



Desarrollo de aplicaciones multiplataforma

Elaborado por: José Antonio Sánchez

Módulo 3

Programación HTML5

SESIÓN 5

9. Canvas

Proporciona un API para dibujar líneas, formas, imágenes, texto, etc en 2D, sobre un “lienzo”

Este API ya está siendo utilizado de manera exhaustiva, en la creación de fondos interactivos, elementos de navegación, herramientas de dibujo, juegos o emuladores

Es uno de los elementos que cuenta con una de las mayores especificaciones dentro de HTML5

9. Canvas

Elementos básicos

```
<canvas id="tutorial" width="150" height="150"></canvas>
```

Los dos atributos width y height, son opcionales y pueden establecerse mediante las propiedades DOM

Si no se especifican, el tamaño inicial será de 300px por 150px de alto

Este elemento pueden modificarse utilizando CSS, pero las reglas afectarán al elemento, no a lo dibujado en el lienzo

9. Canvas

Elementos básicos

Todos los navegadores son compatibles con el elemento canvas, la diferencia entre ellos radica en qué funcionalidades del API han implementado

La manera de dibujar en el lienzo es a través de JavaScript

Lo primero es obtener el contexto de dibujado, que se utilizará para crear y manipular el contenido mostrado

9. Canvas

Elementos básicos

En el contexto de representación 2D, el canvas está inicialmente en blanco. Es necesario el acceso al DOM para usar las funciones de dibujo

```
var canvas = document.getElementById('tutorial');  
var ctx = canvas.getContext('2d');
```

También existe un contexto para “3D” llamado WebGL

```
var gl = canvas.getContext("experimental-webgl");
```

Actualmente es un API de 2D pero proporcionar “shaders” para realizar un efecto de tres dimensiones

9. Canvas

```
<html>
<head>
  <script type="application/javascript">
    window.onload = function() {
      var canvas = document.getElementById("canvas");
      if (canvas.getContext) {
        var ctx = canvas.getContext("2d");
        ctx.fillStyle = "rgb(200,0,0)";
        ctx.fillRect (10, 10, 55, 50);
        ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
        ctx.fillRect (30, 30, 55, 50);
      }
    };
  </script>
</head>
<body><canvas id="canvas" width="150" height="150"></canvas></body>
</html>
```



9.1. Dibujar formas

Dentro del elemento canvas tenemos la “cuadrícula” o espacio de coordenadas

El punto de origen se coloca en la esquina superior izquierda, punto(0,0), y todos los elementos se colocan con relación a este origen

Normalmente, una unidad en la cuadrícula corresponde a un px en el lienzo

Solamente se admite una forma primitiva de dibujo: los rectángulos, el resto de las formas deberán crearse mediante la combinación de una o más funciones

9.1. Dibujar formas

Existen tres funciones que dibujan un rectángulo en el lienzo:

- ✓ `fillRect(x,y,width,height)`: dibuja un rectángulo relleno
- ✓ `strokeRect(x,y,width,height)`: dibuja un contorno rectangular
- ✓ `clearRect(x,y,width,height)`: borra el área especificada y hace que sea totalmente transparente

Cada una de estas funciones tiene los mismos parámetros: x e y especifican las coordenadas en el lienzo, width es la anchura y height la altura

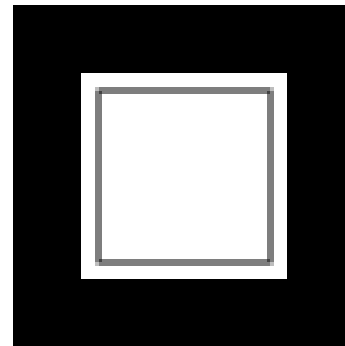
Las tres funciones de rectángulo dibujan inmediatamente en el lienzo

9.1. Dibujar formas

```
var canvas = document.getElementById('tutorial');  
var ctx = canvas.getContext('2d');  
ctx.fillRect(25,25,100,100);  
ctx.clearRect(45,45,60,60);  
ctx.strokeRect(50,50,50,50);
```

9.1. Dibujar formas

```
var canvas = document.getElementById('tutorial');  
var ctx = canvas.getContext('2d');  
ctx.fillRect(25,25,100,100);  
ctx.clearRect(45,45,60,60);  
ctx.strokeRect(50,50,50,50);
```



9.2. Trazar rutas

Las rutas son utilizadas para dibujar formas: líneas, curvas, polígonos, etc

Las rutas se almacenan como una lista de subrutas (líneas, arcos, etc.) que, en conjunto, forman una figura. Cada vez que se llama al método “**beginPath**”, la lista se pone a cero y podemos empezar a dibujar nuevas formas

El paso final sería llamar al método “**closePath**”: este método intenta cerrar la forma trazando una línea recta desde el punto actual hasta el inicial

9.2. Trazar rutas

```
var canvas = document.getElementById('tutorial');  
var context = canvas.getContext('2d');  
  
context.beginPath();  
//... path drawing operations  
context.closePath();
```

