PCW

PROGRAMACIÓN DEL CLIENTE WEB

Tema 07 - API's HTML5

Dept. de Ciència de la Computació i Intel·ligència Artificial
Dpto. de Ciencia de la Computación e Inteligencia Artificial

Universitat d'Alacant
Universidad de Alicante

API's HTML5

- ✓ Introducción
- ✓ Vídeo y audio
- ✓ Gráficos en HTML: canvas y SVG
- ✓ Drag & drop
- ✓ Web Workers
- ✓ Geolocalización
- ✓ Web Storage
- ✓ IndexedDB

Introducción

Introducción

HTML proporciona una serie de APIs (**A**pplication **P**rogramming **I**nterface) que mejoran las posibilidades de programación de los desarrolladores y diseñadores web.

Están en constante evolución, pudiendo aparecer nuevas APIs, cambiar alguna de las existentes o, incluso, desaparecer.

Entre las APIs de HTML, se encuentran las siguientes:

- ✓ **Vídeo y audio**
- ✓ **Canvas**
- ✓ **Geolocalización**
- ✓ **Drag & drop**
- ✓ **Web Storage**
- ✓ **Web workers**
- ✓ **Indexed Database API**
- ✓ **Web sockets**
- ✓ **File API**
- ✓ **Application cache**

Vídeo y audio

Vídeo y audio

- ✓ Permiten ejecutar de modo nativo vídeo y audio en el navegador.
- ✓ Proporcionan una API de atributos, propiedades, métodos y eventos, que permite controlar tanto el vídeo como el audio mediante JavaScript.

Vídeo y audio

Propiedades de la API

- ***.paused***. Devuelve ***true*** o ***false*** si el recurso se está reproduciendo o no, respectivamente.
- ***.ended***. Devuelve ***true*** o ***false*** si la reproducción del recurso terminó correctamente, llegando al final, o no, parando antes por algún error.
- ***.currentTime***. Devuelve o establece el punto de reproducción del fichero de audio. El valor se expresa en segundos.
- ***.duration***. Devuelve la longitud del fichero de audio en segundos.
- ***.src***. La url en la que se encuentra el fichero de audio.
- ***.volume***. Devuelve o establece el volumen del fichero de audio.
- ***.muted***. Propiedad booleana que devuelve o establece si el audio está silenciado o no.
- ***.playBackRate***. Devuelve o establece la velocidad de reproducción. El valor que representa una velocidad normal es 1.0

Vídeo y audio

Métodos de la API

- ***.load()***. Carga y recarga el elemento de vídeo/audio a reproducir.
- ***.play()***. Empieza a reproducir el fichero de vídeo/audio.
- ***.pause()***. Pone en pausa la reproducción del fichero de vídeo/audio.
- ***.canPlayType()***. Comprueba si el navegador puede reproducir el fichero de vídeo/audio.

Eventos / manejadores de la API

- ***onplay***. Se dispara cuando se inicia la reproducción.
- ***onpause***. Se dispara cuando se pausa la reproducción.
- ***onended***. Se dispara al finalizar la reproducción.
- ***onvolumechange***. Se dispara al cambiar el volumen de la reproducción, es decir, cuando se modifica el valor del atributo *volume* o del atributo *muted*.
- ***ontimeupdate***. Se dispara cuando se modifica el punto actual de reproducción.

Vídeo y audio

Para el elemento <video>, además de los anteriores, se tienen también las siguientes ***propiedades*** y ***métodos***:

- ***.width***. Devuelve o establece el ancho del elemento <video> en píxeles.
- ***.videoWidth***. Devuelve el ancho del vídeo a reproducir.
- ***.height***. Devuelve o establece el alto del elemento <video> en píxeles.
- ***.videoHeight.*** Devuelve el alto del vídeo a reproducir.
- ***.requestFullScreen()***. Método que cambia el tamaño de vídeo a pantalla completa si el navegador lo soporta de forma nativa.
- ***.cancelFullScreen()***. Método que devuelve el vídeo a su tamaño normal tras haber estado en modo pantalla completa.

Gráficos en HTML: Canvas y SVG

Gráficos en HTML: Canvas y SVG

Canvas

- Presentado inicialmente por Apple, en 2004, permite crear, de forma dinámica, formas 2D e imágenes de tipo bitmap, en una página web.
- Se compone de una región para dibujar, definida como un elemento HTML **<canvas>** con atributos alto y ancho.
- Se utiliza JavaScript para acceder al elemento canvas y, mediante un conjunto de funciones de dibujo, crear en ella gráficos, animaciones, juegos y composición de imágenes.

Gráficos en HTML: Canvas y SVG

Canvas

- Primero se ha de obtener el contexto “2D” del elemento **<canvas>** del DOM del documento:

```
// Se accede al elemento <canvas> en el que dibujar
var cv = document.getElementById("miCanvas");
// Se obtiene el contexto 2D del canvas
var context = cv.getContext('2d');
```

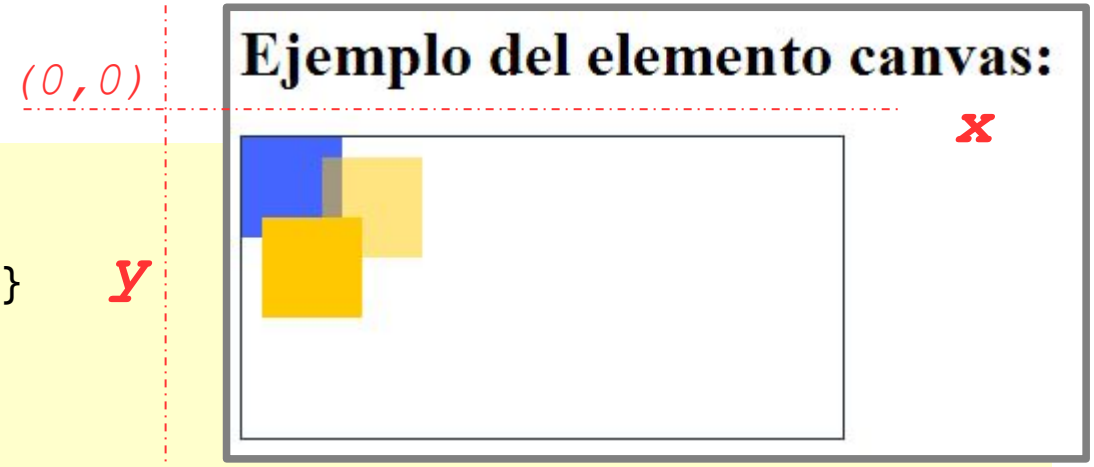
- Una vez obtenido el contexto, ya se puede dibujar en el canvas:

```
// Se dibuja un rectángulo en la posición (x=40, y=50),
// de tamaño ancho=100 y alto=75, de color rojo (#ff0000)
context.fillStyle = '#ff0000'; // Se establece el color
context.fillRect(40, 50, 100, 75); // Se dibuja el rect.
```

Gráficos en HTML: Canvas y SVG

Canvas

Ejemplo:




```
<head>
  <style type="text/css">
    canvas{ border:1px solid #234; }
  </style>
</head>
<body onload="demoCanvas();">
  <h1>Ejemplo del elemento canvas:</h1>
  <canvas id="miCanvas" width="300" height="150">
    Tu navegador no soporta canvas.
  </canvas>
</body>
```

```
function demoCanvas(){
  var cv  = document.getElementById("miCanvas"),
      ctx = cv.getContext('2d');
      ctx.fillStyle = '#4466ff';
      ctx.fillRect(0,0,50,50);
      ctx.fillStyle = 'rgb(255,200,0)';
      ctt.fillRect(10,40,50,50);
      ctx.fillStyle = 'rgba(255,200,0,0.5)';
      ctx.fillRect(40,10,50,50);
}
```

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Especificar estilo a utilizar:**
  - `.fillStyle [ = valor ]`. Devuelve, o establece a *valor*, el estilo utilizado para rellenar las formas.
  - `.strokeStyle [ = valor ]`. Devuelve, o establece a *valor*, el color utilizado para dibujar los bordes de las formas.
  - `.lineWidth [= valor]`. Devuelve, o establece a *valor*, el grosor utilizado para dibujar la línea de los bordes de las formas.
  - `.lineCap [= valor]`. Devuelve, o establece a *valor*, el estilo de los finales (extremos) de la línea. Valores posibles:
    - `butt`. Extremos planos de la línea. Valor por defecto.
    - `round`: Extremos redondeados.
    - `square`: Se añade un cuadrado a cada extremo de la línea.

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Especificar estilo a utilizar:**
  - `.lineJoin [= valor]`. Devuelve, o establece a *valor*, el estilo de la esquina a crear cuando se encuentran dos líneas. Valores posibles:
    - `bevel`. Crea una esquina biselada (punta recortada).
    - `round`: Crea una esquina redondeada.
    - `miter`: Crea una esquina de tipo inglete (acaba en punta).
  - `.miterLimit = valor`. Permite especificar el espacio, en unidades, destinado al inglete cuando `.lineJoin = miter`.
  - `.shadowOffsetX = valor`, `.shadowOffsetY = valor`. Permiten aplicar sombras verticales y horizontales cuando se dibujan rectángulos.
  - `.shadowBlur = valor`, `.shadowColor = valor`. Permiten configurar la sombra a aplicar cuando se dibujan rectángulos.
  - `.globalAlpha = valor`. Permite especificar el valor *alpha* (transparencia) a utilizar cuando se van a dibujar formas o imágenes. El valor va de 0.0 (transparente) a 1.0 (opaco).

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

➢ **Dibujar texto:**

- `.fillText(texto, x, y [, maxWidth ])`. Escribe el *texto* en la posición (***x***, ***y***), esquina inferior derecha, pintando el interior con el valor indicado por ***fillStyle***. Si se especifica *maxWidth* y el ancho del texto es mayor, éste es escalará a *maxWidth*.
- `.strokeText(texto, x, y [, maxWidth ])`. Escribe el *texto* en la posición (***x***, ***y***) pintando el borde con el valor indicado por strokeStyle. Si se especifica *maxWidth* y el ancho del texto es mayor, éste es escalará a *maxWidth*.
- `.font [= valor]`. Devuelve o establece el estilo de la fuente utilizado para dibujar texto. Similar a CSS.
- `.textAlign [= valor]`. Devuelve o establece la alineación horizontal del texto. Similar a CSS. Valores posibles: *left*, *right, center*.
- `.textBaseline [= valor]`. Devuelve o establece la alineación vertical del texto. Valores posibles: *alphabetic*, *middle*, *top*, *bottom*.

