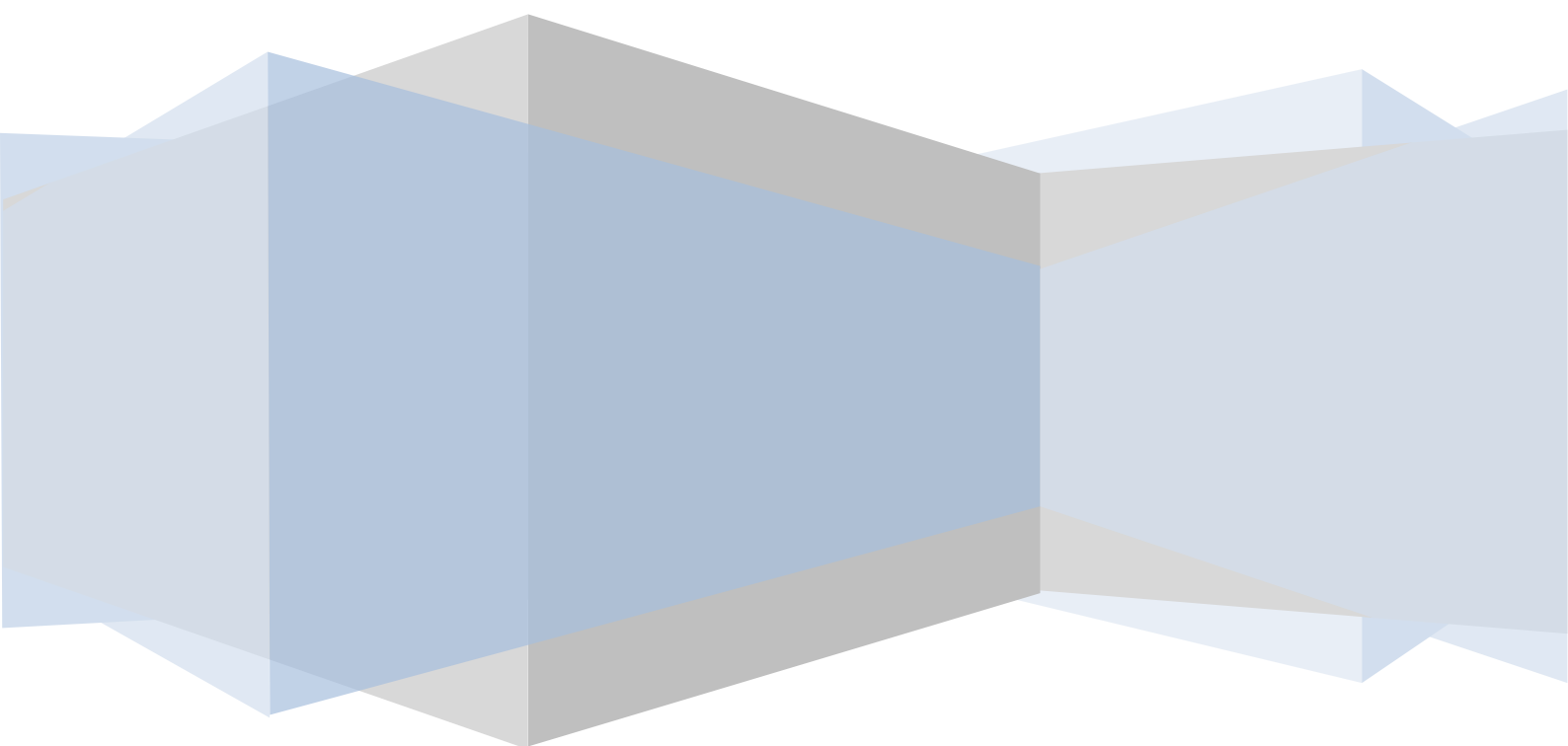


Universidade de Vigo

Javascript para Aplicaciones Web y Móviles

Teoría y Ejemplos

Amador Rodríguez Diéguez



Javascript

Introducción

Javascript es el lenguaje soportado por todos los navegadores HTML modernos, independientemente de la plataforma: teléfonos móviles, tablets, ordenadores portátiles, equipos de escritorio, etc.

Con él, podemos programar el comportamiento de las páginas web, permitiendo acceder y modificar los elementos HTML que componen las páginas web, así como los estilos CSS.

Características generales

Javascript distingue mayúsculas y minúsculas en las palabras reservadas (for, switch, etc.)

Los espacios en blanco y cambios de línea no afectan a la ejecución del código

Aunque es recomendable, las líneas de código no necesitan terminar con punto y coma (;)

Escribir en el documento

```
<html>
<body>
  <h1>Ejemplo de escritura en el documento</h1>
  <script>  /* html5 no necesita el atributo type para js */
    // primer ejemplo de Javascript
    document.write("<h1> HOLA MUNDO </h1>");
    /* este es un comentario
    multilínea */
  </script>
</body>
</html>
```

Caracteres especiales

\'	Comilla simple	\r	Retorno de carro
\"	Comillas dobles	\t	Tabulador
\\	Barra inclinada	\b	Borrar hacia atrás
\n	Nueva línea	\f	Salto de página de impresión

```
var txt="Este texto es un mensaje.";
document.write(txt);
var txt="Este \"texto\" es un mensaje.";
document.write(txt);
```

Este texto es un mensaje.
Este "texto" es un mensaje.

Variables

En los nombres de variables se distinguen mayúsculas y minúsculas y sus nombres tienen que empezar por una letra, un guión de subrayado o el símbolo \$. Si se redeclara una variable, no pierde su valor.

Si se definen con `let`, son locales al par de llaves en las que se declaran y ocultan las globales. Si se definen con `const`, son constantes y no podrá cambiar su valor.

Operadores

Operadores aritméticos: `+` `-` `*` `/` `%` `++` `--` `%=` `+=` `-=` `*=` `/=` `**` (ES6)

Operadores de bit: `&` `|` `~` `^` (*o-exclusiva de bits*) `<<` `>>`

Comparaciones: `!=` `!==` `==` `===` `>=` `<=` `&&` `||` `!` (*no existe la O-Exclusiva*)

Tipos: `typeof` devuelve un texto con el tipo de la variable ("number", "string", "object", etc.)

