrays antes de Internet Explorer 4, y no se tratarán en esta sección. Entre las creadas para HTML dinámico están las **colecciones all, rows, cells y styleSheets** que son las que estudiaremos.

La colección all

La **colección all es un listado de todos los elementos de un documento HTML en particular**. Utilice esta colección cuando necesite pasar por todos los elementos de una página por una determinada razón.

Con la **colección all**, se pueden **utilizar dos métodos** para acceder a los elementos dentro de una página web. El primero es el **método tags**, que permite acceder a todos los elementos de un determinado tipo de etiqueta y el segundo el **método item**, que puede utilizarse para acceder a cualquier elemento o bien por ubicación en la página o mediante identificador.

En el ejemplo siguiente listamos todas las etiquetas HTML presentes en la página web.

```
<html>
<head>
  <script language="javascript">
    <!--
    var nombres;
    function bucle_elementos()
      {
      var un_nombre;
      nombres = "";
      var ct;
      ct = 0;
      var grupo;
      grupo = document.all;
      for(var i = 0; i<grupo.length;i++)
        {
        un_nombre = grupo[i].tagName
        if (un_nombre!=null)
          {
          ct++;
          if (ct < 5)
            nombres += un_nombre + "   ";
          else
            nombres += un_nombre + "<br>";
          ct = 0;
          }
        }
      vis_resultados()
      }

    function vis_resultados()
      {
```

```
        document.all.resultados.innerHTML=nombres
        }
      -->
    </script>
  </head>
  <body onclick="bucle_elementos()">
    <h1 name="cabecera1"> Soy la cabecera uno</h1>
    <h2 name="cabecera2">Soy la cabecera dos</h2>
    <p name="parrafo1"> Soy el párrafo 1</p>
    <img src="Imagenes/x.gif" name="X">
    <ol name="list">
      <li name="item1">Este es el primer elemento</li>
      <li name="item2">Este es el segundo</li>
    </ol>
    <form name="un_formulario">
      <input type=”button” name="boton1" value="Soy el botón1">
    </form>
    <div>
      Y los resultados son:
    </div>
    <div id="resultados">
    </div>
  </body>
  </html>
```

Ejemplo anterior (Visualizar todas las etiquetas HTML presentes): **PruebaAll2.html**
(Funciona en todos los navegadores)

Las colecciones cells y rows

La **colección cells es un array que hace referencia a todas las celdas de una fila de tabla**, ya utilicen la etiqueta de cabecera de tabla <th> o la de datos <td>. La **colección rows tiene un conjunto con todas las filas de una tabla.**

El siguiente listado contiene una tabla con cada celda en diferentes colores de fondo: verde lima, verde, rojo, o azul. Los usuarios pulsan un botón para indicar que han realizado una determinada tarea, y el color de fondo, el color de la fuente y su familia se cambian para indicar que la tarea ha sido realizada.

```
<html>
  <head>
    <script language="javascript">
      <!--
      function bucle_elements()
        {
        var filas = tabla.rows
        // por cada fila
        for (var i=0;i<filas.length;i++)
          {
          var columnas = filas[i].cells
          bandera = 1
          // para cada columna, comprobamos color
          for ( var j=0; j < columnas.length; j++)
            {
            if (columnas[j].style.backgroundColor == "red")
              {
              bandera=0
```

```
          columnas[j].style.backgroundColor = "maroon"
          }
        else if (columnas[j].style.backgroundColor == "blue")
          {
          bandera=0
          columnas[j].style.backgroundColor = "navy"
          }
        else if (columnas[j].style.backgroundColor == "lime")
          {
          bandera=0
          columnas[j].style.backgroundColor = "green"
          }
        else if (columnas[j].style.backgroundColor == "white")
          {
          bandera=0
          columnas[j].style.backgroundColor = "black"
          }
        // si hubo cambio de color de fondo, cambiamos color de
        // primer plano y fuente
        if (bandera == 0)
          {
          columnas[j].style.color="white"
          columnas[j].style.fontFamily="Cursive"
          break;
          }
        } // fin for j
      // si no hay mas valores que cambiar salimos del bucle
      if (bandera == 0)
         break;
      } // fin for i
    } // fin function
  -->
</script>
</head>
<body>
  <center>
  <input type=button value="Púlseme cuando termine cada tarea"
                              onClick="bucle_elements()">
  </center>
  <p>
    <table id="tabla" cols="4" border="3" cellspacing="5"
                                          align="center">
      <tr>
        <td width="125" align=center style="background-color: lime">
          Tarea Uno
        </td>
        <td width="125" align=center style="background-color: white">
          Tarea Dos
        </td>
        <td width="125" align=center style="background-color: blue">
          Tarea Tres
        </td>
        <td width="125" align=center style="background-color: red">
          Tarea Cuatro
        </td>
      </tr>
      <tr>
        <td width="125" align=center style="background-color: white">
          Tarea Cinco
```

```
          </td>
          <td width="125" align=center style="background-color: blue">
            Tarea Seis
          </td>
          <td width="125" align=center style="background-color: red">
            Tarea Siete
          </td>
          <td width="125" align=center style="background-color: lime">
            Tarea Ocho
          </td>
        </tr>
        <tr>
          <td width="125" align=center style="background-color: blue">
            Tarea Nueve
          </td>
          <td width="125" align=center style="background-color: red">
            Tarea Diez
          </td>
          <td width="125" align=center style="background-color: lime">
            Tarea Once
          </td>
          <td width="125" align=center style="background-color: white">
            Tarea Doce
          </td>
        </tr>
      </table>
  </body>
  </html>
```