El siguiente paso es dibujar la forma como tal utilizando las funciones de dibujo de líneas y arcos

9.2. Trazar rutas

moveTo

La función moveTo, no dibuja nada. Es utilizada para colocar el punto de partida en otro lugar o para dibujar rutas inconexas

```
ctx.beginPath();  
ctx.moveTo(25,25); // Punto inicial para dibujar  
ctx.lineTo(105,25);  
ctx.lineTo(25,105);  
ctx.closePath();
```

9.2. Trazar rutas

lineTo

Este método toma dos argumentos x e y, que son las coordenadas del punto final de la línea. El punto de partida depende de las rutas anteriores

```
ctx.beginPath();  
ctx.moveTo(25,25); // Posición inicial  
ctx.lineTo(105,25); // Línea hasta la posición 105, 25  
ctx.lineTo(25,105); // Línea hasta la posición 25, 105  
ctx.closePath(); // Cerrar la figura  
ctx.fill();
```


9.2. Trazar rutas

lineTo

```
ctx.beginPath();  
ctx.moveTo(125,125);  
ctx.lineTo(125,45);  
ctx.lineTo(45,125);  
ctx.closePath();  
ctx.stroke();
```

“stroke” se utiliza para dibujar una forma con contorno

“fill” se utiliza para pintar una forma sólida

9.2. Trazar rutas

arc

Para dibujar arcos o círculos se utiliza el método arc (la especificación también describe el método arcTo)

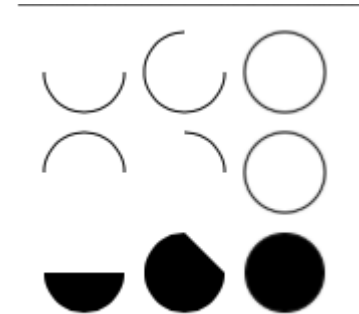
```
ctx.arc(x,y,radius,startAngle,endAngle, anticlockwise);
```

Este método toma cinco parámetros: x e y, el radio, startAngle y endAngle (puntos de inicio y final del arco en radianes) y anticlockwise (booleano)

9.2. Trazar rutas

arc

```
for(var i=0;i<4;i++){  
  for(var j=0;j<3;j++){  
    ctx.beginPath();  
    var x      = 25+j*50;    // coordenada x  
    var y      = 25+i*50;    // coordenada y  
    var radius  = 20;        // radio del arco  
    var startAngle = 0;      // punto inicial del círculo  
    var endAngle   = Math.PI+(Math.PI*j)/2; // punto final  
    var anticlockwise = i%2==0 ? false : true;  
    ctx.arc(x,y,radius,startAngle,endAngle, anticlockwise);  
    if (i>1) ctx.fill();  
    else ctx.stroke();  
  }  
}
```



9.2. Trazar rutas

`quadraticCurveTo()`

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

`bezierCurveTo()`

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

`arc`, `bezier` y `quadratic` utilizan radianes, pero si preferimos trabajar en grados, es necesario convertirlo a radianes:

```
var radians = degrees * Math.PI / 180;
```

9.3. Colores

Si queremos aplicar colores a una forma, hay dos características importantes que podemos utilizar: `fillStyle` y `strokeStyle`.

```
fillStyle = color;    // Relleno de la figura  
strokeStyle = color; //Contorno de la figura
```

El color puede ser una cadena que representa un valor de color CSS, un objeto degradado o un objeto modelo

```
ctx.fillStyle = "orange";  
ctx.fillStyle = "#FFA500";  
ctx.fillStyle = "rgb(255,165,0)";  
ctx.fillStyle = "rgba(255,165,0,1)";
```

9.3. Colores

Ejemplo de fillStyle

```
function draw() {  
  for (var i=0;i<6;i++){  
    for (var j=0;j<6;j++){  
      ctx.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ','  
        + Math.floor(255-42.5*j) + ',0)';  
      ctx.fillRect(j*25,i*25,25,25);  
    }  
  }  
}
```

9.3. Colores

Ejemplo de strokeStyle

```
function draw() {  
  for (var i=0;i<6;i++){  
    for (var j=0;j<6;j++){  
      ctx.strokeStyle = 'rgb(0,' + Math.floor(255-42.5*i)  
        + ',' + Math.floor(255-42.5*j) + ')';  
      ctx.beginPath();  
      ctx.arc(12.5+j*25,12.5+i*25,10,0,Math.PI*2,true);  
      ctx.stroke();  
    }  
  }  
}
```

9.4. Degradados y patrones

Los degradados funcionan de una manera similar a los definidos en CSS3, donde se especifican el inicio y los pasos de color para el degradado.

