Javascript

Javascript

- JavaScript es un lenguaje ligero e interpretado con capacidades básicas de orientación a objetos.
- Este lenguaje es soportado por la **mayor parte de navegadores** web actuales.
- Permitiendo contenido “ejecutable” en las páginas web consiguiendo de esta manera interactuar con el navegador, el usuario e incluso generar contenido de manera dinámica.

- Sintácticamente es un lenguaje muy parecido a c, c++ o Java.
- Es un lenguaje débilmente tipado.

Javascript

- Para introducir un script en una web:

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

- También se puede enlazar con un fichero externo:

```
<script type="text/javascript"
src="unDirectorio/unArchivoJavaScript.js"></script>
```

- Las sentencias en javascript pueden terminar en punto y coma o en salto de línea.
- Es aconsejable el uso de punto y coma para separar sentencia.
  - Para escribir comentarios:

```
// Esto sería un comentario de una sola línea.
/* Esto puede ser un comentario de más de una línea. */
```

- **Variables y tipos de datos**

- Para declarar una variable simplemente debemos asignarle un valor:

```
i = 2; // Crea una variable y le asigna el valor 2.
```

- Aunque no es necesario es recomendable el uso de la palabra reservada var para indicar que vamos a definir una variable:

```
var i = 2;
```

- Javascript es un lenguaje débilmente tipado y por tanto una variable puede contenter diferentes tipos de datos:

```
var a;
a = 3;
a = “hola”;
```

- **Variables y tipos de datos**
  - Los tipos de datos primitivos o básicos que existen en javascript son números, cadenas de texto y booleans.
  - Los números pueden ser:
    - Enteros [-](1-9)(0-9)* ejemplos: 23, 1450, -1
    - Octales [-]0(0-7)* ejemplos: 0077, -0123
    - Hexadecimal [-]0(x|X)(0-9|a-f|A-F)*
      - ejemplos:
        - 0xff, -0xCAFE
    - Coma flotante [(+|-)][digits][.digits][(E|e) [(+|-)]digits]
      - ejemplos:
        - 3.14, -3.1416, .033,6.01e+3, 6.01E-24

- **Variables y tipos de datos**
  - Las cadenas pueden definirse usando comillas dobles o simples:

    ```
    var str = “hola”;
    var str = ‘hola’;
    ```
  - No se debería combinar definiciones con comillas simples y dobles. Se debe adoptar un convenio y seguirlo.
  - Para definir un booleano disponemos de dos palabras reservadas true y false:

    ```
    var cierto = true;
    var falso = false;
    ```

- Expresiones y operadores

P	A	Operator	Operand Type(s)	Operation Performed
0	L	.	object, property	property access
	L	[]	array, integer	array index
	L	()	function, args	function call
1	R	++	number	pre-or-post increment (unary)
	R	--	number	pre-or-post decrement (unary)
	R	-	number	unary minus (negation)
	R	~	integer	bitwise complement (unary)
	R	!	Boolean	logical complement (unary)
	R	typeof	any	return data type (unary)
	R	new	constructor call	create new object (unary)
	R	void	any	return undefined value (unary)

- Expresiones y operadores

2	L	*, /, %	numbers	multiplication, division, remainder
3	L	+, -	numbers	addition, subtraction
	L	+	strings	string concatenation
4	L	<<	integers	left shift
	L	>>	integers	right shift with sign-extension
	L	>>>	integers	right shift with zero extension
5	L	<, <=	numbers or strings	less than, less than or equal
	L	>, >=	numbers or strings	greater than, greater than or equal
6	L	==	primitive types	equal (have identical values)
	L	!=	primitive types	not equal (have different values)
	L	==	reference types	equal (refer to same object)
	L	!=	reference types	not equal (refer to different objects)
7	L	&	integers	bitwise AND

- Expresiones y operadores

8	L	`^`	integers	bitwise XOR
9	L	`|`	integers	bitwise OR
10	L	`&&`	Booleans	logical AND
11	L	`||`	Booleans	logical OR
12	R	`?:`	Boolean, any, any	conditional (ternary) operator
13	R	`=`	variable, any	assignment
	R	`*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=`	variable, any	assignment with operation
14	L	`,`	any	multiple evaluation

- Sentencias de control
  - Condicional

```
if(expresión){
        sentencias;
} else {
        sentencias;
}
```

  - Ejemplo:

```
if (a < 30){
        alert(“hola”);
} else {
        alert(“adios”);
}
```