Ejemplo anterior (Modificar el contenido de las celdas de una tabla):
PruebaCellsRows.html (funciona en todos los navegadores)

Las colecciones styleSheets e imports

La **colección styleSheets** contiene una referencia a todas las hojas de estilo de una página web, mientras que la **colección imports** contiene todas las hojas de estilo incluidas mediante el uso del atributo import.

Una vez que se accede a una determinada hoja de estilo, se puede añadir una hoja de estilo completamente nueva con el **método addImport**, o añadir nuevas reglas de hoja de estilo con el **método addRule** (esto con Internet Explorer).

Como ejemplo realizamos cambio de hoja de estilo y añadimos nuevas reglas sobre la página PruebastyleSheets.htm al pulsar sobre ella.

```
  <script language="javascript">
      num=1;
     function cambio(){
      if (num == 1)
        {
          num = 2;
```

```
      //document.styleSheets[0].href="./estilo2.css";  Este no se ejecuta automáticamente

         document.getElementsByTagName('link')[0].href="estilo2.css";

      //document.styleSheets[0].addRule("body", "background-color:beige");
                     Sólo Internet Explorer
         }
       else
         {
             num = 1;
         //document.styleSheets[0].href="estilo1.css";
             document.getElementsByTagName('link')[0].href="estilo1.css";

         }
    }
</script>
```

Ejemplo anterior (Cambiar de hoja de estilo y añadir nuevas reglas):
PruebaStyleSheets.html (funciona en todos los navegadores)

10.5 HTML 5: CANVAS

¿Qué es Canvas?

Canvas es un **elemento** de **HTML 5** que permite dibujar **gráficos**, manipular imágenes y realizar animaciones en una página web de forma **dinámica**. Es decir, el gráfico se crea en el momento en que se carga la página. El gráfico se define mediante programación, usualmente Javascript. Una vez creado el gráfico se pueden programar acciones para que el usuario interactue con él, por ejemplo, haciendo clic para arrancar una animación o arrastrando un elemento del gráfico.

Aunque el elemento Canvas se crea con la etiqueta ***<canvas> </canvas>*** al hablar de Canvas en sentido amplio nos referimos a toda la API que incluye un conjunto de funciones para dibujar líneas, rectángulos, círculos, etc. así como para rotar, escalar, transformar elementos gráficos, y otras funciones de variado uso.

Al definir un Canvas en una página web se crea un lienzo de dibujo o área de dibujo rectangular. Después de crear el dibujo lo que queda es un "**mapa de bits**", es decir, cada coordenada del lienzo tiene asignado un color, no queda ninguna estructura de lo que contiene el lienzo. Canvas no crea objetos vectoriales al estilo de otros entornos como SVG o Flash, sino mapas de bits como una imagen fotográfica.

De hecho podemos cargar una imagen fotográfica en el Canvas y manipular sus pixeles uno a uno.

Canvas ha sido incorporado recientemente al HTML y **no puede ser representado por los navegadores antiguos**. Se requiere una versión actualizada de Chrome (2.6), Firefox (2.0), Internet Explorer(10), Safari (6.0), etc. Es conveniente dar un contenido alternativo para los navegadores que no soportan Canvas, como una imagen fija o un texto aclaratorio, como veremos más adelante.

Empezando a dibujar

Canvas permite realizar gráficos de todo tipo, desde una línea recta hasta juegos interactivos, también permite manipular imágenes para cambiarles el tono, buscar bordes, aplicaciones de visión artificial, etc. Aquí veremos sólo una introducción al mundo Canvas.

Vamos a ver un ejemplo sencillo que dibuja un rectángulo gris de 450 x 350 pixels ocupando todo el Canvas. Sobre este ejemplo explicaremos los conceptos básicos sobre Canvas.

```
<html>
  <head>
    <title>Ejemplo1 Canvas </title>
    <script type="text/javascript">
      function dibujarCanvas(){
        var canvas = document.getElementById('miCanvas');
        var contexto = canvas.getContext('2d');
           contexto.fillStyle = '#CCCCCC';
           contexto.fillRect (0, 0, 450, 350);
        }
    </script>
  </head>
  <body onLoad="dibujarCanvas();">
  <p>Dibujo con  Canvas:</p>
    <canvas id="miCanvas" width="450" height="350">Su navegador no soporta Canvas.</canvas>
  </body>
</html>
```

En el Body del documento html se declara el elemento <canvas>:

<canvas id="miCanvas" width="450" height="350">Su navegador no soporta Canvas.</canvas>

Los atributos **width** y **height** definen el tamaño del Canvas, es decir, el tamaño de la superficie o lienzo sobre el que dibujaremos.