Los patrones, permiten definir una imagen como origen y especificar el patrón de repetido, de nuevo de manera similar a como se realizaba con la propiedad background-image de CSS

Lo que hace interesante al método createPattern es que como origen podemos utilizar una imagen, un canvas o un elemento de vídeo

9.4. Degradados y patrones

Los argumentos de `createLinearGradient` son el punto de inicio del degradado (x1 e y1) y el punto final del degradado (x2 e y2)

```
var canvas = document.getElementById('tutorial');  
var ctx = canvas.getContext('2d');  
var gradient = ctx.createLinearGradient(0, 0, 0, canvas.height);
```

```
gradient.addColorStop(0, '#fff');  
gradient.addColorStop(1, '#000');  
ctx.fillStyle = gradient;  
ctx.fillRect(0, 0, canvas.width, canvas.height);
```



9.4. Degradados y patrones

Los degradados radiales son muy similares, con la excepción que definimos el radio después de cada coordenada:

```
gradient = ctx.createRadialGradient(canvas.width/2,  
    canvas.height/2,  
    0,  
    canvas.width/2,  
    canvas.height/2,  
    150);
```

El primer punto del degradado se define en el centro del canvas, con radio cero y el siguiente punto se define con un radio de 150px pero su origen es el mismo, lo que produce un degradado circular

9.5. Transparencias

Podemos dibujar formas semitransparentes mediante la propiedad `globalAlpha`

`globalAlpha = transparency value`

```
ctx.globalAlpha = 0.2;  
// Dibujar círculos semitransparentes  
for (var i=0;i<7;i++){  
    ctx.beginPath();  
    ctx.arc(75,75,10+10*i,0,Math.PI*2,true);  
    ctx.fill();  
}
```

9.6. Transformaciones

Podemos definir algunas transformaciones como rotación, escalado, transformación y traslación (similares a las conocidas de CSS3)

ctx.translate(x, y);

Traslada el centro de coordenadas desde su posición por defecto (0, 0) a la posición indicada

9.6. Transformaciones

ctx.rotate(angle);

Inicia la rotación desde su posición por defecto (0,0)

Si se rota el canvas desde esta posición, el contenido podría desaparecer por los límites del lienzo, por lo que puede necesario definir un nuevo origen para la rotación

Sólo precisa tomar el ángulo de rotación que se aplicará al marco. Este parámetro es una rotación dextrógira medida en radianes

9.6. Transformaciones

ctx.scale(x, y);

Aumentar o disminuir las unidades del tamaño de nuestro marco

x e y, definen el factor de escala en la dirección horizontal y vertical respectivamente

Los valores menores que 1.0 reducen el tamaño de la unidad y los valores mayores que 1.0 aumentan el tamaño de la unidad

Por defecto, una unidad en el área de trabajo equivale exactamente a un píxel

10. Geolocalización

El uso del API de geolocalización es muy sencillo y está soportado por todos los navegadores modernos

Permite conocer la posición del usuario con mayor o menor precisión, según el método :

- ✓ Vía IP. Nos da una ligera idea de dónde se encuentra a través de la dirección IP pública
- ✓ Redes GSM. Obtiene una posición aproximada basándose en una triangulación con las antenas de telefonía
- ✓ GPS. Es el método más preciso, pudiendo concretar la posición del usuario con un margen de error de escasos metros

10. Geolocalización

El primer paso debería ser la comprobación de la disponibilidad del API de geolocalización de HTML 5 en el explorador del usuario:

```
if(Modernizr.geolocation) {  
    alert('El explorador soporta geolocalización');  
} else {  
    alert('El explorador NO soporta geolocalización');  
}
```


10. Geolocalización

El API ofrece los siguientes métodos para obtener la posición del usuario:

- ✓ `getCurrentPosition`: obtiene la posición actual del usuario, utilizando la mejor tecnología posible
- ✓ `watchPosition`: consulta cada cierto tiempo la posición del usuario, ejecutando la función de callback indicada únicamente si la posición ha cambiado desde la última consulta

Ambos métodos se ejecutan de manera asíncrona para obtener la posición del usuario

10. Geolocalización

Según la especificación: "El navegador no debe enviar información sobre la localización a sitios sin el permiso explícito del usuario."

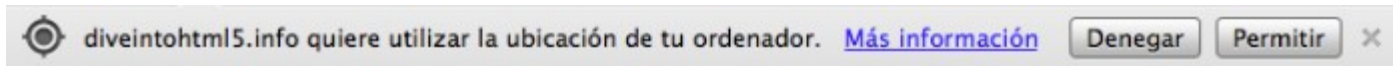
Por tanto si es la primera vez que se solicita la localización al navegador, éste mostrará un mensaje pidiendo permiso al usuario para compartir su localización

Si el usuario no da su permiso, el API llama a la función de error que hayamos definido.