Gráficos en HTML: Canvas y SVG

Algunas propiedades y métodos del API de contexto 2D:

➢ **Dibujar texto:**

Ejemplo:

```
cv = document.querySelector('canvas');
ctx = cv.getContext('2d');

ctx.beginPath();
ctx.strokeStyle = '#a00';
ctx.moveTo(200,50);
ctx.lineTo(200,150);
ctx.stroke();
ctx.fillStyle = '#00a';
ctx.font = 'bold 18px sans-serif';
ctx.textAlign = 'center';
ctx.fillText('Hola Mundo!!',200,100);
```

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

➢ **Dibujar formas rectangulares:**

- `.fillRect(x, y, w, h)`. Dibuja un rectángulo en (***x***, ***y***), con un ancho de ***w*** y un alto de ***h***, pintando interior y borde con el valor indicado por **`fillStyle`**.
- `.strokeRect(x, y, w, h)`. Dibuja un rectángulo en (***x***, ***y***) con un ancho de ***w*** y un alto de ***h***, pintando sólo el borde con el valor indicado por **`strokeStyle`** y **`lineWidth`**.
- `.rect(x, y, w, h)`. Funciona igual que **`.strokeRect()`**, pero para que se dibuje en el canvas es necesario invocar el método **`.stroke()`**.

Ejemplo:

```
ctx.lineWidth = 2;
ctx.strokeStyle = '#a00';
ctx.rect(50,30,100,50);
ctx.stroke();
```

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Dibujar líneas rectas:**
  - `.moveTo(x, y)`. Mueve el pincel a la posición (*x*, *y*) del canvas.
  - `.lineTo(x, y)`. Dibuja una línea desde la posición actual del pincel hasta la posición (*x*, *y*) utilizando el estilo de línea indicado mediante **`strokeStyle`** y **`lineWidth`**.

El uso de estos dos métodos crea un *path* de trazado que no será visible hasta que se invoque el método **`stroke`**`()`.

Ejemplo:

```
...
ctx.strokeStyle = '#a00'; // color a utilizar
ctx.lineWidth = 2; // grosor de la línea
ctx.moveTo(50,60);
ctx.lineTo(110,130);
ctx.lineTo(120,90);
ctx.stroke(); // se dibuja el pathcon
```

Gráficos en HTML: Canvas y SVG

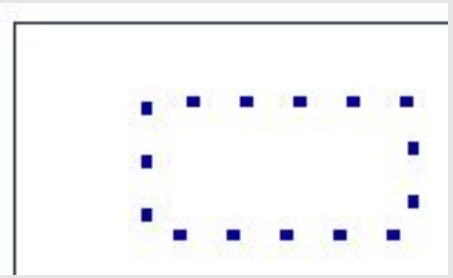
Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Dibujar líneas rectas discontinuas:**
  - `.setLineDash(`*`segmentos`*`)`. Permite configurar cómo será la línea discontinua. Por ejemplo, un valor de *`segmentos`* de **`[5, 15]`** implicaría segmentos de línea de 5 píxeles y espacios en blanco de 15 entre ellos.
  - `.lineDashOffset = `*`valor`*. Permite indicar a partir de qué unidad de la línea empezar a pintarla. El valor por defecto es `0`.

Ejemplo:

```
ctx.strokeStyle = '#00a';
ctx.lineWidth = 4;
ctx.setLineDash([5,15]);
ctx.lineDashOffset = 5;
ctx.strokeRect(50,30,100,50);
```



Gráficos en HTML: Canvas y SVG

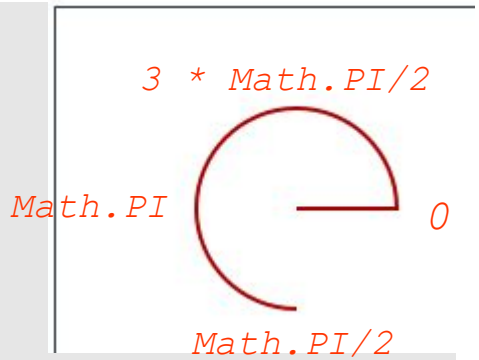
Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Dibujar círculos:**
  - `.arc(x, y, r, angInicio, angFin, sentido)`. Dibuja la circunferencia de un círculo de radio ***r***, centrado en (***x***, ***y***). Empieza a dibujar la circunferencia en el ángulo ***angInicio*** y la termina en ***angFin,*** siguiendo el sentido de las agujas del reloj si el valor de ***sentido*** es *false*, o en sentido contrario si su valor es *true*. El valor de los ángulos se expresa en radianes.

Ejemplo:

```
ctx.beginPath();
ctx.lineWidth = 2;
ctx.strokeStyle = '#a00';
ctx.arc(120, 100, 50, Math.PI/2, 2*Math.PI, false);
ctx.lineTo(120,100);
ctx.stroke();
```



Gráficos en HTML: Canvas y SVG

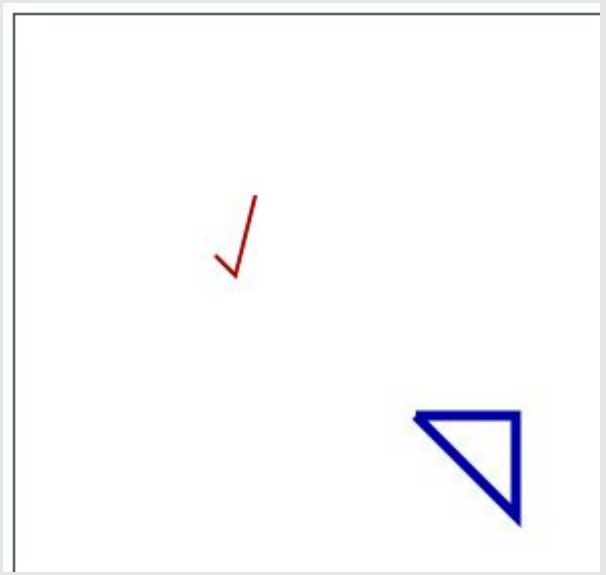
Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Agrupar operaciones:**
  - `.beginPath()`. Permite crear un nuevo *path* sobre el que aplicar un estilo distinto del utilizado hasta el momento.

Ejemplo:

```
ctx.strokeStyle = '#a00'; // color a utilizar
ctx.lineWidth = 2; // grosor de la línea
ctx.moveTo(100,120);
ctx.lineTo(110,130);
ctx.lineTo(120,90);
ctx.stroke(); // se dibuja el path
ctx.beginPath(); // comienza la nueva figura
ctx.strokeStyle = '#00a'; // color a utilizar
ctx.lineWidth = 5; // grosor de la línea
ctx.moveTo(200,200);
ctx.lineTo(250,200);
ctx.lineTo(250,250);
ctx.lineTo(200,200);
ctx.stroke(); // se dibuja el path
```



Gráficos en HTML: Canvas y SVG

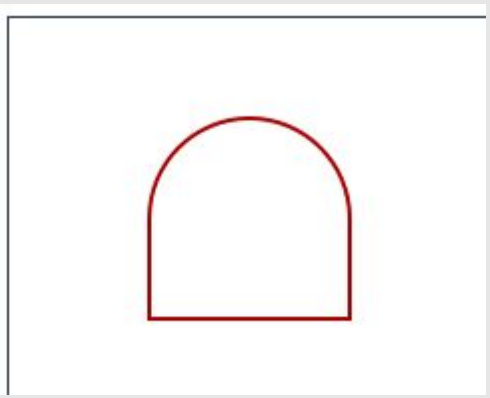
Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Agrupar operaciones:**
  - `.closePath()`. Cierra un *path* uniendo el primer y último punto dibujados, mediante una línea recta.

Ejemplo:

```
...
ctx.beginPath();
ctx.lineWidth = 2;
ctx.strokeStyle = '#a00';
ctx.arc(120,100,50, 0, Math.PI, true);
ctx.lineTo(70,150);
ctx.lineTo(170,150);
ctx.closePath(); // se cierra el path
ctx.stroke(); // se dibuja el path
```



Gráficos en HTML: Canvas y SVG

Canvas

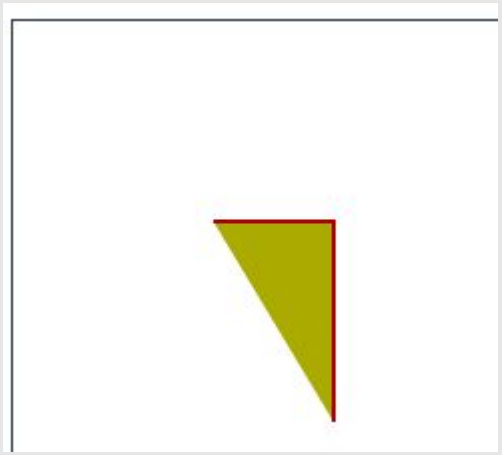
Algunas propiedades y métodos del API de contexto 2D:

- **Relleno del *path*:**
  - `.fill()`. Rellena el interior del path dibujado con el color de relleno que esté activo en el canvas.

Ejemplo:

```
ctx.strokeStyle = '#a00';
ctx.lineWidth = 2;
ctx.moveTo(100,100);
ctx.lineTo(160,100);
ctx.lineTo(160,200);
ctx.fillStyle = '#aa0';
ctx.fill();

ctx.stroke();
```



Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Borrado del canvas:**
  - `.clearRect(x, y, ancho, alto)`. Borra el área del canvas delimitada por el rectángulo con esquina superior en (x,y) y de ancho y alto indicado.