La posición que ocupe este lienzo del Canvas dentro de la página web vendrá definida por el lugar en el que escribamos el elemento <canvas>, como cualquier otro elemento de HTML. Por ejemplo, podemos colocarlo dentro de una tabla o dentro de un contenedor flotante.

El elemento <canvas> debe cerrarse con </canvas>, si escribimos algo entre estas dos etiquetas se ejecuratá en el caso que el navegador no soporte Canvas. En nuestro ejemplo aparecerá el texto "Su navegador no soporta Canvas", también podemos poner una etiqueta <img> con una captura del dibujo que crea el Canvas.

Para poder referenciar al elemento concreto de Canvas en el código Javascript es conveniente darle un identificador, por ejemplo, id="miCanvas".

Una vez hemos creado el elemento Canvas debemos ejecutar el código Javascript que realice los dibujos, esto hay que hacerlo después que el elemento Canvas se haya creado. Una forma de hacerlo es con la propiedad **onLoad:**

<body onLoad="dibujarCanvas();">

Una vez cargada la página ejecutamos la función dibujarCanvas()

La función que dibuja el Canvas tiene la misma sintaxis que cualquier función Javascript, vamos a ver cada instrucción.

```
<script type="text/javascript">
   function dibujarCanvas(){
     var canvas = document.getElementById('miCanvas');
     var contexto = canvas.getContext('2d');
        contexto.fillStyle = '#CCCCCC';
        contexto.fillRect (0, 0, 450, 350);
     }
 </script>
```

Lo primero que tenemos que hacer es obtener el nodo DOM mediante la instrucción:

var canvas = document.getElementById('miCanvas');

Le hemos pasado como parámetro el identificador que dimos al elemento Canvas.

A continuación debemos obtener el objeto contexto:

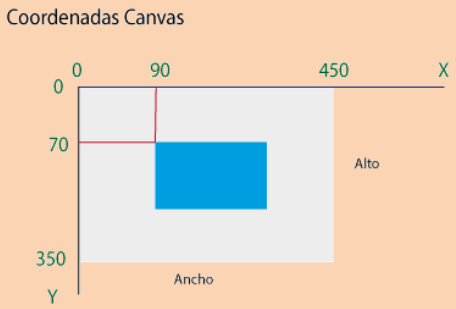
var contexto = canvas.getContext('2d');

Hay dos posibles contextos, "2d" para dibujar en dos dimensiones y "webgl" para dibujar en tres dimensiones. Una vez tenemos el objeto contexto ya podemos utilizar todos las funciones y propiedades que tiene asociados para dibujar.

La propiedad **fillStyle** define el color de relleno.

La función o método **fillRect(x, y, ancho, alto)** dibuja un rectángulo en la posición (x, y) de una anchura y altura. La posición se refiere a la esquina superior izquierda del rectángulo.

Ahora debemos explicar el sitema de coordenadas **x, y** que utliza Canvas.



El origen de coordenadas es el punto (0,0) los valores de la “coordenada **x**” van hacia la derecha **y** los valores de la “coordenada **y**” van hacia abajo (al contrario que en los sitemas de coordenadas tradicionales).

Al situar un objeto lo haremos respecto al origen, por ejemplo, en la figura el rectángulo azul está en la posición (90,70). El tamaño del lienzo del Canvas es, en este caso, de 450 de ancho y 350 de alto.

Por lo tanto, la instrucción:

contexto.fillRect (0, 0, 450, 350);

Dibujará un rectángulo relleno con su esquina superior derecha en el punto (0,0) y 450 de ancho y 350 de alto, es decir, el rectángulo ocupará todo el lienzo de nuestro ejemplo.

Hemos visto cómo dibujar un rectángulo relleno con un color, para dibujar un rectángulo sin relleno tenemos la función:

strokeRect(x,y,ancho, alto)

Ejemplo anterior: **canvasInicio.htm**

Ejemplo Práctico: Creando un Canvas

El elemento <canvas> es similar a los tags , <a> o <table>, con la excepción de que su contenido son renderizados con **JavaScript**.

Para implementarlo, necesitamos escribir el tag **<canvas>** en algún lugar del código **HTML** y crear una función de JavaScript que acceda a dicho tag una vez que la página carga y luego utilizar las APIs de **Canvas de HTML5** para que se puedan visualizar nuestros dibujos.

```
 <body>
      <canvas id="myCanvas"></canvas>
</body>
```

HTML5 Canvas Template (plantilla)

```
<!DOCTYPE HTML>
<head>
<script>
     window.onload = function() {
                     var canvas = document.getElementById("myCanvas");
                     var context = canvas.getContext("2d");

                     // sigue el código

                     };

 </script>
 </head>
 <body>
 <canvas id="myCanvas" width="578" height="200"></canvas>
```

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

Ejemplo de plantilla: **canvastemplate.htm**

El código anterior puede ser tomado como base de todos nuestros futuros proyectos de **Canvas**. Podemos definir el ancho y alto utilizando las propiedades **width** y **height**. En la función inicializadora podemos acceder al objeto canvas **DOM** a través del id y obtener el contexto 2-d utilizando **getContext()**.