10. Geolocalización

Los navegadores serán quienes informen al usuario que estamos intentando acceder a su posición actual, así pues, la forma de realizarlo varía:

- ✓ Por norma general, los navegadores de escritorio muestran un aviso no bloqueante, lo que permite seguir utilizando y ejecutando la aplicación



- ✓ En cambio, los navegadores de dispositivos móviles suelen mostrar una ventana modal que bloquea la ejecución del código hasta que el usuario acepte o deniegue la solicitud



10.1. Métodos el API

El API de geolocalización del objeto **navigator** contiene tres métodos:

- ✓ `getCurrentPosition`
- ✓ `watchPosition`
- ✓ `clearWatch`

`getCurrentPosition` devuelve la localización en ese momento

En cambio, `watchPosition` y `clearWatch` están emparejados: `watchPosition` devuelve un identificador único, que permite cancelar posteriormente las consultas de posición pasando es identificador como parámetro a `clearWatch`

10.1. Métodos el API

Tanto `getCurrentPosition` como `watchPosition`, son métodos muy parecidos, y usan los mismos parámetros:

- ✓ función de éxito
- ✓ función de error
- ✓ opciones de geolocalización

Un ejemplo muy sencillo de utilización sería el siguiente

```
navigator.geolocation.getCurrentPosition(function (position) {  
    alert('We found you!');  
});
```

10.1. Métodos el API

función de éxito

Es el primer argumento de las funciones de `getCurrentPosition` y `watchPosition`

Esta función recibe como parámetro un objeto **position** que contiene dos propiedades:

- ✓ un objeto `coords` (contiene las coordenadas)
- ✓ una marca de tiempo `timestamp`

El objeto de coordenadas es el que contiene la información sobre la geolocalización

10.1. Métodos el API

función de éxito

Las propiedades del objeto **coords** son las siguientes:

- ✓ latitude - readonly attribute double latitude
- ✓ longitude - readonly attribute double longitude
- ✓ accuracy - readonly attribute double accuracy (precisión en metros)

```
if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(function (position) {  
        var coords = position.coords;  
        showMap(coords.latitude, coords.longitude, coords.accuracy);  
    });  
}
```

10.1. Métodos el API

datos extra

Para dispositivos que dispongan de GPS, el objeto **coords** dispone de otras propiedades para proporcionar más información, aunque en la gran mayoría de navegadores tendrán el valor *null*

- ✓ altitude - readonly attribute double altitude
- ✓ altitudeAccuracy - readonly attribute double altitudeAccuracy
- ✓ heading - readonly attribute double heading
- ✓ speed - readonly attribute double speed

10.1. Métodos el API

datos extra

Para la obtención de datos del GPS, en la mayoría de los casos, hay que especificar al API que utilice una mayor precisión para activar el GPS

```
{ enableHighAccuracy: true }
```

Esto se debe a que el uso del GPS consume muchísima batería y hay que utilizar esta tecnología únicamente si es estrictamente necesario

Además, para calcular datos como la velocidad, el dispositivo necesita conocer la diferencia media entre las últimas localizaciones. Por esta razón, es necesario utilizar el método `watchPosition` en lugar de `getCurrentPosition`

10.1. Métodos el API

datos extra

```
var speedEl = document.getElementById('speed');
navigator.geolocation.watchPosition(function (geodata) { // Función de éxito
    var speed = geodata.coords.speed;
    if (speed === null || speed === 0) {
        speedEl.innerHTML = "You're standing still!";
    } else {
        speedEl.innerHTML = speed + "Mps"; // speed is in metres per second
    }
}, function (error) { // Función de error
    speedEl.innerHTML = "Unable to determine speed :-(";
}, { enableHighAccuracy: true } // Opciones de geolocalización
);
```

10.1. Métodos el API

función de error

Esta función es importante si queremos proveer de un segundo método alternativo para introducir la localización o queremos informar al usuario

Se ejecuta cuando el usuario deniega la petición de localización, o cuando el dispositivo pierde la recepción de la localización

La función de error recibe un único argumento con dos propiedades:

- ✓ code - readonly attribute unsigned short code
- ✓ message - readonly attribute DOMString message

10.1. Métodos el API

función de error

Además, el código de error puede contener los siguientes valores:

- ✓ PERMISSION_DENIED (valor = 1)
- ✓ POSITION_UNAVAILABLE (valor = 2)
- ✓ TIMEOUT (valor = 3)

```
navigator.geolocation.getCurrentPosition(function(position) {  
    console.log(position);  
}, function(error) {  
    console.log(error.code+" "+error.message);  
});
```

10.1. Métodos el API

opciones de geolocalización

El tercer argumento para los métodos `getCurrentPosition` y `watchPosition` contiene las opciones de geolocalización

- ✓ `enableHighAccuracy`: booleano, por defecto `false`
- ✓ `timeout`: en milisegundos, por defecto infinito
- ✓ `maximumAge`: en milisegundos, por defecto 0

Estas configuraciones son opcionales y puede que no se tengan en cuenta por los navegadores

10.1. Métodos el API

opciones de geolocalización

Por ejemplo, solicitar una alta precisión, un timeout de dos segundos y no cachear las peticiones:

```
navigator.geolocation.getCurrentPosition(success, error, {  
    enableHighAccuracy: true  
    timeout: 2000  
    maximumAge: 0  
});
```