Ejemplo:

```
ctx.beginPath();
ctx.lineWidth = 2;
ctx.strokeStyle = '#a00';
ctx.arc(120,100,50, 0, Math.PI, true);
ctx.lineTo(70,150);
ctx.lineTo(170,150);
ctx.stroke();

ctx.clearRect( 50, 20, 100, 80);
```

Gráficos en HTML: Canvas y SVG

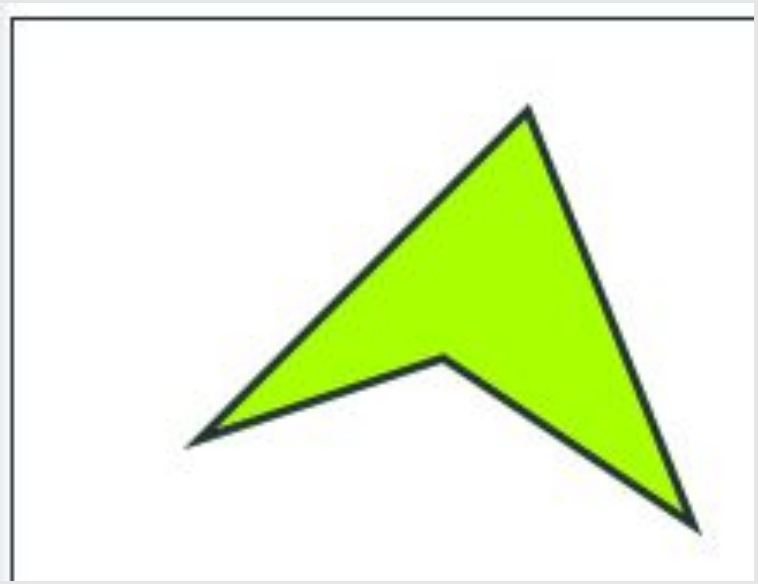
Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Recorte del canvas:**
  - `.clip()`. Permite hacer que el path que se esté dibujando se utilice como región de recorte.

Ejemplo:

```
ctx.beginPath();
ctx.lineWidth = 5;
ctx.strokeStyle = '#234';
ctx.moveTo(50,125);
ctx.lineTo(150,25);
ctx.lineTo(200,150);
ctx.lineTo(125,100);
ctx.lineTo(50,125);
ctx.clip();
ctx.fillStyle = '#af0';
ctx.fillRect(0,0,400,400);
ctx.stroke();
```



Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Guardar y restaurar el estado del canvas:**
  - `.save()`. Permite guardar en una pila de estados el estado del canvas en lo referente a:
    - Transformaciones: traslaciones, rotaciones, escalados, etc.
    - La región de recorte (*clipping region*).
    - Los valores de, entre otros, los siguientes atributos: `strokeStyle`, `fillStyle`, `globalAlpha`, `lineWidth`, `lineCap`, `lineJoin`, miterLimit, `shadowOffsetX`, `shadowOffsetY`, `shadowBlur`, `shadowColor`, `globalCompositeOperation`, `font`, `textAlign`, `textBaseline`.
  - `.restore()`. Permite recuperar de la pila de estados el último estado guardado, aplicando dicha configuración al canvas. El estado recuperado se elimina de la pila de estados.

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Transformaciones:** Se aplican al canvas y afectan a todo lo dibujado a partir del momento de su aplicación. No se aplican a lo que ya hubiera dibujado en el canvas. Las transformaciones básicas son las siguientes:
  - `.scale(x, y)`. Aplica una transformación de escalado. Los valores de ***x*** e ***y*** son valores decimales que representan el factor de escalado aplicado en el eje horizontal y vertical, respectivamente. Por ejemplo, 0.5 representaría escalar a un 50% del tamaño original, mientras que 2.0 escalaría a un 200% del tamaño original.
  - `.rotate(`*ángulo*`)`. Aplica una transformación de rotación indicada por *ángulo*, expresado en radianes y en el sentido de las agujas del reloj.
  - `.translate(x, y)`. Aplica una transformación de traslación al origen de coordenadas del canvas, inicialmente en la esquina superior izquierda.

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

➢ **Transformaciones:**

- `.transform(`*`fEscX, fdY, fdX, fEscY, tX, tY`*`)`. Aplica la siguiente matriz de transformación a la transformación que tenga aplicada el canvas:

fEscX	fdX	tX
fdY	fEscY	tY
0	0	1

  - **fEscX**, **fEscY**. Factores de escalado ***x*** e ***y***.
  - **fdX**, **fdY**. Factores de "estiramiento" (cizalla) ***x*** e ***y***.
  - **tX**, **tY**. Factores de translación ***x*** e ***y***.

- `.setTransform(`*`fEscX, fdY, fdX, fEscY, tX, tY`*`)`. Resetea la actual transformación a la matriz identidad y aplica una nueva transformación llamando a `.transform(`*`fEscX,fdY,fdX,fEscY,tX,tY`*`)`. Por ejemplo, `.setTransform(1, 0, 0, 1, 0, 0)` resetea a la matriz identidad.
- `.resetTransform()`. Resetea la actual transformación a la matriz identidad.

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Transformaciones:**

Ejemplo:

```
var canvas = document.getElementById('micanvas');
var c, ctx = canvas.getContext('2d');
var sin = Math.sin(Math.PI/6);
var cos = Math.cos(Math.PI/6);

ctx.translate(200, 200);

for (var i=0; i <= 12; i++) {
  c = Math.floor(255 / 12 * i);
  ctx.fillStyle = 'rgb(' + c + ',' + c + ',' + c + ')';
  ctx.fillRect(0, 0, 100, 100);
  ctx.transform(cos, sin, -sin, cos, 0, 0);
}

ctx.setTransform(-1, 0, 0, 1, 200, 200);
ctx.fillStyle = 'rgba(100, 100, 255, 0.5)';
ctx.fillRect(50, 50, 100, 100);
```

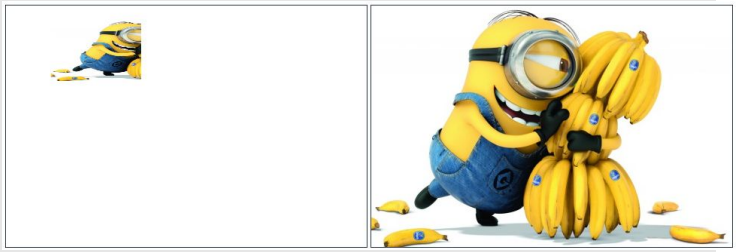
Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Dibujar imágenes:**
  - `.drawImage(imagen, x, y, [ancho, alto])`. Dibuja el objeto `imagen` en la posición `(x,y)`, pudiendo cambiar las dimensiones de destino mediante `ancho` y `alto`.
  - `.drawImage(imagen, x, y, ancho, alto, dx, dy, dAncho, dAlto)`. Copia la región del objeto `imagen` que empieza en la posición `(x,y)`, cuyas dimensiones son `ancho` y `alto`, y la dibuja en la posición (dx, dy) escalada al ancho y el alto indicados por `dAncho` y `dAlto`, respectivamente.

```
var cv = document.querySelectorAll('canvas')[0];
var cv2 = document.querySelectorAll('canvas')[1];
var ctx = cv.getContext('2d');
var ctx2 = cv.getContext('2d');
var img = new Image();
img.onload = function(){
  ctx2.drawImage(img,0,0, cv2.width,cv2.height);
  ctx.drawImage(img,0,0,cv.width/2,cv.height,60,20,120,80);
};
img.src = './img01.jpg';
```



Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

➢ **Obtener la imagen del canvas para trabajar a nivel de bit:**

- **.getImageData**(*x*, *y*, *ancho*, *alto*)**.** Devuelve un objeto **ImgData** que contiene el vector de bytes que representa a los píxeles de la región del canvas cuya esquina superior izquierda es (x,y) y tiene el ancho y alto especificado.

  El objeto **ImageData** utiliza 4 bytes para cada pixel, siendo cada uno de ellos un valor entre 0 y 255 para cada una de las componentes de rojo, azul, verde y transparencia, respectivamente. Para acceder a los puntos de la imagen se utiliza su propiedad **data**, que es un vector cuya longitud es el resultado de multiplicar 4 por el número de píxeles de la imagen.

- **.putImageData**(*imgData*, *x*, *y* [, *xCv*, *yCv*, *anchoCv*, *altoCv*])**.** Dibuja un objeto *imgData* en el canvas, en la posición (*x*,*y*). Si se indican los 4 últimos parámetros optativos, dibujará únicamente la región de *imgData* delimitada por el rectángulo (*xCv*, *yCv*, *xCv* + *anchoCv*, *yCv* + *altoCv*).

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

➢ **Obtener la imagen del canvas para trabajar a nivel de bit:**

Ejemplo:

```
...
var cv1 = document.querySelector('#cv1');
var ctx1 = cv1.getContext('2d');
var imgData = ctx1.getImgData(20, 50, 200, 300);
var cv2 = document.querySelector('#cv2');
var ctx2 = cv2.getContext('2d');
ctx2.putImageData(imgData, 0, 0);
```

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Guardar la imagen del canvas:**
  - **.toDataURL**([*tipoImagen*] [, *calidad*])**.** Permite obtener la imagen del canvas. Para poder guardarla en disco se necesita la acción del usuario, ya que por motivos de seguridad no se puede hacer con JavaScript.

    Los dos parámetros son opcionales. El primero indica el formato de imagen al que copiar la imagen del canvas y los valores admitidos son: **image/jpeg**, **image/webp** e **image/png**. El segundo parámetro permite indicar la calidad de la imagen y admite valores entre 0.0 y 1.0. Si se utiliza el formato *image/png*, el segundo argumento no será tenido en cuenta.

Ejemplo:

```
var cv = document.querySelector('canvas');
var img = cv.toDataURL(); // el formato por defecto es image/jpg

document.querySelector('#res').src = img; // muestra la imagen en un <img>
window.open(img); // muestra la imagen en otra ventana
```

Gráficos en HTML: Canvas y SVG

Canvas

Algunas propiedades y métodos del API de contexto 2D:

- **Animaciones:**
  - `long requestAnimationFrame(callback)`. Proporcionado por el objeto `Window`, permite realizar animaciones de manera más eficiente, en comparación con setTimeout() y setInterval().
    - Permite al navegador optimizar las animaciones, haciéndolas más suaves, y hacer más eficiente el uso de los recursos.
    - Las animaciones se detienen cuando la pestaña (o la ventana) del navegador no es visible. Esto significa un ahorro notable del uso de memoria, de CPU, de GPU y, por consiguiente, de batería.
  - `void cancelAnimationFrame(idAnimación)`. Permite detener la animación lanzada con `requestAnimationFrame()`.

SVG: Scalable Vector Graphics

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

- ✓ Lenguaje basado en XML que permite describir gráficos 2D, cuya primera especificación es de 1999.
- ✓ Permite tres tipos de gráficos: formas vectoriales, imágenes y texto.
- ✓ Permite escalar los gráficos sin perder calidad.
- ✓ Se ubican dentro del documento HTML como un elemento más del DOM, por lo que se les puede aplicar estilo, transformaciones; asignar manejadores de eventos, etc.