Líneas

El método **beginPath()** define un nuevo path de dibujo. El **método *moveTo()*** es una forma de posicionar el cursor de dibujo. El **método lineTo()** dibuja una línea entre los puntos dados. El **método stroke()** asigna color a la línea y la hace visible. El color por defecto (default) es negro.

```
<!DOCTYPE HTML>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #myCanvas {
    border: 1px solid #9C9898;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");

    context.beginPath();
    context.moveTo(100, 150);
    context.lineTo(450, 50);
    context.stroke();
   };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
 </body>
</html>
```

Ejemplo anterior: **canvaslinea.htm**

Otros atributos:

lineWidth = ancho de la línea.

strokeStyle = color de la línea.

Dibujar formas

En Canvas **no hay funciones definidas para dibujar formas poligonales** (salvo para el rectángulo), no existen funciones para dibujar triángulos, pentágonos, etc. La forma de trabajar con Canvas es dibujar "**formas**" o "**caminos**" (paths).

Para construir una forma lo primero es inicializarla con la función **beginPath()**, luego se coloca el cursor en el punto de inicio con la función **moveTo(x,y)** y a partir de ahí se va creando la forma con las distintas funciones existentes para dibujar líneas, arcos, etc. Una vez hemos acabado la forma podemos usar dos funciones para dibujarla realmente sobre el lienzo.

1. **stroke()** dibuja una forma abierta o cerrada, sin color de relleno. Si hacemos coincidir el último punto con el primero la forma será cerrada.

2. **fill()** dibuja una forma cerrada con el color de relleno actual. Si la forma estaba abierta, la propia función **fill()** crea una línea recta desde el último punto al punto inicial para cerrarla.

También se puede usar la función **closePath()** que crea una línea.

```
<html>
  <head>
    <title>Ejemplo2 Canvas </title>
    <script type="text/javascript">
      function dibujarCanvas(){
        var canvas = document.getElementById('miCanvas');
        var contexto = canvas.getContext('2d');
                    // a. Forma abierta
                      contexto.beginPath();
                      contexto.moveTo(50,50);
                      contexto.lineTo(50,150);
                      contexto.lineTo(150,150);
                      contexto.stroke();

                     // b. Forma cerrada con  relleno
                      contexto.beginPath();
                      contexto.moveTo(200,50);
                      contexto.lineTo(200,150);
                      contexto.lineTo(300,150);
                      contexto.fill();
```

```
                // c. Forma  cerrada sin relleno
                  contexto.beginPath();
                  contexto.moveTo(50,200);
                  contexto.lineTo(50,300);
                  contexto.lineTo(150,300);
                  contexto.closePath();
                  contexto.stroke();
        }
   </script>
  </head>
  <body onLoad="dibujarCanvas();">
   <canvas id="miCanvas" width="450" height="350">Su navegador no soporta Canvas.</canvas>
  </body>
</html>
```

Ejemplo anterior: [canvasformas1.htm]

Formas curvadas

Para dibujar curvas diponemos de tres métodos básicos:

1. **Arcos.**
2. **Curvas de Bezier.**
3. **Curvas cuadráticas**.

1. Arcos.

Para dibujar arcos tenemos la función:

arc(x, y, radio, ángulo inicial, ángulo final, sentido de giro)

Donde los parámetros **(x,y)** son las coordenadas del centro de la circunferencia cuyo arco vamos a dibujar. El parámetro **radio** es el radio de dicha circunferencia. La amplitud del arco irá desde el **ángulo inicial** al **ángulo final** en el sentido de las agujas del reloj. Los ángulos se miden en radianes. La equivalencia con los grados nos la da esta expresión: radianes = (Math.PI/180)*grados. Donde Math.PI es el número Pi (3,1416...)

Si el párametro **sentido de giro** tiene el valor lógico *verdadero*, el arco irá en sentido contrario a las agujas del reloj. Este último parámetro es opcional y tiene el valor por defecto *falso*.

Vamos a ver unos ejemplos que nos harán entender esta definición.

```
!DOCTYPE HTML>
  <head>
   <title>Ejemplo3 Canvas </title>
```

```
  <script type="text/javascript">

    function dibujarCanvas(){

      var canvas = document.getElementById('miCanvas');
      var contexto = canvas.getContext('2d');

                  // a. Arco desde 0 grados hasta 90 grados en sentido de las agujas del reloj
                  contexto.beginPath();
                  contexto.arc(60,60,50,radianes('0'),radianes('90'),false);
                  contexto.stroke();

                  // b. Arco desde 0 grados hasta 90 grados en sentido contrario de las agujas del reloj
                  contexto.beginPath();
                  contexto.arc(200,60,50,radianes('0'),radianes('90'),true);
                 contexto.stroke();

                   // c. Arco desde 90 grados hasta 270 grados en sentido de las agujas del reloj
                  contexto.beginPath();
                  contexto.arc(350,60,50,radianes('90'),radianes('270'));
                 contexto.stroke();
        }
         function radianes(grados){
                  var radianes = (Math.PI/180)*grados;
                  return radianes;
        }
   </script>
  </head>
  <body onLoad="dibujarCanvas();">
    <canvas id="miCanvas" width="450" height="350">Su navegador no soporta Canvas.</canvas>
  </body>
</html>
```

Ejemplo anterior: canvasArcos.htm

2. Curvas de Bezier.

Las curvas de Bezier son un tipo de curvas muy usadas en diseño que necesitan un punto de inicio, un punto final y unos puntos de control. Son un poco dificiles de explicar con palabras, sin embargo en este enlace de Wikipedia hay unos gráficos animados muy didácticos.