10.2. API de Google Maps

<https://developers.google.com/maps/?hl=es>

Una vez que disponemos de los datos de localización podemos mostrarlos en un mapa, a través del API de Google (por ejemplo)

El API de JavaScript de Google Maps está en su versión 3 permite crear aplicaciones de HTML5 para utilizar tanto en plataformas de escritorio como de dispositivos móviles

Ejemplos de utilización del API de JavaScript:

<https://developers.google.com/maps/documentation/javascript/examples/?hl=es>

10.2. API de Google Maps

Para utilizar el API tendremos que añadir un nuevo script (no es necesario que vaya en la etiqueta <head>)

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Se indicará “sensor=true” para navegadores que dispongan de sensor GPS

En el documento HTML tendremos un bloque para poder “pintar” el mapa

```
<div id="map-canvas"></div>
```


10.2. API de Google Maps

En JavaScript podremos pintar el mapa utilizando la API de Google Maps

```
function createMap(position) {  
    var latlng = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);  
    var myOptions = {  
        zoom: 15,  
        center: latlng,  
        mapTypeControl: false,  
        navigationControlOptions: {style: google.maps.NavigationControlStyle.SMALL},  
        mapTypeId: google.maps.MapTypeId.ROADMAP  
    };  
    map = new google.maps.Map(document.getElementById("map-canvas"), myOptions);  
  
    return map;  
}
```

10.2. API de Google Maps

Además podremos indicar con un marcador la posición

```
function addMarker(position, map) {  
    var latlng = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);  
  
    var marker = new google.maps.Marker({  
        position: latlng,  
        map: map,  
        title: "¡Usted está aquí!"  
    });  
}
```

10.2. API de Google Maps

Para utilizar estas funciones tendremos que obtener la posición con el API de geolocalización de HTML5 y pasarlas a las funciones de “crear mapa” y “añadir marcador”

```
map = createMap(position);  
addMarker(position, map);
```

10.3. Ejercicio práctico

Utilizando los servicios de geolocalización, realizar las siguientes tareas:

- ✓ Solicitar las coordenadas actuales, y mostrar dichas coordenadas (y la precisión) tanto en formato texto como un punto en el mapa
- ✓ Acceder a la aplicación desde un navegador móvil
- ✓ Realizar las modificaciones para obtener datos extra como velocidad, altura, ...

<http://www.arkaitzgarro.com/html5/capitulo-18.html#ej13>

11. Almacenamiento local

Hasta ahora, el almacenamiento de datos en la web se realizaba en el servidor, y era necesario algún tipo de conexión (o sincronización) con el cliente para trabajar con estos datos

Con HTML5 podemos almacenar datos en el cliente:

- ✓ **Web Storage:** <http://www.w3.org/TR/webstorage/>
Los datos se almacenan en parejas de clave/valor. Ampliamente soportado por todos los navegadores
- ✓ **Web SQL Database:** <http://www.w3.org/TR/webdatabase/>
Sistema de almacenamiento basado en SQL. **NO** va a ser mantenido en el futuro, pero actualmente su uso está muy extendido y es soportado por Chrome, Safari y Opera
- ✓ **IndexedDB:** <http://www.w3.org/TR/Indexeddb/>
Sistema de almacenamiento basado en objetos. Soportado por Chrome, Firefox e Internet Explorer

11.1. Web Storage

Este API de almacenamiento ofrece dos posibilidades para guardar datos en el navegador:

- ✓ `sessionStorage`: mantiene los datos durante la sesión actual (mientras la ventana o pestaña se mantenga abierta)
- ✓ `localStorage`: almacena los datos hasta que sean eliminados explícitamente por la aplicación o el usuario

Ambos modos de almacenamiento se encuentran relacionados con el dominio que los ha creado

11.1. Web Storage

sessionStorage

- ✓ Mantiene un área de almacenamiento disponible a lo largo de la duración de la sesión de la ventana o pestaña
- ✓ La sesión persiste mientras que la ventana permanezca abierta y sobrevive a recargas de página
- ✓ Si se abre una nueva página en una pestaña o ventana, una nueva sesión es inicializada, por lo que no es posible acceder a los datos de otra sesión

11.1. Web Storage

localStorage

- ✓ Es capaz de almacenar los datos por dominio y persistir más allá de la sesión actual, aunque el navegador se cierre o el dispositivo se reinicie

NOTA: Una nueva ventana abierta utilizando el método `window.open()`, pertenece a la misma sesión

11.1. Web Storage

Tanto sessionStorage como localStorage forman parte del Web Storage, por lo que comparten el mismo API:

readonly attribute unsigned long **length**;
getter DOMString **key**(in unsigned long index);
getter DOMString **getItem**(in DOMString key);
setter creator void **setItem**(in DOMString key, in any data);
deleter void **removeItem**(in DOMString key);
void **clear**();

```
sessionStorage.setItem('twitter', '@starkyhach');  
alert( sessionStorage.getItem('twitter') ); // muestra @starkyhach
```