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Cómo usar gráficos SVG en HTML:

- Como un elemento más del DOM

  Se utiliza el elemento **<svg>** de HTML que contendrá todos los elementos svg como nodos hijo.

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480"
     height="320">

    ...

</svg>
```

- Es necesario indicar el espacio de nombres para documentos de tipo image/svg+xml:

  `xmlns="http://www.w3.org/2000/svg"`

- Cuando se utiliza como elemento de un documento HTML5, no es necesario.

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Cómo usar gráficos SVG en HTML:

- Como un elemento más del DOM

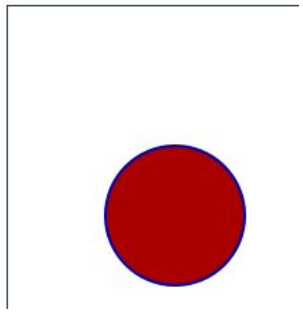
  Existen algunas diferencias a la hora de indicar el estilo de los elementos svg con respecto al CSS:
  - Para definir el color de fondo de una forma no se utiliza la propiedad CSS *background-color*, sino el atributo **fill**.
  - Para definir el color y grosor del borde de la forma se utilizan los atributos **stroke** y **stroke-width**, respectivamente.

```
<svg xmlns="http://www.w3.org/2000/svg"
  width="480"
  height="320">
    <circle class="fondoRojo bordeAzul"
      cx="120" cy="150" r="50" />
</svg>
```

html

```
.fondoRojo{
    fill:#a00;
}
.bordeAzul{
    stroke: #00a;
    stroke-width:2px;
}
```

css



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics)*

Cómo usar gráficos SVG en HTML:

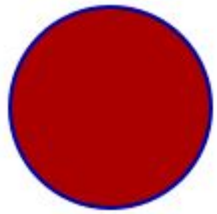
- Como un elemento más del DOM
  - También existe la posibilidad de guardar el gráfico svg en un fichero de imagen externo, con extensión *.svg, que puede ser incorporado al documento html como cualquier otro fichero de imagen. En este caso, las reglas de estilo css deben incluirse en el mismo fichero svg.

html

```
...
<img src="circulorojo.svg" alt="Círculo rojo" />
...
```

circulorojo.svg

```
<svg xmlns="http://www.w3.org/2000/svg" width="104" height="104">
    <style type="text/css">
        .circuloRojo{ fill:#a00; stroke: #00a; stroke-width:2px; }
    </style>
    <circle class="circuloRojo" cx="52" cy="52" r="50"/>
</svg>
```



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

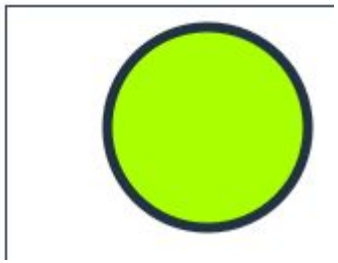
Formas básicas:

- **<circle/>**. Dibuja un círculo.

  Atributos:
  - **cx**, **cy**. Posición en la que se ubicará el centro del círculo.
  - **r**. Radio del círculo.

html

```
 ...
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <circle
      cx="100" cy="60"
      r="50"
      fill="#af0"
      stroke="#234"
      stroke-width="5px"
  />
</svg>
 ...
```



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

- **<ellipse/>**. Dibuja una elipse.

  Atributos:
  - **cx**, **cy**. Posición en la que se ubicará el centro de la elipse.
  - **rx**, **ry**. Radios de la elipse.

html

```
 ...
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <ellipse
      cx="100" cy="60"
      rx="50" ry="25"
      fill="#3a0"
      stroke="#234"
      stroke-width="5px"
  />
</svg>
 ...
```

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

- **<line/>**. Dibuja una recta.

  Atributos:
  - **x1**, **y1**. Coordenadas del punto de inicio de la recta.
  - **x2, y2**. Coordenadas del punto final de la recta.

html

```
 ...
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <line
      x1="100" y1="25"
      x2="150" y2="75"
      stroke="#a00"
      stroke-width="5px"
  />
</svg>
 ...
```

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

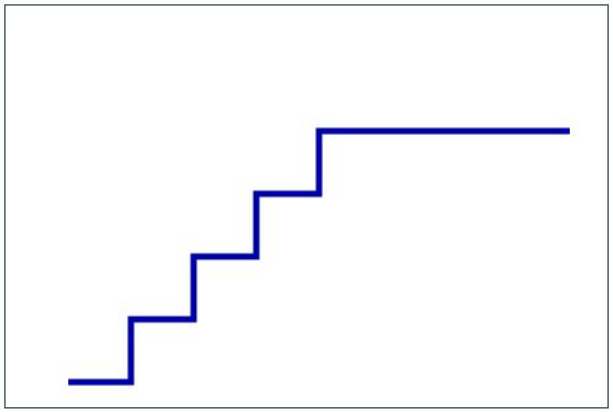
- **<polyline/>**. Permite definir un conjunto de líneas rectas (segmentos) conectadas.

  Atributos:

  - **points**. Lista ordenada de pares de coordenadas x,y de cada uno de los puntos que serán unidos mediante rectas. Los pares están separados entre sí por espacios en blanco.

html

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <polyline
     points="50,300 100,300 100,250 150,250 150,200
                200,200 200,150 250,150 250,100
450,100"
      stroke="#00a"
      stroke-width="5px"
  />
</svg>
```



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

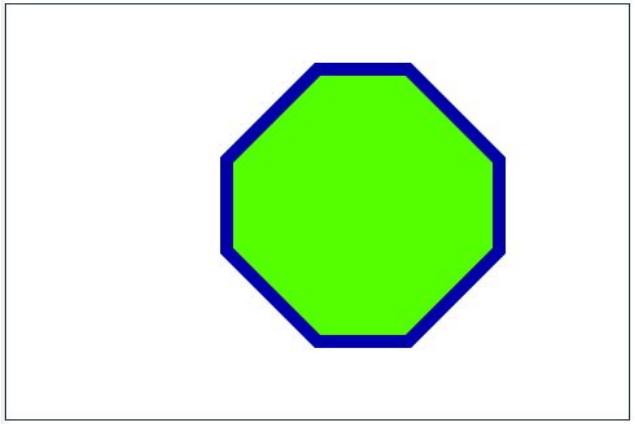
- **<polygon/>**. Permite definir una forma cerrada consistente en un conjunto de líneas rectas (segmentos) conectadas.

  Atributos:

  - **points**. Lista ordenada de pares de coordenadas x,y de cada uno de los puntos que serán unidos mediante rectas. Los pares están separados entre sí por espacios en blanco. El último punto de la lista lo une con el primero.

html

```
 <svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <polygon
     points="240,50 170,120 170,190 240,260
             310,260 380,190 380,120 310,50"
      stroke="#00a" stroke-width="5px"
fill="#5f0"/>
</svg>
```



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

- **<rect/>**. Dibuja un rectángulo.

  Atributos:
  - **x**, **y**. Posición en la que se ubicará la esquina superior izquierda del rectángulo.
  - **width**, **height**. Dimensiones del rectángulo.

html

```
 ...
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <rect
      x="50" y="20"
      width="120" height="100"
      fill="#00a"
  />
</svg>
 ...
```

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

Formas básicas:

- **<path/>**. Se utiliza para definir un *path* de operaciones de dibujo. Las operaciones disponibles son las siguientes:

M x y => mover al punto (x,y) **m dx dy** => desplazamiento relativo (dx,dy)	**C x1 y1, x2 y2, x y** (o **c dx1 dy1, dx2 dy2, dx dy**) => curva cúbica
L x y => línea al punto (x,y) **l dx dy** => línea al punto con posición relativa (dx,dy)	**S x2 y2, x y** (o **s dx2 dy2, dx dy**) => permite enlazar con una curva cúbica anterior
H x => línea horizontal hasta el punto x **h dx** => línea horizontal de dx puntos	**Q x1 y1, x y** (o **q dx1 dy1, dx dy**) => curva cuadrática
V y => línea vertical hasta el punto y **v dy** => línea vertical de dy puntos	**T x y** (o **t dx dy**) => permite enlazar con una curva cuadrática anterior
A rx ry x-axis-rotation large-arc-flag sweep-flag x y (o **a rx ry x-axis-rotation large-arc-flag sweep-flag dx dy**) => círculos o elipses	**Z** (o **z**) => cierra el path (une último y primer punto.

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

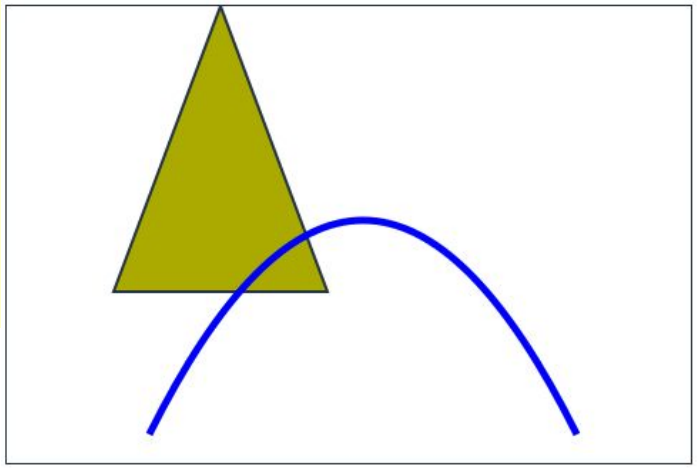
Formas básicas:

- **<path/>**.