- Sentencias de control
  - While

```
while(condicion){
        sentencias;
}
```

  - Ejemplo

```
var contador = 0;
while(contador < 25){
        contador++;
        alert("contador "+contador);
}
```

- Sentencias de control
  - For

```
for (iniciar; test; incremento){
        Sentencias;
}
```

  - Ejemplo

```
for(count=0;count<10;count++){
        alert("count "+count);
}
```

- Sentencias de control
  - For in

```
for (variable in object){
        Sentecias;
}
```

  - Ejemplo

```
for (prop in myObject){
        alert(prop);
}
```

- Sentencias de control
  - La sentencia “break” es usada para romper un bucle de cualquier tipo de forma prematura.
    - Fuera de un bucle su uso producirá un error.
  - La sentencia “continue” es usada para romper la iteración actual dentro de un bucle y continuar la siguiente.
    - Su uso fuera de bucle producirá un error.

- Funciones
  - Las funciones se definen mediante la palabra reservada “function”:
    function nombreFuncion(param1,…, paramN){..}
  - Una función puede devolver un valor mediante el uso de la palabra reservada “return”
  - Para invocar una función simplemente debemos poner el nombre de la misma y entre paréntesis colocar los parámetros que se vayan a pasar la función (puede ser vacío).
  - Las funciones en javascript además de definirse y poder invocarse también pueden ser asignadas a una variable y ser pasadas como argumentos a otras funciones.

- Objetos
  - Los objetos en javascript son conjuntos de datos accesibles mediante un nombre. A esos datos se les llama propiedades.
  - La manera de crear un objeto vacío será:
    var persona = new Object();
  - Podemos asignar ahora propiedades al objeto mediante el operador “.”:
    persona.nombre=“Hugo”;
    persona.verNombre=function()
    {alert(this.nombre);};

- Objetos
  - Para crear un objeto con parámetros (función constructora):

```
function Persona(nombre, apellidos){
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.showNombre = function(){
        return this.nombre + " " + this.apellidos; };
};
var hs= new Persona("Hugo", "Sirvent");
```

- Objetos
  - Para crear métodos en los objetos simplemente debemos asignar a propiedades funciones, ya que estas son un tipo de dato más en javascript.
  - Para invocar un método de un objeto procederemos como sigue:
    ```
    var jl = new Persona("Hugo", "Sirvent");
    jl.showNombre();
    ```
  - En javascript no hay
    - Clases.
    - Herencia basada en clases.
    - Visibilidad (todo es público, sin embargo se puede simular).

- Arrays
  - Para crear un array con datos iniciales:
    var ar = new Array(“hola”,1,true);
  - Para crear un array vacío (los elementos serán undefined) de un tamaño dado X:
    var ar = new Array(X);
  - Para acceder a las posiciones del array (desde 0 hata X-1)
    ar [0] = “hola”;

- Arrays
  - Los arrays en JavaScript son dinámicos.
  - Al usar una posición no definida previamente el array crece aunque solamente ocupan memoria aquellas posiciones a las que se les ha asignado valor.
  - Se debe tener cuidado a la hora de recorrer arrays en javascript mediante bucles for accediendo por posición.

```
var a = new Array();
a[0] = “hola”;
a[3] = “adiós”;
for(c=0;c<a.length;c++){
        alert(a[c]);
}
for(prop in a){
        alert(a[prop]);
}
```

- Arrays asociativos/objetos
  - Aunque hayamos visto los Arrays y los objetos como algo distinto en javascript realmente son lo mismo.
  - Podemos definir un array de la siguiente manera:

```
var arrayObj = new Object();
arrayObj[0] = “hola”;
arrayObj[1] =“adiós”;
```

- Arrays asociativos/objetos
  - Igualmente podemos acceder a las propiedades de un objeto de la siguiente manera:

```
function f(nombre, apellidos){
        this.nombre = nombre;
        this.apellidos = apellidos;
}
var a = new f("Hugo", "Sirvent");
alert(a["nombre"]);
```

- # Notación literal
  - Mediante la notación literal de objetos podemos definir objetos mediante una sintáxis más sencilla y legible:

```
var obj = new Object();
obj.nombre = “Pepelu”;
obj.showNombre = function() {
        alert(this.nombre);
};
var obj = {
        nombre : “Pepelu”,
        showNombre: function() {
                alert(this.nombre);
        };
}
```