La mejor forma de familiarizarse con las curvas de Bezier es dibujarlas con programas como Illustrator o Inkscape (gratuito) o usar este simulador.

bezierCurveTo(cp1x, cp1y, cp2x, cp2y, x, y) donde **cp1x, cp1y**, son las coordenadas del primer punto de control, **cp2x, cp2y**, son las coordenadas del segundo punto de control, **x, y** son las coordenadas del

punto final de la curva. El primer punto de la curva será el punto actual, al que habremos llegado con *moveTo* o con cualquier función de dibujo (arc, lineTo, ...)

3. Curvas cuadráticas.

Son similares a las anteriores pero con un solo punto de control.

quadraticCurveTo(cp1x, cp1y, x, y) donde **cp1x, cp1y,** son las coordenadas del primer punto de control, **cp2x, cp2y** mientras que **x, y** son las coordenadas del punto final de la curva. El primer punto de la curva será el punto actual.

Veamos unos ejemplos.

```
<!DOCTYPE HTML>
 <head>
   <title>Ejemplo 4 Canvas </title>
   <script type="text/javascript">
    function dibujarCanvas(){
      var canvas = document.getElementById('miCanvas');
      var contexto = canvas.getContext('2d');
                    // curva cuadratica, esquina bien redondeada
                   contexto.beginPath();
                   contexto.moveTo(125,25);
                   contexto.lineTo(75,25);
                   contexto.quadraticCurveTo(25,25,25,75);
                   contexto.lineTo(25,125);
                   contexto.stroke();
                    // curva cuadratica, esquina poco redondeada
                   contexto.beginPath();
                   contexto.moveTo(350,25);
                   contexto.lineTo(300,25);
                   contexto.quadraticCurveTo(265,35,250,75);
                   contexto.lineTo(250,125);
                   contexto.stroke();
          // curva bezier, esquinas redondeadas hacia afuera
                   contexto.beginPath();
                   contexto.moveTo(125,200);
                   contexto.lineTo(75,200);
                   contexto.bezierCurveTo(35,185,10,210,25,250);
                   contexto.lineTo(25,300);
                   contexto.stroke();
                          // dos curvas bezier, en forma de letra ese
                   contexto.beginPath();
                   contexto.moveTo(300,200);
                   contexto.bezierCurveTo(260,200,200,220,260,250);
                   contexto.bezierCurveTo(325,275,350,300,300,300);
                   contexto.stroke();
```

```
        }
    </script>
  </head>
  <body onLoad="dibujarCanvas();">
    <canvas id="miCanvas" width="450" height="350">Su navegador no soporta Canvas.</canvas>
  </body>
</html>
```

La primera figura esta formada por dos líneas rectas unidas por una curva cuadrática con el punto de control en la interseccion de las dos rectas.

La figura de la parte superior derecha esta formada por dos líneas rectas unidas por una curva cuadrática con el punto de control un poco más abajo de la interseccion de los dos lados rectos.

La figura de la parte inferior izquierda esta formada por dos líneas rectas unidas por una curva de Bezier con el punto de control un poco más arriba de la interseccion de las dos líneas rectas.

La figura de la parte inferior derecha está formada por dos curvas de Bezier con los puntos de control como se observa en la figura.

Ejemplo anterior: **canvasCurvasCuadráticas.htm**

Paths (caminos) y Curvas (mezcla de los vistos anteriormente)

Para crear un camino (path) con **HTML5 Canvas**, podemos conectar multiples subpaths. El punto final de cada subpath será el nuevo punto de contexto. Podemos usar los métodos **lineTo(), arcTo(), quadraticCurveTo() y bezierCurveTo()** para construir cada subpath. Podemos usar el método **beginPath()** cada vez que deseemos crear un nuevo path.

```
<!DOCTYPE HTML>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
      #myCanvas {
        border: 1px solid #9C9898;
      }
    </style>
    <script>
      window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");

        context.beginPath();
```

```
      context.moveTo(100, 20);

      // line 1
      context.lineTo(200, 160);

      // quadratic curve
      context.quadraticCurveTo(230, 200, 250, 120);

      // bezier curve
      context.bezierCurveTo(290, -40, 300, 200, 400, 150);

      // line 2
      context.lineTo(500, 90);

      context.lineWidth = 5;
      context.strokeStyle = "blue";
      context.stroke();
    };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
 </body>
</html>
```

Ejemplo anterior: [canvaspathycurvas.html]

Formas

Para crear una forma customizada con **HTML5 Canvas**, podemos crear un path y luego cerrarlo utilizando el método **closePath()**. Podemos usar los métodos **lineTo(), arcTo(), quadraticTo() o bezierCurveTo()** para construir cada subpath que necesitemos.