Unario: `+` también se usa para convertir una string en número (devuelve el valor NaN si es imposible)

Ternario: devuelve un valor entre dos dados `k=(z<2)?10:20;` `// (condición)?valor1:valor2`

```
y=5;
z=2;
x=y+z;
txt1="Este es";
txt2="un \
    ejemplo";           // uso de \ para definir Strings multilínea
txt3=txt1+" "+txt2;     // + concatena strings (núm + string = string)
x=5+"5";
document.write(x);      // escribe "55"
document.write(typeof x); // escribe "string"
n = + x;                // n será un número de valor 55
n == "55";              // cierto aunque sean de tipos distintos
n === "55";             // falso porque aunque el valor es el mismo, no coinciden los tipos
n !== "55";             // cierto porque son distintos tipos aunque el valor sea el mismo
n != "55"               // falso porque el valor es el mismo aunque sean distintos tipos
x = 3 ** 2              // exponenciación: 3 elevado a 2
```

Nota: Se cumple que `true == 1` pero no que `true === 1`

If

```
<script>
  var n = 7;
  if (n < 5) {
    document.write("<b>Suspenso</b>"); // 1 instrucción: se pueden omitir llaves
  } else {
    document.write("<b>Aprobado</b>"); // else opcional
  }
</script>
```

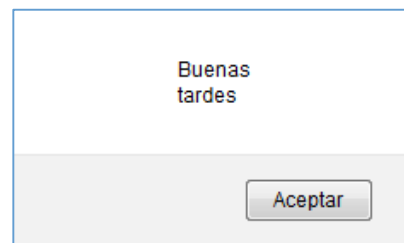
Switch

```
var dia = 7;
switch (dia){ // además de expresiones enteras, también vales strings
  case 1:
    document.write("<b>Lunes</b>"); break; // cada bloque puede
  case 2: // tener un número
    document.write("<b>Martes</b>"); break; // arbitrario de
  case 3: // de instrucciones
    document.write("<b>Miércoles</b>"); break;
  case 4:
    document.write("<b>Jueves</b>"); break;
  case 5:
    document.write("<b>Viernes</b>"); break;
  case 6:
    document.write("<b>Sábado</b>"); break;
  case 7:
    document.write("<b>Domingo</b>"); break;
  default:
    document.write("<b>valor erróneo</b>");
}
```

```
switch (mes) {
  case 1: case 3:
  case 5: case 7:
  case 8: case 10: case 12:
    case 5: d = 31; break;
  case 4: case 6: case 9:
  case 11: d = 30; break;
  case 2: d = 28; break;
  default: d = "mes erróneo";
}
```

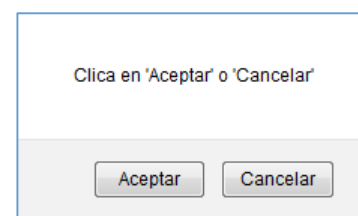
Mensaje de alerta

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <p>demostración de alert</p>
  <script>
    alert("Buenas\ntardes");
  </script>
</body>
</html>
```



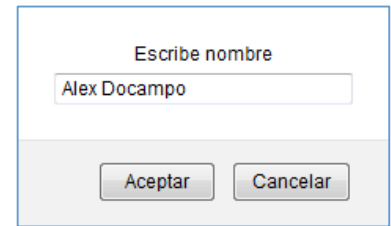
Cuadro de confirmación

```
var r = confirm("Clica en 'Aceptar' o 'Cancelar'");
if (r == true){
  alert("Has pulsado OK");
}
else{
  alert("Has pulsado Cancelar");
}
// confirm devuelve true o false
```



Prompt

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script>
    var nombre=prompt("Escribe nombre", "Alex Docampo");
    if (nombre!=null && nombre!="")
      document.write("Hola, " + nombre);
      // cancelar: devuelve null
  </script>
</body>
</html>
```



For

```
<script>
  for (var i=0;i<=5;i++) {    // la condición puede usar & y |
    document.write("El número es " + i);
    document.write("<br>");
  }
  document.write("El número es " + i); // la variable existe aquí y vale 6
  /* el 1er campo y el 3º del for pueden tener varias instr separadas por comas */
</script>
```

While

```
var i=0;
while (i<=5) {
  document.write("El número es " + i);
  document.write("<br>");
  i++;
}
```

```
var i=0;
do {
  document.write("El número es " + i);
  document.write("<br>");
  i++;
} while (i<=5);
```

No se puede declarar la variable en la condición del bucle

Break

```
<script>
  for (var i=0;i<=10;i++) {
    if (i==3) break;
    document.write("El número es " + i);
    document.write("<br>");
  }
</script>
```

Esta instrucción debe evitarse en la medida de lo posible, ya que complica el código de modo innecesario. Una alternativa al código anterior podría ser el siguiente:

```
for (var i=0;i<=2;i++) { // la condición del if se introdujo en la del for
    document.write("El número es " + i);
    document.write("<br>");
}
```

Continue

```
for (var i=0;i<=10;i++) {
    if (i==3) continue;
    document.write("El número es " + i);
    document.write("<br>");
}
```

Esta instrucción debe evitarse en la medida de lo posible, ya que complica el código de modo innecesario. Una alternativa al código anterior podría ser el siguiente:

```
for (var i=0;i<=10;i++) {
    if (i!=3) {
        document.write("El número es " + i);
        document.write("<br>");
    }
}
```

Funciones

Funciones globales

Existe un reducido conjunto de funciones proporcionadas por Javascript :

- `parseFloat`, `parseInt`: convierten una string a valor real o entero (NaN si no es posible)
`var x = parseFloat("10");`
- `eval`: evalúa y ejecuta una string como si fuese un script de código (uso poco recomendable)
- `isFinite`: indica si el argumento representa un valor numérico legal distinto de $\pm\infty$
- `isNaN`: indica si el argumento representa un valor numérico legal. No se puede comparar directamente con NaN: `a==NaN` ó `a===NaN` no funcionan, sino que habría que hacer `isNaN(a)`

Elementos básicos

Las funciones pueden tener parámetros (separados por comas, si hay más de uno) y devolver valores. No se especifican tipos, ni se comprueba cuántos hay.

```
function cuadrado(a) {
    return a * a;
}
```

Otra forma de definir funciones: como una funciones anónimas:

```
var cuadrado = function (a) {return a * a};
```

Para invocar una función, simplemente se indica su nombre con los argumentos deseados. Se puede recoger el valor en una variable:

```
var resultado = cuadrado(3) * 2;
```

Las funciones se no se suelen definir en el cuerpo (*body*) de la página:

```
<!DOCTYPE html>
<html>
<head>
  <script>
    function cuadrado(a) { return a * a; }
  </script>
</head>
<body>
  <script>
    var resultado = cuadrado(3);
    alert (resultado)
  </script>
</body>
</html>
```

Las funciones pueden invocarse antes de ser declaradas.