- # JSON (JavaScript Object Notation)
  - Es un **formato ligero** de intercambio de información fácil de leer y escribir por parte de los humanos (http://json.org).
  - Se trata de un **subconjunto** de la **notación literal** de objetos de JavaScript.
  - En los últimos tiempos ha tenido una gran adopción en detrimento de xml y otros formatos de intercambio de información ya que es fácil (tanto a nivel computacional como semántico) de generar y de interpretar posteriormente.
  -

JSON (JavaScript Object Notation)

- La anatomía de un marcado JSON es casi idéntica a la de un objeto JavaScript:

```
{
        “id” : 1,
        “nombre” : “ES”,
        “imagen” : {
                “url” : “imagenes/1.jpg”
        },
        “fechaAlta” : “2012-12-11”
}
```

- En los pares nombre-valor el nombre siempre va entre comillas.
- Puede representar seis tipos de valores:
  - objetos,
  - arrays,
  - números,
  - cadenas,
  - booleanos,
  - null.
- Las fechas no son reconocidas como un tipo de objeto propio.

- Event Handlers
  - Un manejador de eventos o event handler es una función especial que es usada cuando se usa javascript en la parte cliente.
  - En el siguiente código definimos una alerta cuando se hace click sobre el link:

```
<a href onclick="alert('hola')" >click</a>
```

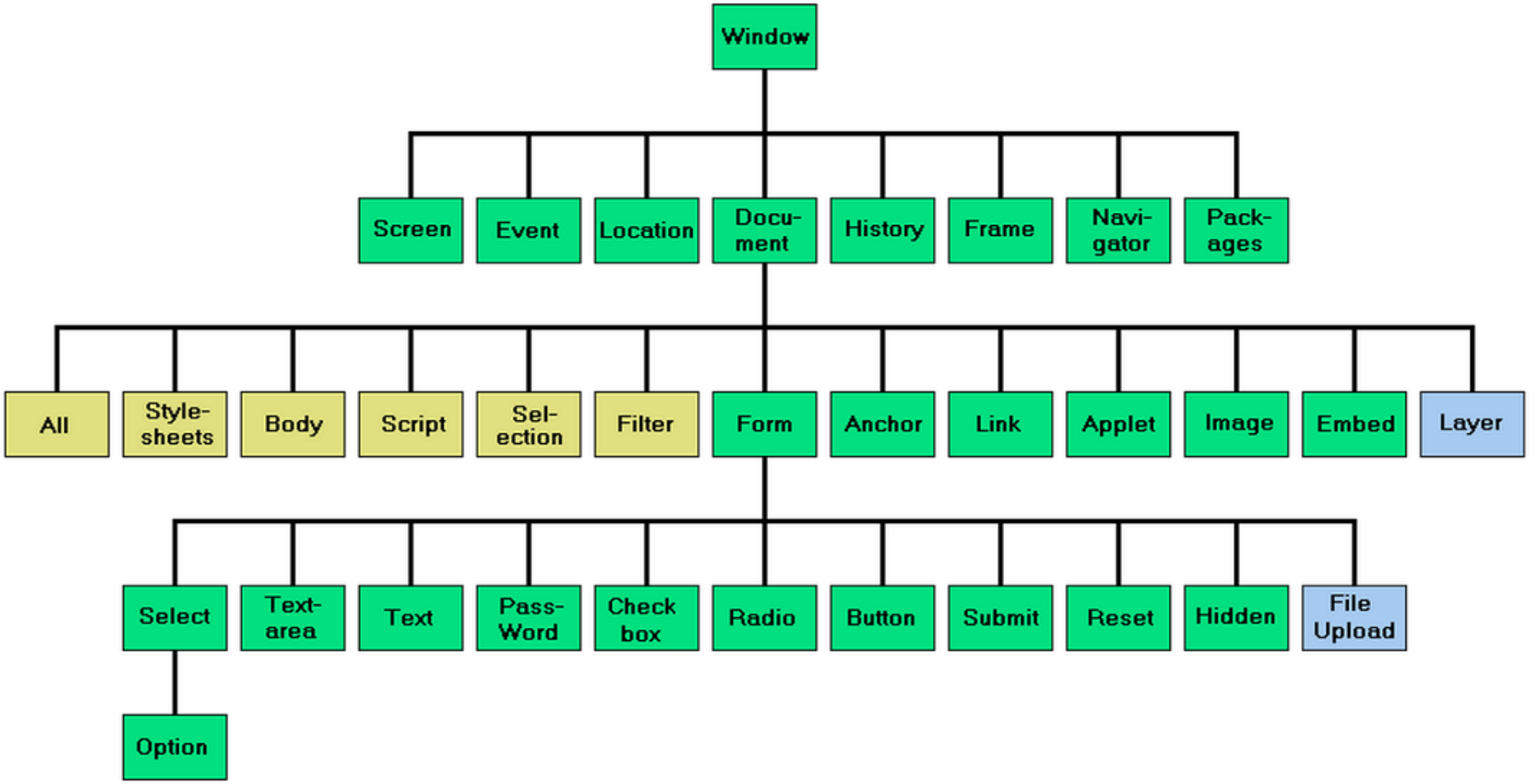
- Event Handlers
  - En el manejador de evento se puede introducir código javascript o realizar una llamada a una función.
  - Se pueden asignar funciones para atender eventos directamente desde el código de manera que el html queda completamente limpio de javascript, **javascript no intrusivo.**
  - Cada elemento html tiene una serie de eventos a los que puede atender.
  - En la siguiente url se puede ver una serie completa de propiedades y eventos que soportan todos los elementos html:

    http://www.w3schools.com/jsref/dom_obj_event.asp

- DOM (Document Object Model)
  - Es un API usada para manipular documentos html y xml.
  - Con este API cada documento se convierte en una jerarquía de nodos.
  - Esa **jerarquía de nodos** muestra las relaciones existentes entre las diferentes partes del documento y nos **permite navegar** a través de ella.
  - DOM no es exclusivo de los navegadores y javascript.
  - Está presente casi en todos los lenguajes usados habitualmente hoy en día.

- BOM (Browser Object Model)
  - Permite interactuar con las ventanas y elementos del navegador modificando sus propiedades.
  - A diferencia de **DOM** que **es un estándar** el **BOM no** es estandarizado y por tanto cada navegador / organización ofrece un BOM distinto (en parte por lo menos).
  - Al cargar una página el navegador genera una serie de objetos accesibles mediante javascript en esa página.
  - Algunos de los objetos ofrecidos son:
    - **Window**: Objeto principal. Es el padre del resto de elementos de navegador.
    - **Navigator**: Contiene información sobre el navegador, tipos de objetos mime soportados etc
    - **Location**: Ofrece información sobre la url que estamos visitando.
    - **History**: Historial de navegación.
    - **Document**: Contenedor de todos los elementos que se han creado en la página.

Javascript


Window
Screen
Event
Location
Docu-ment
History
Frame
Navi-gator
Pack-ages
All
Style-sheets
Body
Script
Sel-ection
Filter
Form
Anchor
Link
Applet
Image
Embed
Layer
Select
Text-area
Text
Pass-Word
Check box
Radio
Button
Submit
Reset
Hidden
File Upload
Option

- DOM – Interactuando
  - Con DOM podremos acceder a las diferentes partes de un documento html como un conjunto de nodos con atributos e hijos.

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el medio.</p>
<ul>
<li>Primer elemento de la lista</li>
<li>Otro elemento de la lista</li>
</ul>
</body>
```

  - El elemento [body] sería el primer nodo y tendría como hijos a [p] y [ul], a su vez [p] tendría un elemento [a] como hijo…
  - Los elementos pueden tener hijos y atributos y podemos asociarlos a los
  - diferentes tags del documento html body, p,a ,img, div, span…
  - Además existen los nodos de texto los cuales no tienen ni hijos ni atributos.

- DOM – Interactuando
  - Accediendo a elementos por id
  - El id de un elemento es único (no puede haber o no debe haber dos elementos con el mismo id)
  - Es la manera ideal para acceder a un nodo en particular.
  - Usaremos document.getElementById(id):

```
<a id="unLink">texto del link</a>
...
var link = document.getElementById("unLink");
```

  - El nombre de un elemento no es único, podemos tener tantos elementos con el mismo nombre como necesitemos.
  - Usaremos document.getElementsByName(name):