  Ejemplo:

html

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <path
    fill="#aa0" stroke="#234" stroke-width="2px"
    d="M150 0 L75 200 L225 200 Z" />
  <path
    stroke="#00a"
    d="M 100 300 q 150 -300 300 0" />
</svg>
```



Gráficos en HTML: Canvas y SVG

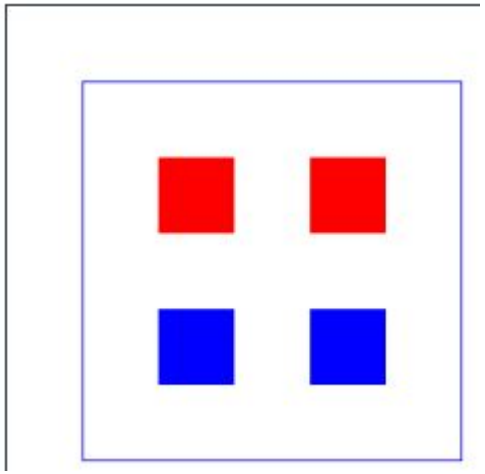
SVG (*Scalable Vector Graphics*)

Otros elementos SVG:

- **<g></g>**. No se trata de una forma básica en sí. Es un elemento contenedor que permite agrupar formas básicas o, incluso, otras agrupaciones.

html

```
 ...
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <g id="group1" fill="red">
    <rect x="2cm" y="1cm" width="1cm" height="1cm"/>
    <rect x="4cm" y="1cm" width="1cm" height="1cm"/>
  </g>
  <g id="group2" fill="blue">
    <rect x="2cm" y="3cm" width="1cm" height="1cm"/>
    <rect x="4cm" y="3cm" width="1cm" height="1cm"/>
  </g>
  <rect x=".01cm" y=".01cm"
        width="4.98cm" height="4.98cm"
        fill="none" stroke="blue" stroke-width=".02cm"/>
</svg>
```



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics)*

Otros elementos SVG:

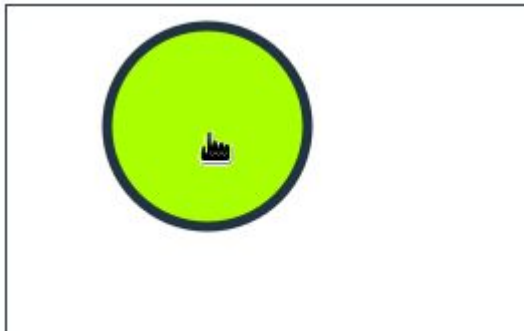
- **<a>**. Permite crear hiperenlaces. Su funcionamiento es el mismo que el elemento **<a>** en html.

Atributos:

- **xlink:href**. URL asociada al hiperenlace.

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <a xlink:href="http://www.ua.es">
    <circle
      cx="100" cy="60"
      r="50"
      fill="#af0"
      stroke="#234"
      stroke-width="5px" />
  </a>
</svg>
```

html



Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics)*

Otros elementos SVG:

- **<image></image>**. Permite incluir una imagen de un fichero externo en el área indicada mediante sus atributos.

Atributos:

- **x**, **y**. Posición en la que se ubicará la esquina superior izquierda del área que contendrá la imagen.
- **width**, **height**. Dimensiones del área que contendrá la imagen.
- **xlink:href**. Path del fichero con la imagen a mostrar.

html

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <image x="50" y="50" width="120px" height="80px"
         xlink:href="./img01.jpg">
    <title>Imagen de muestra</title>
  </image>
</svg>
```

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics)*

Otros elementos SVG:

- **<text></text>**. Define un elemento gráfico consistente en un texto.

Atributos:

- **x**, **y**. Posición en la que se ubicará la esquina inferior izquierda del primer carácter del texto.

html

```
<svg xmlns="http://www.w3.org/2000/svg"
     width="480" height="320">
  <text class="texto" x="50" y="30">
      Hola mundo!!!
  </text>
</svg>
```

css

```
.texto{
  font-family:verdana;
  font-size:22px;
  fill:#00a;
}
```

Hola mundo!!!

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics)*

Transformaciones sobre elementos SVG:

Se pueden aplicar transformaciones a elementos o grupos de elementos mediante el atributo **transform**. Las transformaciones posibles son:

- ✓ **matrix**(a,b,c,d,e,f). Similar a la transformación correspondiente para canvas.
- ✓ **translate**(tx [, ty]). Translación en x e y. Si no se indica ***ty***, se asume como cero.
- ✓ **scale**(sx [, sy]). Escalado en x e y. Si no se indica el valor de ***sy***, se asume el mismo valor que ***sx***.
- ✓ **rotate**(ángulo [, cx, cy]). Rotación por el ángulo (en grados) indicado tomando como origen de la transformación el punto indicado (***cx***,***cy***). Si no se especifican valores para ***cx*** y ***cy***, se utilizan, se utiliza el origen de coordenadas (0,0).

Gráficos en HTML: Canvas y SVG

SVG (*Scalable Vector Graphics*)

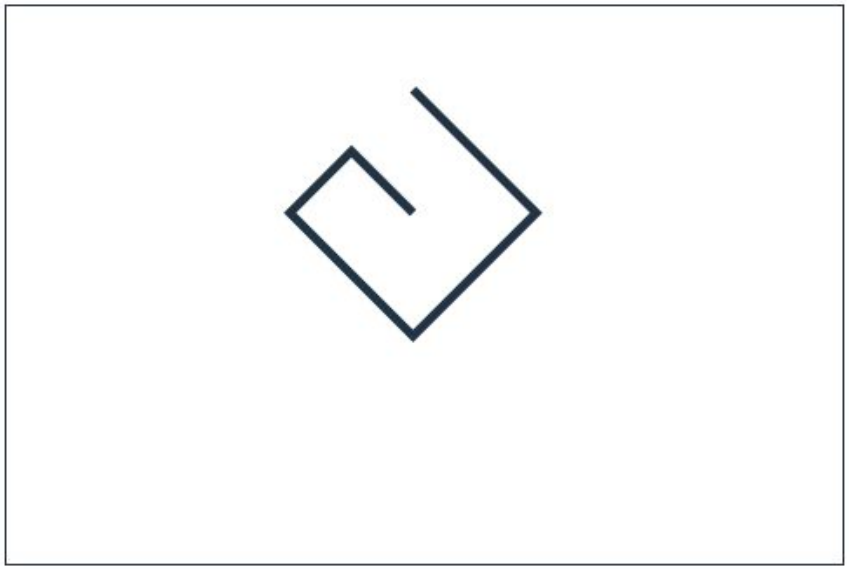
Transformaciones sobre elementos SVG:

- ✓ **skewX**(ángulo). Transformación de cizalla en el eje X.
- ✓ **skewY**(ángulo). Transformación de cizalla en el eje Y.

Se puede aplicar más de una operación en la misma transformación.

html

```
...
 <polyline
   fill="none"
   stroke="#234"
   stroke-width="5px"
   points="50,50 150,50 150,150
           50,150 50,100 100,100"
   transform="translate(50,100)
              rotate(45, 240, 160)"
 />
...
```



Drag & drop

Drag & drop

Permite arrastrar y soltar cualquier elemento de una página web.

Pasos para proporcionar la característica Drag & Drop:

- ✓ Añadir al elemento arrastrable el atributo **draggable**, así como el código necesario para el evento **dragStart** del elemento. La propiedad **dataTransfer**, que debe establecerse en el evento *dragstart*, se utiliza para almacenar los datos a arrastrar.
- ✓ Añadir al elemento receptor el código necesario para los eventos **dragOver** y **drop**. En este código es conveniente cancelar la acción por defecto del navegador incluyendo la instrucción "*return false;*", o bien mediante el método *preventDefault()* del objeto. También es conveniente cancelar la propagación del evento mediante el método *stopPropagation()* del objeto. En el evento *drop* es donde se procesa la propiedad *dataTransfer*.
- ✓ Se puede añadir código a otros eventos como *dragEnter*, *dragOver*, *dragLeave* y *dragEnd*, para enriquecer la experiencia del usuario.

Drag & drop

Propiedad dataTransfer:

- **e**.**dataTransfer**.**setData**(***formatoDatos***, ***datos***)

  El método setData() se utiliza en el evento dragstart para realizar la carga de datos que se envían en la acción de arrastre.

  - **e**: Evento *Drag & Drop*.
  - ***formatoDatos***: Formato de los datos que se van a enviar, por ejemplo, 'text/html' cuando se trata de elementos del DOM del documento.
  - ***datos***: Los datos que se envían, por ejemplo el html del elemento del DOM del documento que se envía.

Ejemplo: Código para el evento dragstart

```
// obj es el elemento del DOM que queremos hacer "arrastrable".
obj.ondragstart = function(e){
  // this es el elemento del DOM origen de la operación drag: obj.
  e.dataTransfer.effectAllowed = 'move'; // tipo de arrastre permitido
  e.dataTransfer.setData('text/html', this.innerHTML); // carga de la información
}
```

Drag & drop

Propiedad dataTransfer:

- e.dataTransfer.getData(*formatoDatos*)

  El método getData() se utiliza en el evento drop para recoger los datos que se envían en la acción de arrastre.

  - e: Evento *Drag & Drop*.
  - *formatoDatos*: Formato de los datos que se van a recoger. Valores permitidos: 'text/plain', 'text/html', 'text/uri-list'.

Ejemplo: Código para el evento dragstart

```
// obj es el elemento del DOM que queremos que reciba la acción drop.
obj.ondrop = function(e){
  if (e.stopPropagation)
    e.stopPropagation(); // Evita la propagación del evento
  // this es el elemento del DOM destino de la operación drop: obj.
  this.innerHTML = e.dataTransfer.getData('text/html'); // recoge la información
                                                        // y la usa como su innerHTML
  return false; // cancela la acción por defecto
}
```

Drag & drop

Propiedad dataTransfer:

DataTransfer también proporciona una serie de propiedades que pueden mejorar la experiencia de usuario:

✓ **e.dataTransfer.effectAllowed**

Restringe el tipo de arrastre permitido. Se utiliza en combinación con la propiedad *dropEffect* y los eventos *dragEnter* y *dragOver*. Los valores que admite son: *none*, *copy*, *copyLink*, *copyMove*, *link*, *linkMove*, *move*, *all* y *uninitialized*. Se establece en el evento *dragstart*.

✓ **e.dataTransfer.dropEffect**

Controla la información que recibe el usuario durante los eventos *dragEnter* y *dragOver*, cambiando el cursor del ratón. Los valores que admite son: *none*, *copy*, *link* y *move*.

✓ **e.dataTransfer.setDragImage(***elemento_img*, *x*, *y***)**

Permite utilizar una imagen personalizada como icono de arrastre, en lugar de la imagen por defecto que utiliza el navegador. *elemento_img* debe ser un elemento *img* o *canvas* existente en el DOM del documento.

Drag & drop

Ejemplo completo:

Código JavaScript