Autoinvocación

Las funciones pueden llamarse de forma automática (autoinvocación) en la definición. Para ello hay que definir la función entre paréntesis y añadir paréntesis vacíos a continuación.

```
<!DOCTYPE html>
<html>
  <head>
    <script>    ( function () {
                  alert ("Hola");
                }
              ) ();
    </script>
  </head>
  <body>
    Ejemplo de función que se autoinvoca;
  </body>
</html>
```

Argumentos

Si se invoca una función con menos argumentos que en la definición, en el interior de la función, los parámetros no asignados toman el valor *undefined* (siempre sin comillas):

```
function f(x, y) {
  if (y === undefined) y = 0;
}
```

Existe la posibilidad de proporcionar valores por defecto a los argumentos (comenzando por el final):

```
function f(x, y=1, z=2) { ... }
```

Cuando se invoca la función sin alguno de los parámetros, en el cuerpo de la función, los argumentos tomarán el valor por defecto indicado en la cabecera de la función.

Todos los argumentos pasados a una función se reciben también en un array llamado *arguments*.

```
function maximo() {
    var i, max;
    for (i = 0; i < arguments.length; i++) {
        if (arguments[i] > max || i == 0) {
            max = arguments[i];
        }
    }
    return max;
}
x = maximo(1, 123, 500, 115, 44, 88);
```

En el array *arguments*, aparecen tanto los argumentos declarados como los no declarados.

Variables locales y globales

Una variable declarada (palabra reservada *var*) dentro de una función es local (solo accesible dentro de la función). Se pueden tener variables locales con el mismo nombre en funciones distintas. Las variables locales se eliminan al salir de la función.

```
function f1() {
    var a = 4;
    return a * a;
}
```

Las variables declaradas fuera de las funciones son globales (accesibles desde cualquier punto de la página y de la ventana). Éste tipo de variables se eliminan cuando se cierra la página. Se debe reducir su uso al mínimo. Las variables locales *ocultan* a las variables globales del mismo nombre.

```
var a = 4;
function f2() {
    return a * a;
}
```

Si se usa una variable sin declararla previamente, se crea como variable global, incluso dentro de una función.

```
var x=5; // var es opcional. La inicialización también es opcional
var coche="Volvo"; // el texto también se puede delimitar por comillas simples
```

Las variables declaradas dentro de un bloque ({...}) no son locales a ese bloque. No existen variables estáticas. Se recomienda declarar todas las variables al inicio de las funciones.

Las variables y funciones globales sobrescriben a las del objeto *window* y viceversa.

Funciones anidadas

Las funciones definidas dentro otras sólo son accesibles desde ellas y tienen acceso a las variables locales de sus funciones padres como si fuesen globales:

```
function incrementa() {  
    var contador = 0;  
    function mas1() {contador += 1;}  
    mas1();  
    return contador;  
}
```

Cierres (*closures*)

Un cierre es una función que tiene acceso a las variables de su función padre incluso tras haberse cerrado ésta. Esto permite la definición de variables privadas.

```
<!DOCTYPE html>  
<html>  
  <head>  
    <script>  
      var incrementa = (function() {  
        var contador = 0;  
        return function() {  
          return contador += 1;  
        };  
      })();  
    </script>  
  </head>  
  <body>  
    <script>  
      document.write(incrementa()+"<br>");  
      document.write(incrementa()+"<br>");  
      document.write(incrementa()+"<br>");  
    </script>  
  </body>  
</html>
```

En el ejemplo anterior, la función autoinvocada sólo se ejecuta una vez (por lo que la variable contador se pone a cero sólo una vez) y devuelve la función anónima interna. Cada vez que se invoque a la función contenida en la variable incrementa, se ejecutará la función anónima. A pesar de que la función autoinvocada terminó, la variable contador sigue existiendo.

El resultado de la página son los valores 1, 2, y 3 cada uno en una línea.

Permite la captura y procesamiento de excepciones sin que se termine la ejecución del script.

```
var txt="";
function mensaje() {
  try {
    addlert("Bienvenido"); // no existe esta función
  } catch(err) {
    txt="Ha habido un error en esta página.\n\n";
    txt+="Pulsa OK para continuar en esta página,\n";
    txt+="o Cancel para ir a la página del curso.\n\n";
    if(!confirm(txt))
      document.location.href="http://momentum.uvigo.es/cursos/javascript.html";
  } finally {
    alert ("saliendo del try-catch-finally");
  }
}
```

La cláusula *finally* permite ejecutar un bloque de código tanto si se produjo error como si no.

Se pueden lanzar errores personalizados con *throw*:

```
<html>
<head>
  <script>
    function funcion(x) {
      var ok = 1;
      try {
        if(x == "") throw "Sin valor";
        if(isNaN(x)) throw "No es numérico";
        if(x > 10) throw "demasiado alto";
        if(x < 5) throw "demasiado bajo";
      }
      catch(err) {
        ok = 0;
        alert("Mensaje: " + err);
        return;
      }
      finally { // se ejecuta a pesar del return en el catch
        if (ok === 1) document.write("Sin problemas");
        else document.write("Hubo algún problema");
      }
    }
  </script>
</head>
<body>
  <script>
    var valor = prompt("Introduce un valor");
    funcion(valor);
  </script>
</body>
</html>
```

Temporizadores

Se puede programar la ejecución de una función una única vez tras un período de espera. También es posible cancelar dicha temporización. Se le pasa el nombre de la función, no una llamada.