Con la propiedad **fillStyle** podemos setear el color de la forma.

```
<!DOCTYPE HTML>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #myCanvas {
    border: 1px solid #9C9898;
```

```
    }
   </style>
   <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");

      // begin custom shape
      context.beginPath();
      context.moveTo(170, 80);
      context.bezierCurveTo(130, 100, 130, 150, 230, 150);
      context.bezierCurveTo(250, 180, 320, 180, 340, 150);
      context.bezierCurveTo(420, 150, 420, 120, 390, 100);
      context.bezierCurveTo(430, 40, 370, 30, 340, 50);
      context.bezierCurveTo(320, 5, 250, 20, 250, 50);
      context.bezierCurveTo(200, 5, 150, 20, 170, 80);

      // complete custom shape
      context.closePath();
      context.fillStyle = "#8ED6FF";
      context.fill();
      context.lineWidth = 5;
      context.strokeStyle = "blue";
      context.stroke();
    };

   </script>
  </head>
  <body>
   <canvas id="myCanvas" width="578" height="200"></canvas>
  </body>
</html>
```

Ejemplo anterior: [canvasformas.htm]

Rectángulos

Para crear rectángulos podemos usar el método **rect()**. A este lo podemos posicionar con los parámetros x e y, y podemos setear su ancho y alto con **width** y **height** respectivamente. La posición es relativa al borde superior izquierdo del rectángulo.

```
<!DOCTYPE  HTML>
 <head>
  <style>
   #myCanvas {
```

```
      border: 1px solid #9C9898;
    }
    body {
      margin: 0px;
      padding: 0px;
    }
  </style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');

      context.beginPath();
      context.rect(188, 50, 200, 100);
      context.fillStyle = '#8ED6FF';
      context.fill();
      context.lineWidth = 5;
      context.strokeStyle = 'black';
      context.stroke();
    };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
 </body>
</html>
```

Ejemplo anterior: [canvasrectangulo.htm]

Círculo

Para dibujar un círculo en **HTML5 Canvas** podemos crear un arco usando el método **arc()** y definir como ángulo de comienzo 0 y el ángulo final como 2 * PI.

```
<!DOCTYPE HTML>
<html>
  <style>
    body {
      margin: 0px;
      padding: 0px;
    }
    #myCanvas {
      border: 1px solid #9C9898;
    }
  </style>
```

```
  <script>
    window.onload = function() {
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      var centerX = canvas.width / 2;
      var centerY = canvas.height / 2;
      var radius = 70;

      context.beginPath();
      context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
      context.fillStyle = "#8ED6FF";
      context.fill();
      context.lineWidth = 5;
      context.strokeStyle = "black";
      context.stroke();
    };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
 </body>
</html>
```

Ejemplo anterior: [canvascirculo.htm]

Combinando formas y estilos

Hemos visto que para dibujar una forma hay que inicializarla (**beginPath()**) colocar el cursor (**moveTo(x,y)**) crear las formas, opcionalmente cerrarla (**closePath()**) y dibujarla con relleno **stroke()** o sin relleno **fill()**.

Este proceso se puede repetir tantas veces como se desee. Podríamos pensar que si estamos dibujando con un lápiz en una hoja de papel, cada vez que acabamos un trazo y levantamos el lápiz del papel para colocarlo en una nueva posición deberíamos hacer un **moveTo(x, y)**.

La excepción a la forma general de construir formas son las funciones para dibujar rectángulos **(strokeRect(x,y,ancho,alto)** y **fillRect(x,y,ancho,alto))** que implícitamente empiezan con un **moveTo(x,y)** y acaban con **stroke()** o **fill()**.

Dentro de cada forma podemos enlazar y combinar todas las funciones de dibujo que deseemos. Cada vez que inicializamos una forma, Canvas inicializa una estructura de datos en forma de lista, esa lista se va rellenando con cada instrucción de dibujo y cuando ejecutamos **stroke()** o **fill()** se ejecutan todas la instrucciones de la lista.

Colores

Hasta ahora hemos dibujado con el color por defecto de Canvas que es el negro. Para cambiar el color del trazo tenemos la propiedad **strokeStyle** y para el color de relleno la propiedad **fillStyle**. Podemos asignarles un color en formato hexadecimal, por ejemplo,

strokeStyle = '#FFFF99'

o en formato RGB, por ejemplo,

fillStyle = 'rgb(221,64,4)'

También se puede poner el nombre en inglés de los colores simples (white, black, red, green, blue, ...)

Una forma de fijar la **transparencia** de un color es utilizar la función **rgba(r,g,b,a)**, donde el último parámetro determina la transparencia, debe estar comprendido entre 0 (totalmente transparente) y 1 (sin transparencia). Por ejemplo,

fillStyle = 'rgba(221,64,4, 0.2)'

da un color rojo transparente al 80% (u opaco al 20%)

Si queremos que aparezca el color del trazo y el color del relleno, debemos colocar la instrucción *strokeStyle* después de *fillStyle*, en otro caso el relleno tapará el trazo.

Ancho de línea

La propiedad **lineWidth** fija el ancho del trazo, por ejemplo,

lineWidth = 5

Ejemplo anterior: **canvasCombinado.html**

Degradados.