```
var elemOrigen = ""; // Elemento origen del drag
function prepararDnD(){
  var cols = document.querySelectorAll('.columna');
  for(var i=0; i<cols.length; i++){
    cols[i].ondragstart = function(e){
      elemOrigen = this; // Se guarda referencia al objeto que inicia el drag
      e.dataTransfer.setData('text/html', this.innerHTML);
    }
    cols[i].ondragover = function(e){
      if (e.stopPropagation) e.stopPropagation(); // Evita la propagación
      return false; // Evita el acción asociada por defecto
    }
    cols[i].ondrop = function(e){
      if (e.stopPropagation) e.stopPropagation(); // Evita la propagación
      elemOrigen.innerHTML = this.innerHTML; // Se actualiza el elemento origen
      this.innerHTML = e.dataTransfer.getData('text/html'); // recoge el html
      return false; // Evita el acción asociada por defecto
    }
  }
}
```

Código HTML

```
<body onload="prepararDnD();">
  <div class="columna" draggable="true"><header>A</header></div>
  <div class="columna" draggable="true"><header>B</header></div>
  <div class="columna" draggable="true"><header>C</header></div>
</body>
```

Drag & drop

Drag & drop de ficheros

Este API también permite arrastrar y soltar ficheros entre el ordenador y el navegador y viceversa.

➢ **Arrastrar ficheros del escritorio al navegador**

Para permitir arrastrar ficheros sólo es necesario hacer un pequeño cambio en el código asociado al evento *drop.*

```
// obj es el elemento del DOM que queremos que reciba la acción drop.
obj.ondrop = function(e){
  if (e.stopPropagation) e.stopPropagation(); // Evita la propagación del evento
  e.preventDefault(); // Cancela la acción por defecto. Alternativa a: return false;
  var ficheros = e.dataTransfer.files;
  console.log("Total ficheros: " + ficheros.length);
  for(var i = 0; i < ficheros.length; i++) {
    // Lee los objetos de tipo File del vector ficheros.
    console.log( "Fichero " + i + ": " + ficheros[i].name); // escribe el nombre
    // ...
  }
}
// también hay que asociar código al evento dragover para que cancele la acción por defecto
obj.ondragover = function(e){
  if (e.stopPropagation) e.stopPropagation(); // Evita la propagación
  e.preventDefault(); // Cancela la acción por defecto. Alternativa a: return false;
}
```

Web Workers

Web Workers

- Permiten ejecutar scripts en segundo plano en una página web, sin afectar al funcionamiento del resto de scripts de la página.
- No es conveniente abusar de ellos, ya que su consumo de recursos es relativamente importante.
- Se utilizan para realizar tareas de larga duración, con alto coste de procesamiento y memoria.
- No pueden acceder al DOM del documento desde el que se llama.

Web Workers

Uso:

- ✓ **Crear el objeto *worker*:**

```
var worker = new Worker(url_fichero_código.js)
```

- ✓ Para **iniciar el *worker*** basta con invocar su método **postMessage**. Si se necesita pasar información al *worker*, este método puede aceptar como único argumento un *string* o un *objeto JSON*.

```
worker.postMessage([argumento]);
```

El ***worker*** también utiliza el método **postMessage** para envíar mensajes.

- ✓ Para recoger los mensajes enviados con **postMessage**, es necesario asociar código al evento **onmessage**.

```
worker.onmessage = function(e){
// e.data es la información enviada con postMessage
}
```

- ✓ Se puede forzar la finalización del ***worker*** antes de tiempo:

```
worker.terminate( datos )
```

Web Workers

Ejemplo:

```
var worker = new Worker('doWork1.js'); // Se crea el worker
worker.onmessage = function(e){ // Se asigna el handler para el evento message
  document.getElementById('resultado').textContent = e.data.res;
};
function multiplicar(){
  var op1 = document.getElementById("op1").value;
  var op2 = document.getElementById("op2").value;
  worker.postMessage({'cmd': 'multiplicar', 'op1':op1, 'op2':op2});
}
function sumar(){
  var op1 = document.getElementById("op1").value;
  var op2 = document.getElementById("op2").value;
  worker.postMessage({'cmd': 'sumar', 'op1':op1, 'op2':op2});
}
function parar(){
  worker.postMessage({'cmd': 'parar'});
}
```

Código JavaScript

```
<input type="text" id="op1" placeholder="op1"><br>
<input type="text" id="op2" placeholder="op2"><br>
<button onclick="multiplicar()">Multiplicar</button>
<button onclick="sumar()">Sumar</button>
<button onclick="parar()">Parar worker</button><br>
<span>Resultado:</span><output id="resultado"></output>
```

Código HTML

Web Workers

Ejemplo:

Código **doWork1.js**

```
self.onmessage = function(e){
  var data = e.data;
  switch (data.cmd) {
    case 'multiplicar':
        var op1 = data.op1, op2 = data.op2;
        var r = parseInt(op1) * parseInt(op2);
        self.postMessage( {'op':true, 'res': r} );
      break;
    case 'sumar':
        var op1 = data.op1, op2 = data.op2;
        sumar(op1, op2);
      break;
    case 'parar':
        self.postMessage( {'op':true, 'res': 'Worker detenido'} );
        self.close(); // Termina el worker.
      break;
    default:
        self.postMessage( {'op':true, 'res': r} );
  };
};
function sumar(op1, op2){ // Se pueden crear y utilizar funciones dentro del worker.
  var r = parseInt(op1) + parseInt(op2);
  self.postMessage( {'op':true, 'res': r} );
}
```

Geolocalización

Geolocalización

- ✓ El API de geolocalización define un **interfaz de alto nivel** que proporciona acceso a la información de localización (latitud y longitud) asociada sólo con el dispositivo que alberga la implementación.
- ✓ Las **fuentes de la información** de localización incluyen GPS y localización a través de señales de red tales como direcciones IP, RFID, direcciones MAC de WIFI y Bluetooth, repetidores GSM/CDMA, así como también la posición indicada por el usuario.
- ✓ **No garantiza** que devuelva **la localización real** del dispositivo.

Geolocalización. Descripción del API.

- Interfaz **Geolocation**

  Es el objeto utilizado para obtener la localización del dispositivo.
  Es necesario comprobar si el navegador la implementa:

```
if (window.navigator.geolocation){
  // Está soportada por el navegador. Aquí iría el código.
}
```

Métodos y objetos de la interfaz Geolocation

- **getCurrentPosition**(*función_si_OK* [, *función_si_ERROR* [, *opciones*]])

  Método para obtener la posición actual del dispositivo. Si obtiene la posición correctamente llama a *función_si_OK* pasándole un objeto **Position** con la posición.

  Si se produce un error y no se puede obtener la posición, se llama a función_si_ERROR pasándole un objeto PositionError con el código de error indicando el motivo.

  En *opciones* (**PositionOptions**) se puede modificar las condiciones por defecto en las que se realiza la consulta.

Geolocalización. Descripción del API.

- Interfaz **Geolocation**

```
✗ Interfaz Position{
   readonly attribute Coordinates coords;
   readonly attribute DOMTimeStamp timestamp;
};
```

```
✗ Interfaz Coordinates {
   readonly attribute double latitude;
   readonly attribute double longitude;
   readonly attribute double? altitude; // Puede no estar disponible
   readonly attribute double accuracy;
   readonly attribute double? altitudeAccuracy; // Puede no estar
disponible
   readonly attribute double? heading; // Puede no estar disponible
   readonly attribute double? speed; // Puede no estar disponible
};
```

```
✗ unsigned long DOMTimeStamp; // Milisegundos transcurridos desde
                              // el 1 de enero de 1970
```

Geolocalización. Descripción del API.

- Interfaz **Geolocation**

```
✗ Interfaz PositionError{
    readonly attribute unsigned short code; // Código del error
    readonly attribute DOMString message; // Texto del error
};

Posibles valores de code:

➢  PositionError.PERMISSION_DENIED = 1; // Permiso denegado de acceso al gps
➢  PositionError.POSITION_UNAVAILABLE = 2; // No se puede obtener la posición
➢  PositionError.TIMEOUT = 3; // Agotado el tiempo máximo para localización
```

```
✗ Interfaz PositionOptions {
    attribute boolean enableHighAccuracy; // Habilitar máxima precisión
    attribute long timeout; // Tiempo máximo de espera, en milisegundos.
    attribute long maximumAge; // Permite utilizar localizaciones de caché
                               // que no lleven en ella más del número de
                               // milisegundos indicados por maximumAge
};
```

Geolocalización. Descripción del API.

- Interfaz **Geolocation**
  - **watchPosition**(*función_si_OK* [, *función_si_ERROR* [, *opciones*]])

    Método que puede tomar los mismos argumentos que *getCurrentPosition* e inicializa un proceso de monitorización de la posición del dispositivo. Al ejecutarlo, devuelve un valor de tipo *long* que identifica unívocamente el proceso de monitorización inicializado.
  - **clearWatch**(*id_watch*).

    Método que detiene el proceso de monitorización inicializado con *watchPosition* y cuyo identificador se le pasa como argumento.

Geolocalización. Descripción del API.

- Interfaz **Geolocation**

Ejemplo: **getCurrentPosition**

Código JavaScript

```
function mostrarPosicion(posicion){
  var latitud = posicion.coords.latitude;
  var longitud = posicion.coords.longitude;
  alert("Latitud : " + latitud + " Longitud: " + longitud);
}
function errorHandler(err){
  if(err.code == 1) {
    alert("Error: Acceso denegado!");
  }else if( err.code == 2) {
    alert("Error: La posición no está disponible!");
  }
}
function getPosicion(){
  if(navigator.geolocation){// Se configura un timeout de 60 segundos
    var opciones = {enableHighAccuracy:true, timeout:60000};
    navigator.geolocation.getCurrentPosition(mostrarPosicion,errorHandler,opciones);
  }else{ alert("Tu navegador no soporta geolocalización!"); }
}
```

Código HTML

```
<form>
  <input type="button" onclick="getPosicion();" value="Obtener posición"/>
</form>
```

Geolocalización. Descripción del API.

- Interfaz **Geolocation**

Ejemplo: **watchPosition**

Código JavaScript

```
var watchID, geoLoc;
function mostrarPosicion(posicion) {
  var latitud = posicion.coords.latitude, longitud = posicion.coords.longitude;
  alert("Latitud : " + latitud + " Longitud: " + longitud);
}
function errorHandler(err) {
  if(err.code == 1) {
    alert("Error: Acceso denegado!");
  }else if( err.code == 2) {
    alert("Error: La posicion no está disponible!");
  }
}
function getPosicionActualizada(){
  if(navigator.geolocation){ // timeout en 60 seconds
    geoLoc = navigator.geolocation;
    watchID = geoLoc.watchPosition(showLocation, errorHandler, {timeout:60000} );
   }else{ alert("Tu navegador no soporta geolocalización!"); }
}
```