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function temporizador() {
      var t=setTimeout("mensaje()",3000);      // clearTimeout(t) cancela
    }
    function mensaje() {alert("Hola");} // alternativa:
  </script>                                     // var t=setTimeout("alert('Hola')",3000);
</head>
<body>
  <script>
    respuesta = confirm("¿Temporizar?");
    if (respuesta == true) temporizador();
    else document.write("Sin temporización");
  </script>
</body>
</html>
```

Pueden coexistir múltiples temporizadores.

Si se quiere utilizar temporizadores periódicos (se disparan cada *n* milisegundos) se utiliza *setInterval* y *clearInterval*:

```
<html>
<head>
  <script>
    var tempo = setInterval(function() {miTemporizador()}, 1000);
    function miTemporizador() {
      var d = new Date();
      var t = d.toLocaleTimeString();
      document.getElementById("hora").innerHTML = t;
    }
    function pararReloj () {
      clearInterval(tempo);
    }
  </script>
</head>
<body>
  <p id="hora"></p>
  <button onclick="pararReloj()">Parar</button>
</body>
</html>
```

La llamada al *setInterval* también podría haberse hecho de alguna de las dos formas siguientes:

```
var tempo = setInterval("miTemporizador()", 1000);
var tempo = setInterval(function() {
    var d = new Date();
    var t = d.toLocaleTimeString();
    document.getElementById("hora").innerHTML = t;
}, 1000); // sobra la definición de miTemporizador()
```

Scripts externos

Los ficheros de scripts externos no pueden contener las etiquetas `<script></script>`.

```
<html>
  <head>
    <script type="text/javascript" src="funciones.js"></script>
    // en HTML5 se puede omitir el atributo type:
    <script src="http://uvigo.es/js/funciones2.js"></script> // fich remoto
  </head>
  <body></body>
</html>
```

Eventos

Lista de eventos

blur, change, click, dblclick, error, focus, keydown, keypress, keyup, load, mousedown, mousemove, mouseout, mouseover, mouseup, resize, select, unload, etc.

Principales eventos

Load y Unload

Se disparan cuando el usuario entra y sale de la página, respectivamente. El evento `onLoad` se usa frecuentemente para comprobar el tipo y version del navegador y cargar la versión apropiada de la página.

Focus, Blur y Change

Se disparan cuando un elemento gana el foco, pierde el foco y cambia su valor (input, keygen, textarea y select). Se utilizan habitualmente en combinación para la validación de los formularios.

Submit

Detecta el envío del formulario. Se usa para validar los campos de un formulario antes de enviarlo.

MouseOver, MouseOut

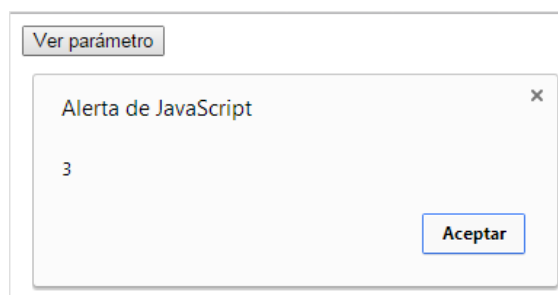
Se produce cuando el cursor pasa por encima de un elemento HTML o cuando lo abandona.

Modo clásico de respuesta a eventos

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function saludo(){
      alert ("HOLA");
    }
  </script>
</head>
<body>
  <button type="button" onclick="saludo()">Para saludar</button>
</body> // alternativa: onclick="alert('HOLA') "
</html>
```

Se le puede pasar argumentos a la función, tanto literales como variables:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script>
      function verParam(p) { alert(p); }
    </script>
  </head>
  <body>
    <script> var a = 3; </script>
    <button type="button" onclick="verParam(a)">Ver parámetro</button>
  </body> // alternativa: onclick="alert(a) "
</html>
```



Otra alternativa sería usar la notación `window.onload=hacerclíc` (en este caso añade "on" al nombre del evento)

Objetos de Javascript

Uso básico de los objetos

Propiedades

```
var txt="Hola mundo";
document.write(txt.length); // length: propiedad del objeto String
```

Métodos

```
var str="Hola mundo";
document.write(str.toUpperCase()); // toUpperCase: propiedad del objeto String
```

Objetos propios de Javascript

Array, Date, String, Math, Boolean, Number y RegExp.

Number

Javascript sólo tiene un tipo de número. La clase Number nos proporciona constantes de gran utilidad: MAX_VALUE, MIN_VALUE, POSITIVE_INFINITY, NEGATIVE_INFINITY y NaN (Not a Number).

Están disponibles los siguientes métodos:

- toExponential(x): convierte el número a notación exponencial.
- toFixed(x): formatea el número con x decimales.
- toPrecision(x): formatea el número con x dígitos.
- toString(): convierte el número en texto.
- isFinite(), isInteger(), isNaN(): comprueban si es distinto a \pm infinito, entero o si es un número
- constantes EPSILON (mínimo real positivo), MIN_SAFE_INTEGER, MAX_SAFE_INTEGER (ES6)

```
var num = "5";
alert(Number(num)+1);           // constructor con argumento string

var y = 123e-5;                 // 0.00123
var num = 5.56789;
var n = num.toExponential()     // resultado: 5.56789e+0

var n2 = num.toFixed(2);        // resultado: 5.57

var num3 = 123.9714;
var n3 = num3.toPrecision(3);   // resultado: 124
var n4 = num3.toPrecision(2);   // resultado: 1.2e+2
```

Debido al formato interno de almacenamiento, se pueden producir errores de precisión:

```
var x = 0.2 + 0.1;              // resultado: 0.30000000000000004

var x = 0xFF;                   // hexadecimal: 255
var n = 128;
n.toString(16);                 // muestra en hexadecimal: 80
n.toString(8);                  // muestra en octal: 200
n.toString(2);                  // muestra en binario: 10000000

var n = 2;
while (n != Infinity) n = n * n; // también existe -Infinity como ctes globales

var x = 2 / 0;                  // x = Infinity
var y = -2 / 0;                 // y = -Infinity
typeof Infinity;               // devuelve "number"
var m = (-Number.MAX_VALUE) * 2; // devuelve el valor -Infinity

var k = 100 / "Nada";
isNaN(k);                       // isNaN es una función global. Devolverá true

var x = NaN;                    // constante global
var y = 5;
var z = x + y;                  // z = NaN
s = x + "5";                    // s = "Nan5"
```

Arrays