```
<a name="nombre">texto del link 1</a>
<a name="nombre">texto del link 2</a>
...
var links= document.getElementsByName("nombre");
```

- # DOM – Interactuando
  - También puede resultar útil acceder a todos los elementos iguales (del mismo tag):
  - Usaremos getElementsByTagName(tagName):

    ```
    <a>texto del link 1</a>
    <a>texto del link 2</a>
    …
    var links = document.getElementsByTagName(“a”);
    ```
  - Para crear elementos de un tipo usaremos la función createElement(“tipoElemento”) que crea un elemento del tipo indicado y devuelve una referencia al mismo:

    ```
    var link = document.createElement(“a”);
    ```

DOM – Interactuando

- Ya comentamos que a la hora de asignar eventos es mejor realizarlo desde el código javascript, de manera no intrusiva:

```
<head>
        <script type="text/javascript" src="events.js"></script>
</head>
<body onload="loadEvents();">
        <a id="elLink" href="#">haz click</a>
</body>
```

- El fichero events.js podría contener algo como esto:

```
function linkClick(){
    alert("se ha lanzado el evento click");
    return false;
}
function loadEvents(){
    var link = document.getElementById("elLink");
    link.onclick = linkClick;
}
```

- # Ámbitos variables
  - Una variable (también una función) global puede ser llamada desde cualquier lugar de nuestra aplicación javascript.
  - Una variable o función definida dentro de otra función solamente es accesible desde esa función.
  - Si definimos una variable dentro de una función y colisiona en nombre con una global pueden pasar dos cosas:
    - Se ha definido usando la palabra reservada **var**.
      - En este caso siempre se accederá a la variable local de la función, para todos los accesos a esa variable (incluidos los anteriores a la definición).
    - Si **no** se usa la palabra reservada **var** se accede siempre a la variable global.

- This
  - La palabra reservada this se usa para referirse al objeto en el cual nos encontramos.
  - Para el caso de una función global, el propietario es la propia página, en un navegador web será el objeto window:

```
function foo(){
        alert(this);
}
foo(); // window
```

  - Llamar a this desde un método de un objeto da, como resultado, dicho objeto:

```
var foo = {
        bar: function(){
                return this;
        }
}
alert(foo.bar()); // object foo
```

- Closures
  - La wikipedia define las closures como:
    - “...es una función que es evaluada en un entorno conteniendo una o más variables dependientes de otro entorno. Cuando es llamada, la función puede acceder a estas variables...”
    - Los closures son funciones/métodos que hacen uso de alguna variable no definida en su cuerpo y que para su correcto funcionamiento deben ser “encerrados” dentro de un ámbito que proporcione las referencias válidas a dichas variables.
    - En javascript las closures se construyen mediante **funciones anidadas**.

- Closures
  - Las closures son especialmente útiles para:
    - Asociar funciones con métodos de instancia de otros objetos.
    - Programar eventos de botones,
    - Links…
    - Encapsular de forma “privada” ciertas funcionalidades.
    - Ejemplo:

```
function master(){
    var nombre = 'Javascript';
    function muestraNombre(){
        alert(nombre);
    }
    muestraNombre();
}
```

Funciones Anonimas

- Como su nombre indica son funciones que no tienen nombre.
- Pueden ser autoejecutables:

```
(
        function(){
                alert('me ejecuto sola');
        }
)();
```

- O pueden ser asignadas a una variable:

```
var sumar = function(a,b){
        return a+b;
}
alert(sumar(1,1));
```

- Comprobar código Javascript online:
  - http://www.webtoolkitonline.com/javascript-tester.html
  - http://js.do/
  - https://jsfiddle.net/

APIs

- Canvas
- Drag & Drop
- History
- Inline Editing
- Messaging
- Offline Apps
- Video & Audio
- Geolocation
- Local Storage
- Selectors
- Server Events
- Web Sockets
- Workers

Geolocation

- El API de geolocalización nos permite obtener la posición aproximada en la que se encuentra el navegador desde el cual se accede a nuestra página.
- Entre la información que devuelve la posición estarán la latitud, la longitud y la altitud.
- Para acceder a la posición simplemente se la solicitaremos al navegador.
  - Antes de obtenerla se le solicitará permiso al usuario.

```
navigator.geolocation.getCurrentPosition(
        function(pos) {
                alert( 'latitud: ' + pos.coords.latitude +'\nlongitud: ' + pos.coords.longitude);
        },
        function(error) {
                var msg = 'Error desconocido';
                if(error.code == 1) {
                        msg = 'Permiso denegado';
                } else if(error.code == 2) {
                        msg = 'Posición no disponible';
                } else if(error.code == 3) {
                        msg = 'Timeout';
                }
                console.log('Error: ' + error.code + ', ' + msg);
        }
);
```

- Geolocation
  - ¿Qué hace este fragmento de código?