```
<form>
  <input type="button" onclick="getPosicionActualizada();" value="Actualizar pos."/>
</form>
```

Código HTML

Web Storage

Web Storage

- Este API permite guardar información de forma local, es decir, en la parte del cliente.
- Es más eficiente que las cookies. Permite almacenar cantidades de datos mucho mayores, siempre en modo texto.
- Al contrario que las cookies, la información almacenada mediante este API **SÓLO** es accesible desde el navegador, **NO** desde el servidor.

Comparativa Cookies / Web Storage	COOKIES	WEB STORAGE
Capacidad de almacenamiento	4KB aprox.	5MB a 10MB, aprox.
Peticiones	Recurrencia	Mejor rendimiento
Acceso	getCookie(), setCookie()	clave/valor

Web Storage

Tipos de almacenamiento

Este API proporciona dos tipos de almacenamiento:

- **sessionStorage**. Este sistema de **almacenamiento** guarda los datos de forma **temporal** durante la sesión de navegación en una ventana del navegador. Es decir, sólo estarán disponibles en la ventana del navegador en la que fueron guardados y hasta que ésta se cierre.
- **localStorage**. Este sistema de **almacenamiento** permite guardar los datos de forma **permanente** siendo, además, accesibles desde todas las ventanas del navegador, no sólo desde la ventana en la que fueron guardados. Estarán disponibles hasta que el usuario los elimine.

Web Storage

Interfaz Storage

Tanto **sessionStorage** como **localStorage** implementan la interfaz **Storage**.

```
Interfaz Storage{
    readonly attribute unsigned long length;
    DOMString? key(unsigned long index);
    getter DOMString getItem(DOMString key);
    setter creator void setItem(DOMString key, DOMString value);
    deleter void removeItem(DOMString key);
    void clear();
};
```

- **length**: Devuelve el número de pares clave/valor.
- **key(*n*)**: Devuelve el nombre de la clave que ocupa la posición *n,* o *null* si no existe.
- **getItem(*key*)**: Devuelve el valor asociado a la clave *key*.
- **setItem(*key*, *value*)**: Si la clave *key* no existe, crea un nuevo par clave/valor con *key*/*value*. Si la clave ya existe, actualiza su valor a *value*.
- **removeItem(*key*)**. Borra el par clave/valor cuya clave coincida con *key*.
- **clear()**: Borra de la lista todos los pares clave/valor.

Web Storage

Ejemplo:

JavaScript

```
function comprobar(){
  if(window.localStorage){ // Se comprueba si hay soporte para Web Storage
    var frm = document.querySelectorAll("form")[0];
     if(frm.ckbGuardar.checked){ // Si se ha marcado guardar datos ...
       localStorage.setItem("login", frm.login.value);
       localStorage["pass"] = frm.pass.value; // modo alternativo
    }
  }
}
function rellenar(){ // Se comprueba si hay soporte para Web Storage
  if(window.localStorage){
    var frm = document.querySelectorAll("form")[0];
     if(localStorage.getItem("login")){ // Si hay datos en loginStorage ...
       frm.login.value = localStorage.getItem("login");
       frm.pass.value = localStorage.pass; // modo alternativo
     }
  }
}
```

HTML

```
<body onload="rellenar();">
  <form onsubmit="return comprobar();">
    Login:<input type="text" name="login"><br>
     Password:<input type="password" name="pass"><br>
     <input type="checkbox" name="ckbGuardar"> Usar localStorage<br>
     <input type="submit" value="Enviar">
  </form>
</body>
```

IndexedDB

IndexedDB

- ✓ **No** es una base de datos relacional.
- ✓ Permite crear un almacén para guardar localmente objetos JavaScript de un mismo tipo de datos y poder trabajar con ellos ***offline***.
- ✓ Cada almacén puede tener una **colección de índices** para facilitar las consultas de objetos y su recorrido.
- ✓ Las consultas no se hacen en SQL sino por un índice. Esto proporciona un ***cursor*** que permite recorrer los objetos resultado de la consulta.
- ✓ Utilizan como estructuras de datos para guardar la información los **árboles-B**, que mantienen los datos ordenados y permiten hacer inserciones y eliminaciones en tiempo logarítmico, así como recorrer grandes cantidades de datos de forma eficiente.

IndexedDB

Operaciones básicas

- Comprobar si el navegador soporta indexedDB

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

if(window.indexedDB)
{
    alert('Tu navegador soporta IndexedDB!!!');
}
else
{
    alert('Tu navegador NO soporta IndexedDB!!!');
}
```

IndexedDB

Operaciones básicas

- **Abrir / Crear una base de datos**

```
var request = indexedDB.open(nombre, versión)
```

- *nombre*: Nombre de la base de datos.

- *versión*: Optativo. Valor > 0. Por defecto es 1. Permite abrir distintas versiones de la base de datos. Cuando se quiere hacer algún cambio en la estructura de la BD, ésta se debe abrir con un número de versión nuevo.

Al realizar la petición para abrir la BD se dispara uno de los siguientes tres eventos, por lo que es conveniente asignarles una función *callback*:

- *request.***onupgradeneeded**. Se dispara si la BD no existe. En la misma petición se crea automáticamente la BD y en el código asociado al evento hay que crear las tablas que vaya a contener.
- *request.***onsuccess**. Se dispara cuando la BD ya existe.
- *request.***onerror**. Se dispara si se ha producido un error al abrir la BD.

IndexedDB

Operaciones básicas

- **Abrir / Crear una base de datos**

Ejemplo: Creación de una base de datos

```
var request = indexedDB.open('bd_prueba');
request.onupgradeneeded = function(e){ // La BD no existía
  var db = e.target.result; // Se crea la BD y se guarda la conexión.
  // Aquí se crea la estructura de la BD: tablas, etc.
};
request.onsuccess = function(e){ // La BD ya existe
  // Se hacen las operaciones de consulta/actualización de los datos, NUNCA
  // de la estructura de la tabla o la BD. Es conveniente cerrar la
conexión.
}
request.onerror = function(e){ // Se ha producido un error. Además,
  // es el manejador genérico de errores que se produzcan en las
  // peticiones de la base de datos
  console.log('ERROR en la base de datos: ' + e.target.errorCode);
}
```

En cualquier momento se puede cerrar la conexión a la BD mediante: db.**close**()

IndexedDB

Operaciones básicas

- **Borrar una base de datos**

```
var request = indexedDB.deleteDatabase(nombre)
```

- nombre: Nombre de la base de datos a borrar.

Al realizar la petición para borrar la BD se dispara uno de los siguientes eventos, por lo que es conveniente asignarles una función *callback*:

✓ *request.***onsuccess**. Se dispara cuando la BD se ha borrado correctamente.

✓ *request.***onerror**. Se dispara si se ha producido un error al borrar la BD.

Ejemplo:

```
...
var request = indexedDB.deleteDatabase('bd_prueba');
request.onsuccess = function(e){
  console.log("BD borrada correctamente.");
}
```

IndexedDB

Operaciones básicas

- **Crear una tabla (*almacén*) en la base de datos**

```
var store = db.createObjectStore(nombre, parámetros)
```

- *nombre*: Nombre de la tabla.

- *parámetros*: Optativo. Permite los siguientes pares atributos/valor.

✓ *keyPath*: nombre del campo que servirá de clave primaria.

✓ *autoIncrement*: valor booleano para añadir un campo autonumérico (*true*) o no (*false*). El campo autonumérico sería *keyPath*.

Ejemplo:

```
var request = indexedDB.open('bd_prueba');
request.onupgradeneeded = function(e){ // La BD no existía
  var db = e.target.result; // Conexión a la BD
  var store = db.createObjectStore("usuarios", {keyPath:'dni'});
  ...
};
```

IndexedDB

Operaciones básicas

- **Crear una tabla (*almacén*) en la base de datos**

  Se pueden crear **índices** (campos) en la tabla mediante el siguiente método:

```
var idx = store.createIndex(nombre_índice, campo, parámetros)
```

  - *nombre_índice*: Nombre del índice a crear.

  - *campo*: campo de la tabla sobre el que se creará el índice.

  - *parámetros*: Optativo. Permite los siguientes pares atributos/valor:

  ✓ *unique*: valor booleano. Si es *true* no permitirá valores duplicados en el campo especificado.

  ✓ *multiEntry*: valor booleano. Si es *true* se añadirá una entrada al índice por cada valor duplicado en el campo. Si es *false*, sólo habrá una entrada en el índice por valor y si hay más de un registro con el mismo valor de campo, éstos estarán almacenados en un array al que apuntará la única entrada en el índice.

IndexedDB

Operaciones básicas

- **Crear una tabla (*almacén*) en la base de datos**

  Ejemplo: Creación de índices (campos) en una tabla

```
var request = indexedDB.open('bd_prueba');
request.onupgradeneeded = function(e){ // La BD no existía
  var db = e.target.result; // Se crea la BD
  var store = db.createObjectStore('usuarios', {keyPath: 'dni'});

  var nomIndex = store.createIndex('idx_nombre', 'nombre');
  var apeIndex = store.createIndex('idx_apellidos', 'apellidos');
  var emaIndex = store.createIndex('idx_email','email',{unique:
true});
  ...
};
```

IndexedDB

Operaciones básicas

- **Borrar un *almacén* (store)**

```
var request = db.deleteObjectStore(nombre)
```

- nombre: Nombre del almacén (*store*) a borrar.

Al realizar la petición para borrar un almacén (*store*) se dispara uno de los siguientes eventos, por lo que es conveniente asignarles una función *callback*:

- ✓ *request.***onsuccess**. Se dispara cuando la BD se ha borrado correctamente.
- ✓ *request.***onerror**. Se dispara si se ha producido un error al borrar la BD.