```
var coches=new Array();           // imprescindible la declaración.
coches[0]="Saab";                 // alternativa:
coches[1]="Volvo";                // var coches=new Array("Saab","Volvo","BMW");
coches[2]="BMW";
coches[0]="Opel";
document.write(coches[0]);
var coches=["Saab","Volvo","BMW"];
```

El operador `in` devuelve un dato lógico que indica si la propiedad o posición indicados existen (no busca valores):

```
"Saab" in coches           // false (no funciona con valores)
0 in coches                // true
1 in coches                // true
4 in coches                // false (no existe la posición 4)
"length" in coches         // true (length es una propiedad del array)
```

```
<html>
<head> <meta charset="UTF-8"> </head>
<body>
  <script>
    var frutas = ["Uva", "Naranja", "Manzana", "Mango"];
    // splice añade o elimina de un vector:
    // argumento 1: índice (negativo: posición desde el final)
    // argumento 2: número de elementos a eliminar (0 para insertar)
    // resto de los argumentos: valores a añadir.
    // devuelve los elementos eliminados
    frutas.splice(2,0,"Pera"); //añade
    document.write(frutas);
  </script>
</body>
</html>
```

La clase `array` dispone de múltiples funciones:

```
var frutas = [];
var n = frutas.push("Mandarina", "Pera"); // añade al final y devuelve tamaño
```

Otras funciones interesantes son: `concat`, `join` (array a string), `shift` (elimina y devuelve primer elemento), `pop`, `indexOf`, `lastIndexOf`, `sort`, `slice`, `unshift` (añade al final), `find`, `valueOf`, `delete`, etc.

```
var frutas = ["Pera", "Naranja", "Manzana", "Mango", "Pera", "Naranja", "Manzana"];
var a = frutas.indexOf("Manzana", 4); // a recibirá el valor 6
```

```
var puntos = [40, 100, 1, 5, 25, 10];
puntos.sort(function(a, b){return b-a}); // ordenación descendente
```

Si se asigna valor a una posición inexistente, dicha posición se crea, pero ninguna otra (puede dar lugar a arrays con posiciones no definidas):

```
frutas[20] = "Granada"; // el elemento frutas[19] no existiría (no se crea)
```

Las posiciones inexistentes se pueden detectar con el operador `in` o comparándolas con `undefined`:
`if (frutas[17]===undefined) ...`

Posiciones inexistentes en la declaración: `valores = [3,,9,4];`

La propiedad *length*, contiene el tamaño del array. Las posiciones inexistentes también cuentan. En el siguiente ejemplo, *length* devuelve el valor 4:

```
var vector = [];  
vector[3] = 5; // las posiciones de 0, 1 y 2 no existen  
document.write (vector.length);
```

Los arrays de múltiples dimensiones se construyen por anidamiento:

```
var m = [[1, 2], [3, 4], [5, 6]]; // matriz 3x2  
document.write(m[1][0]); // devuelve undefined si la pos. no existe
```

Los arrays pueden contener funciones:

```
function duplica(a) {return 2*a;}  
function triplica(a) {return 3*a;}  
var array_de_funciones = [duplica, triplica]; // crea array  
document.write(array_de_funciones[0](3) + "<br>"); // invoca usando array en  
document.write(array_de_funciones[1](3)); // lugar del nombre
```

Objeto Date

Constructores

```
new Date() // fecha y hora actual  
new Date(milisegundos) // milisegundos desde 01/01/1970  
new Date(FechaComoTexto) // en inglés: "October 31 2015, 17:00:00"  
new Date(año, mes, día, horas, minutos, segundos, milisegundos)
```

Ejemplo de uso

```
var f=new Date();  
var dia=["Domingo", "Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado"];  
document.write("Hoy es " + dia[f.getDay()]);
```

Existen muchas otras funciones útiles de este objeto: *getFullYear*, *getTime* (*timestamp* en milisegundos) , *getHours*, *getMinutes*, *toLocaleDateString*, *Date.now()*, etc.

Objeto String

Este objeto proporciona multitud de funciones para trabajar con texto:

charAt(), *concat()*, *find()*, *forEach()*, *indexOf()*, *lastIndexOf()*, *localeCompare()*, *match()*, *replace()*, *search()*, *slice()*, *split()*, *substr()*, *substring()*, *toLocaleLowerCase()*, *toLocaleUpperCase()*, *toLowerCase()*, *toString()*, *toUpperCase()*, *trim()*, etc. Además se puede indexar usando corchetes

```
var str = "Ejemplo de búsqueda en una cadena de texto";  
document.write(str.indexOf("de") + "<br>"); // muestra 8  
document.write(str.indexOf("de, 15") + "<br>"); // muestra 29  
document.write(str.indexOf("abc") + "<br>"); // muestra -1  
document.write(str[2]); // muestra e
```

El objeto Math

Contiene multitud de funciones y constantes matemáticas. No existe función de redondeo.

```
var x = Math.PI;  
var y = Math.sqrt(16);  
document.write(Math.round(4.7)); // redondea al entero más cercano  
document.write(Math.random()); // número aleatorio en el intervalo [0, 1]  
document.write(Math.floor(Math.random()*11)); // trunca
```


Creación directa de objetos

Hay varias formas de crear objetos. Usando el operador *new* sería así:

```
persona=new Object();
persona.nombre="Laura";
persona.apellido="Regueiro";
persona.edad=24;
persona.colorOjos="azul";
```

Como "objeto literal" se definiría así:

```
persona={nombre:"Laura", apellido: "Regueiro", edad:24, colorOjos:"azul"};
```

En esta modalidad (creación directa de objetos) se puede añadir y eliminar tanto propiedades como métodos a un objeto concreto del siguiente modo:

```
persona.profesion = "ingeniero";           // no funcionaría con corchetes
persona.cambiaApellido = cambiaApellido;    // en algunos navegadores se pueden
function cambiaApellido(nuevo_apellido) {   // unificar las dos sentencias
    this.apellido=nuevo_apellido;
}
delete persona.profesion;                   // también valdría: delete persona["profesion"]
delete persona.cambiaApellido;
```

En esta modalidad de creación de objetos, no existe constructor.

El operador *delete* solo elimina propiedades de objetos, y no produce ningún efecto en variables o funciones. Las variables globales son realmente propiedades del *objeto global*.

Creación de objetos desde el constructor (definición de clase)

De este modo se pueden crear múltiples objetos con la misma estructura:

```
function persona(nombre,apellido,edad,colorOjos) { // constructor
    this.nombre=nombre;
    this.apellido=apellido;
    this.edad=edad;
    this.colorOjos=colorOjos;

    // definir métodos (en algunos navegadores se pueden unificar las 2 líneas):
    this.cambiaApellido=cambiaApellido;
    function cambiaApellido(nuevo_apellido) { this.apellido=nuevo_apellido; }
}
```

La palabra reservada *this* representa al objeto propietario de la función durante la ejecución. No debe confundirse con la palabra reservada *self*, que hace referencia a la ventana en la que se está ejecutando el código:

```
self.setInterval(3000) ↔ window.setInterval(3000) ↔ setInterval(3000)
```

Para crear objetos a partir del constructor:

```
var madre = new persona("Laura", "Regueiro", 24, "azul");
var padre = new persona("Miguel", "Paz", 48, "verde");
```

En esta modalidad no es posible eliminar atributos ni métodos sin modificar el constructor.

Uso de propiedades y métodos

Para acceder a las propiedades hay dos posibilidades: *objeto.propiedad* u *objeto[propiedad]*

```
var nom = persona.nombre;      // equivalente a: var nom = persona["nombre"]
persona["edad"] = 25;          // equivalente a: persona.edad = 25;
var campo = "edad";            // las dos últimas líneas equivalen a la anterior
persona[campo] = 25;
```

Para invocar un método: `madre.cambiaApellido("Río");`

La asignación de referencias no genera nuevos objetos: *persona1 = persona2* (dos referencias al mismo objeto).

Operador *in*

Devuelve *true* si la propiedad especificada existe en el objeto y *false* en caso contrario (también funciona con objetos predefinidos de Javascript).

```
var persona = {nombre:"Carlos", apellidos:"López Rivas", edad:25};
"nombre" in persona      // true
"edad" in persona        // true
"PI" in Math              // true
"NaN" in Number          // true
"length" in String       // true
```

Operador *instanceof*

Devuelve *true* si el objeto especificado es una instancia de una clase dada y *false* en caso contrario. También funciona con objetos definidos por el programador, como por ejemplo, la clase *persona* usada en los ejemplos anteriores.

```
var coches = ["Saab", "Volvo", "BMW"];
coches instanceof Array;      // true
coches instanceof Object;     // true
coches instanceof String;     // false
coches instanceof Number;     // false
```

Objetos de métodos (ES5)

ECMAScript 5 añadió una gran cantidad de métodos que facilita el trabajo con objetos: `defineProperty`, `defineProperties`, `getOwnPropertyDescriptor`, `getOwnPropertyNames`, `keys`, `getPrototypeOf`, `preventExtensions`, `isExtensible`, `seal`, `isSealed`, `freeze` y `isFrozen`.

Definición de clases con Class (ES6)

ECMAScript 6 proporciona una nueva forma de definir clases usando la palabra reservada *class*, la cual crea una clase y, obligatoriamente, define el constructor en el que se definirán los atributos.

```
class Coche {  
  constructor(marca) {  
    this.fabricante = marca;  
  }  
}
```

Para crear objetos, se usa el operador *new*:

```
coche_nuevo = new Coche ("Audi");
```

El constructor se invoca automáticamente al ejecutarse el *new*, inicializándose así el objeto de forma automática. El nombre del constructor no puede cambiar y, en caso de no existir, Javascript añadirá un constructor *vacío* e *invisible*.

Métodos

Se pueden añadir métodos que podrán usar los atributos definidos en el constructor.

```
class Coche {  
  constructor(marca) {  
    this.fabricante = marca;  
  }  
  mensaje(saludo) {  
    return saludo + ". Mi coche es un " + this.fabricante;  
  }  
}  
coche_nuevo = new Coche ("Audi");  
alert (coche_nuevo.mensaje("Buenos días"));
```

Métodos estáticos

Los métodos estáticos se invocan sobre la clase y no es necesario que exista ningún objeto de la clase. Por ello, no pueden usar los atributos creados en el constructor.

```
class Coche {  
  constructor(marca) {  
    this.fabricante = marca;  
  }  
  static saludo(d) {  
    return "Hola, " + d;  
  }  
  mensaje(s) {  
    return s + ". Mi coche es un " + this.fabricante;  
  }  
}  
coche_nuevo = new Coche ("Audi");  
alert (coche_nuevo.mensaje("Buenos días"));  
alert (Coche.saludo("Pepa"));
```

Getters y setters

Se pueden crear propiedades que permitan, por ejemplo, el acceso controlado al objeto:

```
class Coche {
  constructor(marca) {
    this.fabricante = marca;
    this.pvp = 0;
  }
  mensaje(saludo) {
    return saludo + ". Mi coche es un " +
      this.fabricante;
  }
  get precio() { return this.pvp; }
  set precio(x) { if (x >= 0) this.pvp= x; }
}

coche_nuevo = new Coche ("Audi");
coche_nuevo.precio = -3;           // no asigna el valor por ser negativo
alert(coche_nuevo.precio);
coche_nuevo.precio = 45000;
alert(coche_nuevo.precio);
```

For in

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <script>
    var persona={nombre:"Iria", apellido:"Paz", edad:25}; // declaración
    for (x in persona){
      document.write(x + ": " + persona[x] + "<br>");
    }
  </script>
</body>
</html>
```

Este bucle también se puede usar con arrays, aunque hay que tener en cuenta que no se recorren las posiciones inexistentes. En el ejemplo siguiente sólo se mostraría la posición 5, ya que las anteriores no existen.

```
var a = [];
a[5] = 5;           // a.length → 6 porque incluye las posiciones indefinidas
for (var x in a) document.write (x + "<br>");
```