11.1. Web Storage

Tal y como se indica en el API, el método `getItem` siempre devuelve un **String**, por lo que si intentamos almacenar un objeto, el valor devuelto será "[Object object]"

El mismo problema ocurre con los números, por lo que es importante tenerlo en cuenta para evitar posibles errores:

```
sessionStorage.setItem('total', 120);  
function calcularCosteEnvio(envio) {  
    return sessionStorage.getItem('total') + envio;  
}
```

```
alert(calcularCosteEnvio(25)); // Devuelve 12025
```

11.1. Web Storage

Eliminar datos del almacenamiento local

- ✓ `removeItem`, toma como parámetro el nombre de la clave a eliminar (el mismo que utilizamos en `getItem` y `setItem`), para eliminar un ítem en particular
- ✓ `clear`, elimina todas las entradas del objeto.

```
sessionStorage.setItem('twitter', '@starkyhach');  
sessionStorage.setItem('flickr', 'starky.hach');  
alert( sessionStorage.length );           // Muestra 2  
sessionStorage.removeItem('twitter');  
alert( sessionStorage.length );           // Muestra 1  
sessionStorage.clear();  
alert( sessionStorage.length );           // Muestra 0
```

11.1. Web Storage

Almacenamiento de objetos

Una manera de almacenar objetos es utilizando JSON. Como la representación de los objetos en JSON puede realizarse a través de texto, podemos almacenar estas cadenas de texto y recuperarlas posteriormente para convertirlas en objetos

```
var videoDetails = {  
    title : 'Matrix',  
    author : ['Andy Wachowski', 'Larry Wachowski'],  
    description : 'Wake up Neo, the Matrix has you...',  
    rating: '-2'  
};  
sessionStorage.setItem('videoDetails', JSON.stringify(videoDetails) );  
var videoDetails = JSON.parse(sessionStorage.getItem('videoDetails'));
```

11.1. Web Storage

Eventos de almacenamiento

Web Storage es que incluye una serie de eventos que nos indican cuándo se ha producido un cambio en los datos almacenados

Estos eventos **no** se lanzan en la ventana actual donde se han producido los cambios, sino en el resto de ventanas donde los datos pueden verse afectados

- ✓ Para sessionStorage, son lanzados en los iframe dentro de la misma página, o en las ventanas abiertas con window.open()
- ✓ Para localStorage, todas las ventanas abiertas con el mismo origen (protocolo + host + puerto) reciben los eventos

11.1. Web Storage

Eventos de almacenamiento

Cuando se lanzan los eventos, éstos contienen toda la información asociada con el cambio de datos:

```
StorageEvent {  
    readonly DOMString key;  
    readonly any oldValue;  
    readonly any newValue;  
    readonly DOMString url;  
    readonly Storage storageArea;  
};
```

storageArea hace referencia al objeto sessionStorage o localStorage

11.1. Web Storage

Eventos de almacenamiento

Estos eventos se lanzan dentro del objeto window:

```
function handleStorage(event) {  
    event = event || window.event; // support IE8  
    if (event.newValue === null) { // it was removed  
        // Do something  
    } else {  
        // Do something else  
    }  
}
```

```
window.addEventListener('storage', handleStorage, false);  
window.attachEvent('storage', handleStorage);
```

11.2. Ejercicio práctico

Implementad las siguientes funcionalidades utilizando `SessionStorage` y `LocalStorage`:

- ✓ Crear una caja de texto, a modo de editor de contenidos, y utilizando `SessionStorage` almacenar el contenido de la caja en cada pulsación de tecla. Si la página es recargada, el último contenido almacenado debe mostrarse en la caja de texto. Comprobad que cerrando la pestaña actual, o abriendo una nueva ventana, los datos no se comparten
- ✓ Modificar el código anterior para utilizar `LocalStorage`. Comprobad que en este caso, aunque cierre la ventana o abra una nueva, los datos se mantienen. Añadir la posibilidad de actualizar el resto de ventanas abiertas, cada vez que se modifique el valor de la caja de texto en cualquiera de ellas

11.3. Web SQL

Web SQL **no** forma parte de la especificación de HTML5, pero es ampliamente utilizado para el desarrollo de aplicaciones web

<http://www.w3.org/TR/webdatabase/>

Se trata de una base de datos basada en SQL, más concretamente en SQLite

La utilización del API se resume en tres simples métodos:

- ✓ openDatabase: abrir (o crear y abrir) una base de datos en el navegador del cliente
- ✓ transaction: iniciar una transacción
- ✓ executeSql: ejecutar una sentencia SQL

11.3. Web SQL

El API de Web SQL es asíncrono, por lo que es necesario tener cuidado con el orden en el que se ejecutan las sentencias SQL

Sin embargo, las sentencias SQL se encolan y son ejecutadas en orden, por lo que podemos estar tranquilos en ese sentido: podemos crear tablas y tener la seguridad que van a ser creadas antes de acceder a los datos

11.3. Web SQL

openDatabase

Al abrir la base de datos por primera vez, ésta es creada automáticamente

```
var db = openDatabase('mydb', '1.0', 'My first database', 2 * 1024 * 1024);
```

Es necesario especificar el **número de versión** de base de datos con el que se desea trabajar, y si no especificamos correctamente éste número de versión, es posible obtener error del tipo `INVALID_STATE_ERROR`

El valor de retorno de esta función es un objeto que dispone de un método `transaction`, a través del cual vamos a ejecutar las sentencias SQL.