Ejemplo:

```
...
var request = db.deleteObjectStore('usuarios');
request.onsuccess = function(e){
  console.log("Almacén (store) borrado correctamente.");
}
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**

  Para poder realizar las operaciones básicas con registros (añadir, eliminar, recuperar) fuera del evento **onupgradeneeded** es necesario hacerlo en una ***transacción***. Para crear una transacción se utiliza la siguiente instrucción:

```
var tx = db.transaction([nombre_tablas], modo)
```

  - nombre_tablas: Nombres de las tablas (separados por comas y entre comillas) con las que se va a trabajar. Si sólo se va a trabajar con una tabla, se pueden obviar los corchetes [ ].

  - modo: Optativo. Permite indicar el tipo de acceso que se va a realizar a las tablas. Los posibles valores son ***readonly***, que es el valor por defecto; y ***readwrite***.

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**

  Ejemplo: Uso de transacciones

```
// Se abre la BD. Si existe ya, se dispara el método onsuccess
var request = indexedDB.open('bd_prueba');

request.onsuccess= function(e){ // La BD existía
  // Se obtiene el enlace a la BD
  var db = e.target.result;

  // Se crea la transacción para poder operar con la tabla 'usuarios'
  var tx = db.transaction(['usuarios'], 'readwrite');
  // Se obtiene el objeto almacén (tabla o store) 'usuarios'
  var store = tx.objectStore('usuarios');
  ...
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Añadir registros**

```
var request = store.put(valor, clave)
```

- valor: Valor a almacenar. Puede ser un objeto en formato *literal object*.

- clave: Optativo. Valor de clave con la que se guardará el *valor* y mediante el cual se podrá recuperar más tarde. Si *valor* es un objeto y una de sus propiedades es la clave, no hace falta especificar este parámetro.

Todas las operaciones asíncronas que se realizan a bases de datos de *indexedDB* y a sus objetos, devuelven un objeto de tipo ***request*** con el resultado. Este objeto *request* tiene las propiedades *result* (resultado de la operación), *error* (indica el error que se haya producido), *source* (origen de la petición), *transaction* (transacción en la que se hace la petición) y *readyState* (estado de la petición, estado inicial *pending* y estado final *done*). También tiene los eventos *onerror* y *onsuccess*.

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Añadir registros**

Ejemplo: Añadir registros a una tabla (almacén o *store*)

```
var request = indexedDB.open('bd_prueba');
request.onsuccess = function(e){ // La BD existía
    var db = e.target.result; // Enlace a la BD
    var tx = db.transaction(['usuarios'], 'readwrite'); // Transacción
    var store = tx.objectStore('usuarios'); // Se obtiene la tabla
    var u = {dni:'123456A', nombre:'Juan',
             Apellidos:'García Ruiz', email:'jgarcia@correo.es'};
    var request = store.put(u); // Se añade el nuevo registro
    ...
};
```

Existe otro método ***add()*** que, a diferencia de ***put()***, sólo permite añadir nuevos registros, mientras que *put()* permite, también, modificarlos.

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Eliminar registros**

```
var request = store.delete(clave)
```

- clave: Valor de la clave del registro a borrar. La transacción utilizada debe ser del tipo *readwrite*.

Ejemplo: Eliminar registros a una tabla (almacén o *store*)

```
var request = indexedDB.open('bd_prueba');
request.onsuccess = function(e){ // La BD existía
    var db = e.target.result; // Enlace a la BD
    var tx = db.transaction(['usuarios'], 'readwrite'); // Transacción
    var store = tx.objectStore('usuarios'); // Se obtiene la tabla
    var request = store.delete('32145B'); // Se elimina el registro con
                                          // valor de clave '32145B'
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Buscar un registro**

    Para obtener un registro de una tabla se puede hacer de dos maneras: **utilizando la clave primaria** o **utilizando uno de los índices creados en la tabla**. En ambos casos se utiliza el método **get**(*valor*) y sólo se devuelve un registro, si lo encuentra. La única diferencia es que en el primer caso se aplica al objeto *store* en el que se quiera buscar; y en el segundo caso se aplica al índice mediante el cuál se quiere buscar.

```
var request = objeto.get(valor)
```

- *objeto*: Es el objeto tabla (almacén o *store*) o el objeto índice (*index*) en el que se hace la búsqueda.

- *valor*: Valor a buscar en la clave primaria o el índice.

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Buscar un registro**

Ejemplo: Buscar un registro mediante la clave primaria

```
var request = indexedDB.open('bd_prueba');
request.onsuccess = function(e){ // La BD existía
  var db = e.target.result; // Enlace a la BD
  var tx = db.transaction(['usuarios'], 'readonly'); // Transacción
  var store = tx.objectStore('usuarios'); // Se obtiene la tabla
  var request = store.get('32145B'); // Se pide el registro
  request.onsuccess = function(e){
    var registro = e.target.result;
    if(registro != undefined){ // registro encontrado
      ...
    }
  };
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Buscar un registro**

Ejemplo: Buscar un registro mediante un índice

```
var request = indexedDB.open('bd_prueba');
request.onsuccess = function(e){ // La BD existía
  var db = e.target.result; // Enlace a la BD
  var tx = db.transaction(['usuarios'], 'readonly'); // Transacción
  var store = tx.objectStore('usuarios'); // Se obtiene la tabla
  var idx = store.index('nomIndex'); // Se selecciona el índice a usar
  var request = idx.get('Juan'); // Se pide el registro
  request.onsuccess = function(e){
    var registro = e.target.result;
    if(registro != undefined){ // registro encontrado
      ...
    }
  };
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Actualizar registros**

    Se puede modificar un registro de una tabla una vez encontrado utilizando cualquiera de los dos métodos anteriores para buscarlo.

    Ejemplo:

```
var tx = db.transaction(['usuarios'], 'readwrite'); // Transacción
readwrite
var store = tx.objectStore('usuarios'); // Se obtiene la tabla
var request = store.get('32145B'); // Se pide el registro
request.onsuccess = function(e){
  var registro = e.target.result;
  if(registro != undefined){ // registro encontrado
    registro.nombre = 'María'; // se cambia el nombre
    var requestUpdate = store.put(registro); // guarda registro modificado
    requestUpdate.onsuccess = function(e){ // todo ok!!    };
    requestUpdate.onerror = function(e){ // error    };
  }
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Obtener varios registros y recorrerlos**

    Se pueden hacer consultas que nos devuelvan no sólo un registro, sino un conjunto de ellos. Para ello se debe utilizar el objeto ***cursor***. Un objeto *cursor* se puede abrir (al igual que ocurre al aplicar el método **get**) sobre un objeto *store* o sobre un objeto *índice* para recorrer todos los registros o filtrar y seleccionar un subconjunto de ellos.

    Junto al objeto *cursor*, se utiliza la interfaz **IDBKeyRange** que va a permitir establecer la condición de filtro a aplicar sobre el conjunto de registros.

```
var request = objeto.openCursor(valor)
```

    - *valor*: puede ser un valor concreto para localizar un registro, o bien un intervalo expresado mediante la interfaz **IDBKeyRange**.

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Obtener varios registros y recorrerlos**

Interfaz **IDBKeyRange**:

Rango	Expresión
X <= *a*	`IDBKeyRange.upperBound(a)`
X < *a*	`IDBKeyRange.upperBound(a, true)`
X >= *b*	`IDBKeyRange.lowerBound(b)`
X > *b*	`IDBKeyRange.lowerBound(b, true)`
a <= X <= *b*	`IDBKeyRange.bound(a, b)`
a < X < *b*	`IDBKeyRange.bound(a, b, false, false)`
a < X <= *b*	`IDBKeyRange.bound(a, b, true, false)`
a <= X < *b*	`IDBKeyRange.bound(a, b, false, true)`
X = *a*	`IDBKeyRange.only(a)`

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Obtener varios registros y recorrerlos**

Ejemplo:

```
var tx = db.transaction(['usuarios'], 'readonly'); // Transacción
var store = tx.objectStore('usuarios'); // Se obtiene la tabla

var rango = IDBKeyRange.bound('G','M'); // Se establece el intervalo
var idx = store.index('apeIndex'); // Se selecciona el índice a usar
var request = idx.openCursor(rango); // Se piden los registros
request.onsuccess = function(e){
  var cursor = e.target.result;
  if(cursor){ // registro encontrado
    console.log(JSON.stringify(cursor.value)); // imprime el registro
    cursor.continue(); // siguiente registro
  }
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**
  - ➢ **Recorrer todos los registros de una tabla (almacén o *store*)**

Ejemplo:

```
var tx = db.transaction(['usuarios'], 'readonly'); // Transacción
var store = tx.objectStore('usuarios'); // Se obtiene la tabla

var request = store.openCursor(); // Se piden los registros
request.onsuccess = function(e){
  var cursor = e.target.result;
  if(cursor){ // registro encontrado
    console.log(JSON.stringify(cursor.value)); // imprime el registro
    cursor.continue(); // siguiente registro
  }
};
```

IndexedDB

Operaciones básicas

- **Operaciones con registros en una tabla (*almacén*)**

  Interfaz **IDBCursor**:

  ✓ **Métodos**

  - *IDBCursor*.**advance(*n*)**. Indica el número de registros a avanzar.
  - *IDBCursor*.**continue()**. Avanza al siguiente registro.
  - *IDBCursor*.**delete()**. Elimina el registro actual.
  - *IDBCursor*.**update(*nuevoValor*)**. Actualiza el objeto en el que se encuentra el cursor y almacena en él el nuevo valor que se le pasa como parámetro.

  ✓ **Propiedades** (sólo lectura)

  - *IDBCursor*.**source**. Devuelve el *store* o el *index* sobre el que está iterando el cursor.
  - *IDBCursor*.**direction**. Devuelve la dirección del cursor: *next*, *nextunique*, *prev*, *prevunique*.
  - *IDBCursor*.**key**. Devuelve la clave del registro en el que se encuentra el cursor, dependiendo de la fuente (*source*) que esté recorriendo.
  - *IDBCursor*.**primaryKey**. Devuelve la clave primaria efectiva del registro en el que se encuentra el cursor, independientemente del *source*.

IndexedDB

Operaciones básicas

- **Abrir una base de datos para añadirle más tablas (*stores*)**

  Se pueden añadir, modificar o eliminar *stores* de una base de datos ya creada y/o *índices* de un *store* ya creado. Para ello, la base de datos se debe abrir con un nuevo número de versión no utilizado.

  Ejemplo:

```
var request = indexedDB.open('bd_prueba', 2); // Versión de la BD: 2
request.onupgradeneeded = function(e){
  var db = e.target.result; // Conexión a la BD
  // Si el store existe en la versión anterior, es necesario borrarlo.
  var store = db.createObjectStore('foros', {KeyPath:'idForo',
                                              autoIncrement:true});
  var idx = store.createIndex("idx_tema","tema");
  var store2 = e.target.transaction.objectStore('mensajes');
  var idx2 = store2.createIndex("idx_titulo","titulo");
};
```