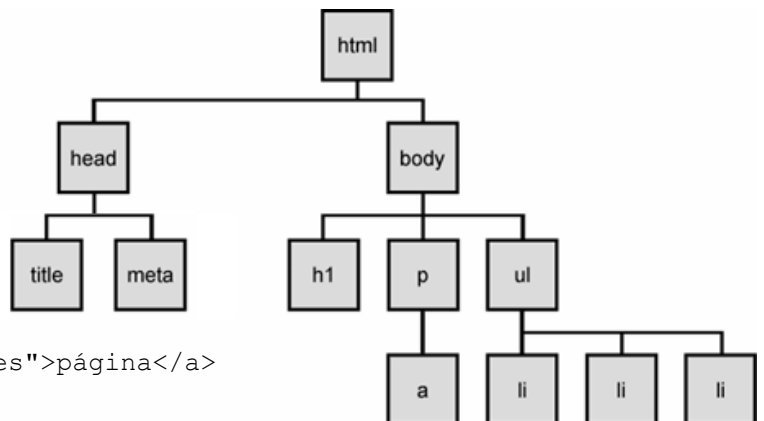
Si recorriéramos con un for normal, veríamos que para las posiciones inexistentes muestra el valor *undefined*.

```
v=[1,2, ,3];

for (i=0;i<v.length;i++){
  if (v[i]!==undefined) document.write(i + ": " + v[i] + "<br>");
}
```

Introducción

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Página sencilla</title>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Encabezado de la página</h1>
  <p>
    Esta
    <a href="http://momentum.uvigo.es">página</a>
    es muy interesante
  </p>
  <ul>
    <li>HTML5</li>
    <li>Java</li>
    <li>PHP</li>
  </ul>
</body>
</html>
```



Listado de elementos

document	form	input password	option
event	frame/iframe	input radio	select
htmlelement	frameset	input reset	style
anchor	image	input submit	table
area	input button	input text	tablecell
base	input checkbox	link	tablerow
body	input file	meta	textarea
button	input hidden	object	

Tipos de nodos

En el modelo de objetos del documento (DOM), todo se representa por medio de nodos:

- El documento: nodo de tipo documento, que es la raíz de todo el esquema.
- Los elementos HTML: nodos de tipo elemento
- Los atributos de los elementos: nodos de tipo atributo
- El texto interno de los elementos: nodos de tipo texto
- Los comentarios: nodos de tipo comentario

Algunas de las propiedades de los nodos son: *nodeName*, *nodeValue*, *nodeType*, *ChildNodes*, *attributes*, *firstChild*, *lastChild*, *previousSibling* y *nextSibling*.

Algunos métodos de los nodos son: *appendChild*, *removeChild*, *replaceChild*, *createElement*, *createTextNode*, *createAttribute*, *getAttribute*, *setAttribute* y *removeAttribute*.

Acceso a los nodos

getElementById()

La función `getElementById()` devuelve el elemento cuyo atributo `id` coincide con el parámetro indicado en la función (distingue mayúsculas y minúsculas). Como el atributo `id` debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo indicado. Cuando está repetido suele devolver el primero. Esta función y `getElementsByName`, `getElementByClassName` y `getElementByTagName` localizan los elementos independientemente del nivel de anidamiento en el que se encuentren.

```
<html>
  <body>
    <h1 id="encabezado">Ejemplo getElementById</h1>
    <script> // innerHTML: lee o escribe el texto contenido en el elemento
      document.write(document.getElementById("encabezado").innerHTML)
    </script>
  </body>
</html>
```

Si el identificador está repetido, el comportamiento de esta función es impredecible (dependerá del navegador)

Acceso secuencial

El código del documento es leído e interpretado por el navegador de modo secuencial, por lo que no es posible acceder a la referencia a un elemento antes de su definición. Por ello, de los dos siguientes fragmentos de código, el de la izquierda funciona, pero el de la derecha no:

<pre><!DOCTYPE html> <html> <body> <h1 id="e">Título</h1> <script> var x=document.getElementById("e"); alert(x.innerHTML); </script> </body> </html></pre>	<pre><!DOCTYPE html> <html> <body> <script> var x=document.getElementById("e"); alert(x.innerHTML); </script> <h1 id="e">Título</h1> </body> </html></pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Para evitar este problema, el código que use referencias al documento, debe ser ejecutado cuando la página esté totalmente cargada. Dicho código se escribirá en funciones (en el *head*, típicamente) y se invocarán automáticamente (a no ser que se quieran invocar por otros eventos) por medio del evento *load* de la página. Esta técnica funciona porque las funciones se cargan pero no se interpreta su contenido.

```
window.onload=funcion123();
```

o, de modo equivalente:

```
window.addEventListener('load', funcion123);
```

getElementsByName()

Obtiene todos los elementos de la página cuyo atributo *name* coincida con el parámetro que se le pasa a la función (distingue mayúsculas y minúsculas). Devuelve una colección que no se puede recorrer con el `for in`. Dicha colección contiene multitud de métodos y propiedades, como por ejemplo *length* e *item* (recibe el índice y devuelve el elemento en esa posición). Para acceder a cada elemento se puede, aparte de usar el método *item*, indexar la propia colección. Por ejemplo:

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <p name="parrafo">Párrafo 0</p>
  <p name="parrafo">Párrafo 1</p>
  <p name="parrafo">Párrafo 2</p>
  <script>
    var parrafos = document.getElementsByName("parrafo");
    for (var i = 0; i < parrafos.length; i++) {
      parrafos[i].innerHTML += "-"; // variante 1
      parrafos.item(i).innerHTML += (2 * i); // variante 2
    }
  </script>
</body>
</html>
```

Párrafo 0-0

Párrafo 1-2

Párrafo 2-4

getElementsByTagName()

Obtiene todos los elementos de la página cuya etiqueta sea igual al parámetro que se le pasa a la función (no distingue mayúsculas y minúsculas). Devuelve una colección que se puede recorrer igual que en la función *getElementsByName*:

```
var parrafos = document.getElementsByTagName("p");
for (var i=0; i < parrafos.length; i++) {
  parrafos[i].innerHTML += "-"; // variante 1
  parrafos.item(i).innerHTML += (2 * i); // variante 2
}
```

getElementsByClassName()