Para dibujar degradados lineales Canvas dispone de la función:

createLinearGradient(x_ini,y_ini,x_fin,y_fin)

Esta función define un degradado lineal mediante una **línea** definida por su punto inicial (x_ini,y_ini) y su punto final (x_fin, y_fin), la dirección y el sentido de esta línea determina la dirección y el sentido del degradado. Por ejemplo, si queremos que el degradado vaya en la dirección vertical, deberemos dar una línea vertical, como veremos a continuación en el primer ejemplo.

El sentido de la línea determina el primer color del degradado, en el ejemplo que vamos a ver a continuación, como el sentido es hacia abajo el primer color es el rojo, mientras que si el sentido fuese hacia arriba el primer color sería el verde. La longitud de la línea indica la intensidad del degradado, cuanto más larga, más suave. Si habeis utilizado programas como Photoshop es funcionamiento es el mismo.

Para acabar de definir un degradado hay que indicar los colores que lo forman mediante la función:

addColorStop(posición, color)

El parámetro posición determina desde que lugar de la línea de degradado se aplica el color indicado por el segundo parámetro. La posición se expresa con un número entre cero y uno. Como mínimo, se necesitan dos colores distintos para formar un degradado.

Por ejemplo, si queremos un degrado que empiece en rojo y acabe en blanco

```
addColorStop(0, 'red');

addColorStop(1, 'white');
```

Para dibujar degradados radiales Canvas dispone de la función:

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

Esta función define un degradado radial en base a dos circunferencias definidas, cada uno de ellas, por la posición de su centro (x,y) y su radio (r). Si el centro de ambas circunferencias es el mismo el efecto será de un degradado uniforme, mientras que en otro caso se puede producir un efecto de esfera o más extraño, según los tamaños de los radios.

Ejemplo anterior: canvasDegradado.html

Imágenes

Canvas puede trabajar con imágenes en los **formatos más comunes** (GIF, JPEG, PNG) y dispone de funciones para manipular las imágenes a nivel completo (escalarlas, girarlas,...) y también a nivel de pixel. Esto da una gran potencia al manejo de imágenes con Canvas. Se pueden usar imágenes como fondo, se pueden realizar miniaturas y ampliaciones, se les puede cambiar el color, etc.

El primer paso es **cargar** la imagen, hay varios métodos para hacerlo, pero siempre hay que tener presente que hasta que la imagen no se haya cargado completamente no se puede trabajar con ella.

Veamos dos de los métodos más comunes para cargar imágenes:

1- Creando el objeto dentro de Javascript

Mediante **new Image** creamos una instancia del objeto Image, luego le asignamos una imagen y no empezamos a usarla hasta que no se haya cargado.

Por ejemplo:

```
var imagen = new Image();
imagen.src = 'http://www.aulaclic.es/articulos/graficos/cabeza_pato.png';
imagen.onload = function(){
// usar la imagen }
```

2- Usando una imagen de la página web

Con la etiqueta <*img*> cargamos la imagen desde el body de la página web y le damos un identificador (por ejemplo, *"nombre_imagen"*), para acceder a la imagen desde Javascript utilizamos

document.getElementById("nombre_imagen")

Si queremos que la imagen se vea solo después de procesarla con Canvas podemos utilizar la propiedad "hidden" en la etiqueta <*img*>.

Hay más formas de cargar imágenes, por ejemplo, se puede utilizar el imagen generada en otro objeto Canvas. También se pueden cargar mediante el método "data:url".

Una vez tenemos cargada la imagen podemos dibujarla mediante la función **drawImage** que admite varios formatos:

1. **drawImage(imagen, x, y)**

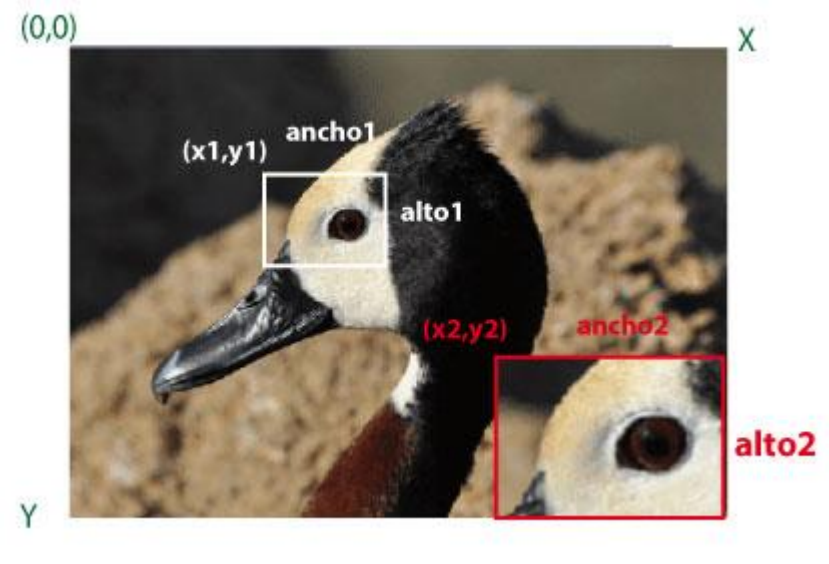
Dibuja la imagen en el lienzo colocando la esquina superior izquierda de la imagen en las coordenadas (x,y)

2. **drawImage(imagen, x, y, ancho, alto)** Permite, además, escalar la imagen.