11.3. Web SQL

openDatabase

```
var db = openDatabase('mydb', '1.0', 'My first database', 2 * 1024 * 1024);
```

Los valores que pasamos a esta función son los siguientes:

- ✓ El nombre de la base de datos
- ✓ El número de versión de base de datos con el que deseamos trabajar
- ✓ Un texto descriptivo de la base de datos
- ✓ Tamaño estimado de la base de datos, en bytes

La última versión de la especificación incluye un quinto argumento en la función, pero no es soportado por muchos navegadores

11.3. Web SQL

transaction

La idea de utilizar transacciones, en lugar de ejecutar las sentencias directamente, es por la posibilidad de realizar rollback, es decir, volver al estado inicial

```
db.transaction(function (tx) {  
    tx.executeSql('DROP TABLE foo');  
    tx.executeSql('INSERT INTO foo (id, text) VALUES (1, "foobar")');  
}, function (err) {  
    alert(err.message);  
});
```

En ocasiones se fuerza un rollback indicando una consulta errónea (dirty trick), ya que no existe un método “abort”

11.3. Web SQL

executeSql

El método `executeSql` es utilizado tanto para sentencias de escritura como de lectura

Incluye protección contra ataques de inyección SQL y proporciona llamadas a métodos (callback) para procesar los resultados devueltos por una consulta SQL

```
db.transaction(function (tx) {  
    tx.executeSql(sqlStatement, arguments, callback, errorCallback);  
});
```

11.3. Web SQL

executeSql

`tx.executeSql(sqlStatement, arguments, callback, errorCallback);`

- ✓ **sqlStatement:** indica la sentencia SQL a ejecutar: creación de tabla, insertar un registro, realizar una consulta, etc
- ✓ **arguments:** corresponde con un array de argumentos que pasamos a la sentencia SQL. El propio método se ocupa de prevenir inyecciones SQL
- ✓ **callback:** función a ejecutar cuando la transacción se ha realizado de manera correcta. Toma como parámetros la propia transacción y el resultado de la transacción
- ✓ **errorCallback:** función a ejecutar cuando la transacción se ha producido un error en la sentencia SQL. Toma como parámetros la propia transacción y el error producido

11.3. Web SQL

executeSql

```
db.transaction(function (tx) {  
    tx.executeSql('SELECT * FROM foo WHERE id = ?',  
        [5],  
        function callback(tx, results) {  
            var len = results.rows.length, i;  
            for (i = 0; i < len; i++) {  
                alert(results.rows.item(i).text);  
            }  
        },  
        function errorCallback(tx, error) {  
            alert(error.message);  
        }  
    );  
});
```


11.3. Web SQL

executeSql

La función de callback recibe como argumentos la transacción (de nuevo) y un objeto que contiene los resultados

Este objeto contiene una propiedad rows, donde **rows.item(i)** contiene la representación de la fila concreta

Si nuestra tabla contiene un campo que se llama nombre, podemos acceder a dicho campo de la siguiente manera:

```
results.rows.item(i).nombre
```

11.3. Web SQL

Crear tablas

La primera tarea a realizar cuando trabajamos con una base de datos es crear las tablas necesarias para almacenar los datos

Este proceso se realiza a través de una sentencia SQL, dentro de una transacción:

```
db = openDatabase('tweetdb', '1.0', 'All my tweets', 2 * 1024 * 1024);
db.transaction(function (tx) {
    tx.executeSql('CREATE TABLE IF NOT EXISTS tweets(id, user, date, text)', [], getTweets);
});
```

11.3. Web SQL

Insertar datos

El siguiente paso es insertar los datos correspondientes en las tablas. En este ejemplo la función `getTweets` realizaría una llamada AJAX para obtener ciertos datos

```
function getTweets() {  
    var tweets = $.ajax({...});  
    $.each(tweets, function(tweet) {  
        db.transaction(function (tx) { //CADA TWEET EN UNA TRANSACCION → Procesa todas  
            var time = (new Date(Date.parse(tweet.created_at))).getTime();  
            tx.executeSql('INSERT INTO tweets (id, user, date, text) VALUES (?, ?, ?, ?)',  
                [tweet.id, tweet.from_user, time / 1000, tweet.text]);  
        });  
    });  
}
```