```
<script src="https://maps.google.com/maps/api/js?sensor=false"></script>
var latlng = new google.maps.LatLng(lat, lng);
var opciones = {
        zoom: 19,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.SATELLITE
};
var map = new google.maps.Map(document.getElementById("mapa"), opciones);

<div id=”mapa”></div>
```

- # Canvas
  - Sirve para dibujar gráficos al vuelo usando JavaScript.
  - Se trata de un área rectangular de la que podemos controlar cada uno de sus píxeles.
  - Así, nos va a permitir crear gráficos personalizados en el cliente.
  - Su API ofrece toda una serie de métodos para pintar líneas, rectángulos, círculos, imágenes, texto...

```
// Obtenemos el elemento canvas.
var canvas = document.getElementById('unCanvas');
// Obtenemos el contexto de dibujo en 2d.
var ctx = canvas.getContext('2d');
ctx.fillStyle = “rgba(0, 0, 0, 0.5)”;
ctx.fillRect(100, 10, 60, 50);
ctx.lineWidth = 5;
ctx.strokeStyle = “red”;
ctx.strokeRect(200, 70, 60, 50);
```

- Drag and Drop
  - Para hacer que un elemento se pueda arrastrar simplemente estableceremos su atributo draggable a true:

  - Ejemplo:
    - http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_draganddrop

- Web Workers
  - Permiten poder lanzar tareas en segundo plano que se ejecutan en un proceso aislado sin que interfieran con el hilo principal.
  - Así, pueden lanzar secuencias de comandos con tiempos de ejecución largos sin bloquear la interfaz de usuario.
  - Permiten el intercambio de mensajes con el hilo principal (en ambas direcciones).
  - Ejemplo:
    - http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_webworker

- Web Storage
  - En ocasiones es interesante guardar información a nivel de cliente para nuestras aplicaciones web:
    - **Rapidez** de acceso.
    - **Disponibilidad** aún cuando falta la conexión a internet.
  - La manera de almacenar información en cliente soportada por todos los navegadores más comúnmente usada ha sido el mecanismos de **cookies**
  - Han aparecido soluciones particulares de navegadores:
    - userData (IE5+)
    - Local Shared Objects (Flash)
    - Google Gears (Plugin para IE y Firefox)
    - Dojo Toolkit con dojox.storage

- Web Storage
  - Con el fin de estandarizar la manera de almacenar información se definió Web Storage como parte de HTML5.
  - Web Storage permite almacenar pares clave-valor donde la clave será una cadena y el valor otra cadena.
  - Web Storage define dos tipos de almacenamiento con características distintas:
    - Session Storage
    - Local Storage

- Web Storage
  - Algunas características comunes para el **session storage** y el **local storage** son:
    - Permite más de 4ks de almacenamiento.
    - Depende de cada navegador pero suele ser de varias megas.
  - **Session storage:**
    - Elimina el problema de las cookies compartidas entre diferentes tabs – ventanas sobre un mismo dominio ya que cada tab / window mantiene su propio **session storage**.
    - La información guardada en el sesión **storage** se **borra** cuando la **ventana o el tab** se **cierra**.

- Web Storage
  - **local storage:**
    - Permite almacenar información que perdurará aún cuando la ventana o el tab se cierre.
    - Es compartido entre las diferentes ventanas/tabs dentro de un mismo dominio.
    - Los espacios de datos entre dominios no son visibles.
  - **Session/local storage**:
    - Se diferencian en como se comportan los datos en cuanto a durabilidad y ámbitos para el acceso

- IndexedDB
  - Sistema de almacenamiento basado en objetos.
  - API de bajo nivel que ofrece almacenamiento en el cliente de cantidades significativas de datos estructurados, incluyendo archivos y blobs.
  - Para búsquedas de alto rendimiento en esos datos usa índices.
  - Soportado por la gran mayoría de navegadores.

- IndexedDB
  - La forma habitual de usar IndexedDB contempla los siguiente pasos:
    1. Abrir la base de datos y comenzar unatransacción.
    2. Crear o actualizar el object store.
    3. Lanzar una petición de inserción, recuperación de datos...
    4. Esperar a que la petición se complete (evento de finalización).
    5.Realizar algo con el resultado.

- IndexedDB
  - La manera de acceder es usando un prefijo sobre la forma definida en el estándar:

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

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

  - Para abrir una base de datos simplemente ejecutaremos el método open indicando el nombre y la versión de la base de datos a abrir.
  - En caso de que la base de datos exista la abrirá para operar con ella y en caso de que no existir la creará.
  - Las base de datos están asociadas a dominio.