Los tres primeros parámetros tienen el mismo significado que en el caso anterior. Los parámetros **ancho** y **alto** definen el tamaño (en pixels) con el que se dibujará la imagen. Si son mayores que el original obtendremos una ampliación de la imagen, mientras que si son menores obtendremos una miniaturización de la imagen. Si queremos que la imagen no se deforme deberemos mantener la proporción *ancho/alto* de la imagen original.

3. **drawImage(imagen, x1, y2, ancho1, alto1, x2, y2, ancho2, alto2)** Permite escalar una porción de la imagen.

Vamos a explicar este caso sobre la imagen siguiente:



Lo que se dibujaría en el lienzo con esta función es **sólo** el contenido del recuadro rojo, hemos incluido como fondo la imagen original para poder explicarlo mejor.

Esta función toma una porción de la imagen original (definida por x1,y1, ancho1, ancho2) y dibuja esa porción en otra posición (x2,y2) y con otra escala (ancho2, alto2)

En este caso, hemos creado una ampliación de la zona del ojo del pato porque los valores de ancho2 y alto2 son mayores que ancho1 y alto1.

Ejemplo anterior: **canvasImagen1.html**

EJEMPLO SENCILLO:

Para dibujar una imagen usando **HTML5 Canvas**, podemos usar el método **drawImage()** que requiere un objeto imagen y un punto de destino. Este último es relativo al borde izquierdo superior de la imagen.

```
<!DOCTYPE HTML>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #myCanvas {
    border: 1px solid #9C9898;
   }
  </style>
  <script>
   window.onload = function() {
```

```
      var canvas = document.getElementById("myCanvas");
      var context = canvas.getContext("2d");
      var imageObj = new Image();

      imageObj.onload = function() {
        context.drawImage(imageObj,  110,  90);
      };
      imageObj.src = "imagenes/rhodie.jpg";
    };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="400"></canvas>
 </body>
</html>
```

Ejemplo anterior: **canvasimagen.htm**

Texto: Fuentes, Tamaños y Estilos

Para cambiar la fuente, el tamaño y el estilo de un texto en **HTML5 Canvas**, podemos usar la **propiedad font del contexto de canvas**. El estilo puede ser normal, negrita o cursiva.

```
<!DOCTYPE HTML>
 <head>
  <style>
   body {
    margin: 0px;
    padding: 0px;
   }
   #myCanvas {
    border: 1px solid #9C9898;
   }
  </style>
  <script>
   window.onload = function() {
    var canvas = document.getElementById("myCanvas");
    var context = canvas.getContext("2d");
    var x = 80;
    var y = 110;

    context.font = "60pt Calibri";
    context.lineWidth = 3;
```

```
      // stroke color
      context.strokeStyle = "blue";
      context.strokeText("Hola Mundo!", x, y);
    };

  </script>
 </head>
 <body>
  <canvas id="myCanvas" width="578" height="200"></canvas>
 </body>
</html>
```

Ejemplo anterior: [canvastexto.htm]

Hoja resumen de órdenes Canvas: HTML5 Canvas Cheat Sheet.pdf

HTML5 canvas

http://www.w3schools.com/html/html5_canvas.asp

HTML5 canvas reference

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

EJEMPLOS EFECTOS Y ANIMACIONES

Movimiento oscilante objeto **canvasMovIzqDer.html**

Dibujo abstracto en movimiento **canvasMovimiento.html**

Transformations

Rotate	**canvasRotate.html**
State Stack	**canvasStateStack.html**

Composites

Shadows	**canvasShadow.html**
Global Alpha	**canvasGlobalAlpha.html**
Clipping Region	**canvasClipplingRegion.html**
Operations	**canvasOperations.html**

Image Data & URLs

Image Data	**canvasImageData.html**	
Invert Colors	**canvasInvertColors.html**	
Grayscale	**canvasGraceScale1.html**	**canvasGraceScale.html**
Get Data URL	**canvasGetDataURL.html**	
Save Drawing as an Image	**canvasSaveDrawing.html**	

Animation

Clear Canvas	**canvasClearCanvas.html**
Animation Frames	**canvasAnimationFrames.html**
Linear Motion	**canvasLinearMotion.html**
Acceleration	**canvasAcceleration.html**
Oscillation	**canvasOscillation.html**
Start and Stop	**canvasStartStop.html**

Barra de progreso (progress Bar) **canvasprogressbar.htm**

Más Ejemplos de canvas (muy artísticos):

http://www.forosdelweb.com/f4/coleccion-ejemplos-con-canvas-954714/

Frameworks

Los **frameworks** son librerías con multitud de funciones predefinidas que ayudan a trabajar más eficientemente con Canvas, liberando al programador de realizar muchas tareas rutinarias.

Algunos de los frameworks ultizados con Canvas son los siguientes:

- EaselJS
- Fabric.js
- KineticsJS
- Paper.js
- three.js