11.3. Web SQL

Obtener datos

Una vez que los datos se encuentran almacenados en la base de datos, sólo nos queda consultarlos y proporcionar la función que los procese

```
db.transaction(function (tx) {  
    tx.executeSql('SELECT * FROM tweets WHERE date > ?', [time],  
        function(tx, results) {  
            var html = [], len = results.rows.length;  
            for (var i = 0; i < len; i++) {  
                html.push('<li>' + results.rows.item(i).text + '</li>');  
            }  
            tweetEl.innerHTML = html.join("");  
        });  
});  
}
```

11.4. Ejercicio práctico

Crear un objeto que encapsule una base de datos WebSQL, que nos permitir acceder a una base de datos para añadir, modificar, eliminar y obtener registros. Dicha base de datos va a almacenar tweets procedentes de Twitter, que tienen asociado el hashtag #html5. Los requisitos son los siguientes:

- ✓ Disponer de una tabla para almacenar los tweets. Los campos mínimos son: identificador del tweet, texto, usuario, y fecha de publicación
- ✓ Disponer de una tabla para almacenar los usuarios que publican los tweets. Esta tabla debe estar relacionada con la anterior. Los campos mínimos son: identificador del usuario, nombre e imagen
- ✓ Crear un método addTweet que dado un objeto que corresponde con un tweet, lo almacene en la base de datos. Almacenar el usuario en caso de que no exista, o relacionarlo con el tweet si existe

11.4. Ejercicio práctico

- ✓ Crear un método `removeTweet` que dado un identificador de tweet, lo elimine de la base de datos. Éste método debe devolver el tweet eliminado
- ✓ Crear un método `updateTweet` que dado un objeto que corresponde con un tweet, actualice los datos correspondientes al tweet en la base de datos
- ✓ Crear un método `getTweets` que dado un parámetro de fecha, me devuelva todos los tweets posteriores a esa fecha. Cada tweet debe incluir sus datos completos y el usuario que lo creó

Obtener los últimos 25 tweets que tienen como hashtag `#html5` de la siguiente consulta a Twitter: `http://search.twitter.com/search.json?q=%23html5&rpp=25&result_type="recent"`

<https://dev.twitter.com/docs/api/1/get/search>

11.5. IndexedDB

IndexedDB **no** es una base de datos relacional, sino que se podría llamar un almacén de objetos

En la base de datos existirán almacenes y en su interior añadimos objetos

```
{  
  id:21992,  
  nombre: "Memoria RAM"  
}
```

11.5. IndexedDB

Los pasos que seguiremos en IndexedDB serán:

- ✓ Abrir una base de datos indicando su nombre y la versión concreta
- ✓ Crear los almacenes de objetos, que es muy parecido a un archivador con índices, que nos permite encontrar de una manera muy rápida el objeto que buscamos
- ✓ Almacenar cualquier tipo de objeto con el índice que definamos
- ✓ No importa el tipo de objeto que almacenemos, ni tienen que tener las mismas propiedades

11.5. IndexedDB

Abrir la base de datos

Al abrir una base de datos por primera vez se creará, después podremos trabajar directamente con el objeto que nos devuelve

Al igual que en Web SQL, las peticiones a la base de datos se realizan de manera asíncrona

Previamente, necesitaremos hacer comprobaciones sobre el navegador

```
window.indexedDB = window.indexedDB || window.webkitIndexedDB || window.mozIndexedDB;  
if ('webkitIndexedDB' in window) {  
    window.IDBTransaction = window.webkitIDBTransaction;  
    window.IDBKeyRange = window.webkitIDBKeyRange;  
}
```

11.5. IndexedDB

Abrir la base de datos

Para abrir la base de datos únicamente necesitamos un nombre:

```
var request = indexedDB.open('videos');
request.onerror = function () {
    console.log('failed to open indexedDB');
};
request.onsuccess = function (event) {
    // handle version control
    // then create a new object store
};
```

11.5. IndexedDB

Abrir la base de datos

Ahora que la base de datos esta abierta (y asumiendo que no hay errores), se ejecutará el evento onsuccess

Antes de poder crear almacenes de objetos, tenemos que tener en cuenta los siguiente:

- ✓ Necesitamos un manejador para poder realizar transacciones de inserción y obtención de datos
- ✓ Hay que especificar la versión de la base de datos. Si no existe una versión definida, significa que la base de datos no está aún creada

11.5. IndexedDB

Abrir la base de datos

El método `onsuccess` recibe como parámetro un evento, dentro de este objeto, encontramos una propiedad llamada `target`, y dentro de ésta otra propiedad llamada `result`, que contiene el resultado de la operación

```
var db = null;
var request = indexedDB.open('videos');
request.onsuccess = function (event) {
    db = event.target.result;
    // handle version control
    // then create a new object store
};
```

11.5. IndexedDB

Abrir la base de datos

```
request.onerror = function (event) {  
    alert('Something failed: ' + event.target.message);  
};
```

En IndexedDB, los errores que se producen escalan hasta el objeto request

Si un error ocurre en cualquier petición (por ejemplo una consulta de datos), se ejecutaría “request.onerror”