```
var request = indexedDB.open('HolaMundo', 1);
```

- IndexedDB
  - Una vez se solicita la apertura de una base de datos habrá que programar los
  - manejadores para un cambio de versión, para la apertura correcta y para los errores.

```
request.onupgradeneeded = function(e){
        // crear y actualizar object stores e índices.
}
request.onsuccess = function(e){
        // código que se ejecutará si la apertura va bien
        var bd = request.result; // también e.target.result.
}
request.onerror = function(e){
        // código que se ejecutará si hay algún error
}
```

  - **IndexedDB** usa **object stores** en vez de tablas para guardar la información.
  - Una base de datos de **IndexedDB** puede contener tantos **object stores** como sea necesario.
  - Un **object store** es una asociación clave-valor.
  - Utilizaremos el método **createObjectStore** para crear un object store.

```
request.onupgradeneeded = function(e) {
        db = e.target.result;
        var objectStore = db.createObjectStore('usuarios', {
                'keyPath' : 'email'
        });
}
```

- IndexedDB
  - Para crear un object store debemos proporcionar un nombre como primer parámetro y opcionalmente un objeto que puede contener una propiedad keyPath.
  - Si se informa esta propiedad el objeto a almacenar debe poseer la propiedad indicada como keyPath.
  - Podemos crear índices sobre los object stores.

```
objectStore.createIndex('name', { 'unique' : false });
```

- IndexedDB
  - Transacciones:
    - Antes de poder realizar una acción sobre la base de datos se debe comenzar una transacción.
    - A la hora de crear la transacción debemos indicar a qué object stores se propagará y si es de lectura o lectura-escritura.
    - Por defecto las transacciones son de lectura.

```
var txLecturaEscritura = db.transaction(
        [objectStoreName], “readwrite”);
var txLectura = db.transaction(
        [objectStoreName], “readonly”);
```

    - En una transacción pueden ocurrir tres eventos que son complete, abort y error.

```
var tx = db.transaction([objectStoreName], “readwrite”);

tx.oncomplete = function(event){
        alert('Transacción completada');
}

tx.onerror = function(event){
        event.preventDefaut();
}
```

- IndexedDB
  - Insertar:

```
var tx = db.transaction([objectStoreName], “readwrite”);

var objectStore = tx.objectStore(objectStoreName);

var putRequest = objectStore.put(DATA);

putRequest.onsuccess = function(event){
        // OK
};
putRequest.onerror = function(event){
        // ERROR
};
```

- IndexedDB
  - Borrar:

```
var tx = db.transaction([objectStoreName], “readwrite”);

var objectStore = tx.objectStore(objectStoreName);

var deleteRequest = objectStore.delete(KEY);

deleteRequest.onsuccess = function(event){
        // OK
};
deleteRequest.onerror = function(event){
        // ERROR
};
```

- IndexedDB
  - Obtener Información:

```
var tx = db.transaction([objectStoreName], “readonly”);

var objectStore = tx.objectStore(objectStoreName);

var getRequest = objectStore.get(email);

getRequest.onsuccess = function(event){
        var result = event.target.result;
        alert('Get por email ' + e.target.result.name);
};
```

- IndexedDB
  - Cursores:

```
var users = [];

var tx = db.transaction([objectStoreName], “readonly”);

var objectStore = tx.objectStore(objectStoreName);

objectStore.openCursor().onsuccess = function(event){

        var cursor = event.target.result;
        if (cursor){
                users.push(cursor.value);
                cursor.continue();
        } else {
                alert('Ya hemos obtenido todos los usuarios');
        }
};
```

- File API
  - HTML5 introduce nuevas posibilidades para seleccionar ficheros y trabajar con ellos.
  - También ofrece un sandbox en el que se pueden crear, modificar y borrar tanto directorios como ficheros tanto de texto como binarios

- Offline
  - En ocasiones puede resultar interesante que nuestras aplicaciones funcionen cuando no disponemos de conexión.
  - HTML5 define una manera estándar de conseguir esto mediante el uso de un simple fichero de meta-información.
  - Mediante el fichero **cache manifest**:

```
<!DOCTYPE HTML>
<html manifest="cache-manifest.appcache">
<body>
...
</body>
</html>
```