Obtiene una colección con todos los elementos de la página cuya clase sea igual al parámetro que se le pasa a la función (distingue mayúsculas y minúsculas). El recorrido de la colección se hace como en los dos casos anteriores (*getElementsByTagName* y *getElementsByName*):

```
<body>
  <p class="parrafo">Párrafo 0</p>
  <p class="parrafo">Párrafo 1</p>
  <p class="parrafo">Párrafo 2</p>
  <script>
    var parrafos = document.getElementsByClassName("parrafo");
    for (var i = 0; i < parrafos.length; i++) {
      parrafos[i].innerHTML += "-"; // variante 1
      parrafos.item(i).innerHTML += (2 * i); // variante 2
    }
  </script>
</body>
```

Modificar el estilo CSS de un nodo

El campo *style* del nodo permite leer y modificar su estilo. Esta acción tiene prioridad sobre cualquier estilo previo asignado de cualquier modo (en línea, hoja de estilos, etc.):

```
document.getElementById("p2").style.color = "blue";
```

En el siguiente ejemplo, *getComputedStyle* devuelve el estilo actual del elemento pasado como argumento (sin embargo, no sería posible modificar el estilo obtenido por medio de dicha función):

```
maximo = parseInt(window.getComputedStyle(barra).getPropertyValue('width'));
```

Agregar nodos

Los métodos de inserción, borrado y reemplazo, se aplican sobre el elemento padre de los nodos a insertar, borrar o reemplazar. Los métodos de creación, se invocan sobre el objeto *document*.

Ejemplos

```
// inserta un nuevo párrafo <p> al final
var p = document.createElement("p");
var texto = document.createTextNode("Párrafo nuevo");
p.appendChild(texto);
document.body.appendChild(p);

// inserta un nuevo párrafo <p> al principio
var nuevoP = document.createElement("p");
var texto = document.createTextNode("Segundo parrafo, antes del primero");
nuevoP.appendChild(texto);
var anteriorP = document.getElementsByTagName("p")[0];
document.body.insertBefore(nuevoP, anteriorP); // llama sobre el padre de ambos
```

Reemplazar nodos

```
var nuevoP = document.createElement("p");
var texto = document.createTextNode("Parrafo nuevo sustituye al original");
nuevoP.appendChild(texto);
var anteriorP = document.body.getElementsByTagName("p")[0];
anteriorP.parentNode.replaceChild(nuevoP, anteriorP);
```

Eliminar nodos

```
var p = document.getElementsByTagName("p")[0];
document.body.removeChild(p); // se invoca sobre el padre del nodo a eliminar
```

Atributos

```
var laImagen = document.getElementById("logo");

// acceder al valor de los atributos:
var archivo = laImagen.getAttribute("src");
var borde = laImagen.getAttribute("border");

// modificar atributos:
laImagen.setAttribute("src", "nuevo_logo.gif"); // set también crea si no existe
laImagen.setAttribute("border", "1");

// eliminar atributos
laImagen.removeAttribute("border");
```


Se pueden usar de modo alternativo, los métodos `createAttribute` y `setAttributeNode`:

```
var h1 = document.getElementsByTagName("H1")[0]; // primer <h1> del documento
var att = document.createAttribute("class");      // Crea un atributo "class"
att.value = "demo";                              // da valor al atributo
h1.setAttributeNode(att);                       // asocia el atributo al elemento
```

Es importante tener en cuenta que el contenido añadido o modificado por medio de JS no se tiene en cuenta para el posicionamiento de los buscadores web.

Objetos del documento

Para cada elemento de HTML se dispone del objeto que permite gestionarlo completamente. Cada uno de ellos contiene atributos y métodos tanto comunes a los nodos (`nextSibling`, p. ej.) como específicos (por ejemplo, `href` en el objeto *anchor*). A continuación se presentan algunos ejemplos:

Document

Este objeto dispone de multitud de campos y métodos: `URL`, `body`, `forms`, `images`, `lastModified` ...

```
<html>
<head> <meta charset="UTF-8"></head>
<body>
  El título de este documento es: <script>document.write(document.title);</script>
</body>
</html>
```

Anchor

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function cambiaDestino() {
      document.getElementById('uvigo').target="_blank";
    }
  </script>
</head>
<body>
  <a id="uvigo" href="http://www.uvigo.es"> Visita la web de la Universidad de Vigo
  </a><br><br>
  <input type="button" onclick=" cambiaDestino()" value="Cambia destino" />
  <p>Prueba el enlace antes y después de pulsar el botón</p>
</body>
</html>
```

Button

```
<html>
<head><meta charset="UTF-8"></head>
<body>
  <input type="button" id="boton1" value="Haz click aquí" />
  El texto del botón es:
  <script>document.write(document.getElementById("boton1").value);</script>
</body>
</html>
```

Form

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <form id="frm1" name="formularioEjemplo">
    Nombre: <input type="text" name="nombre" /><br>
    Apellido: <input type="text" name="apellido" /><br>
  </form>
  El nombre del formulario es:
  <script>
    document.write(document.getElementById("frm1").name);
  </script>
</body>
</html>
```

Image

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function cambiaDim() {
      document.getElementById("escudo").height="470";
      document.getElementById("escudo").width="264";
    }
  </script>
</head>
<body>
  <br><br>
  <input type="button" onclick="cambiaDim()" value="Cambia dimensiones" />
</body>
</html>
```

Cuando el navegador tiene que mostrar una imagen, antes de solicitar el envío desde el servidor, la busca entre todos los objetos *Image* disponibles, tratando de localizar uno con el mismo campo *src* que el solicitado. Por ello, se puede reducir los tiempos de ejecución descargando las imágenes antes de necesitarlas (por ejemplo cuando hay imágenes que cambian al pasar el cursor sobre ellas):

```
<html>
  <head>
    <script>
      function precarga() {
        imgPesada = new Image();
        imgPesada.src = "nueva.gif";
      }
    </script>
  </head>
  <body onLoad="precarga()">
    <a href="#" onMouseOver="document.img01.src='nueva.gif'">
      </a>
    </body>
</html>
```

Nótese que el objeto creado para descargar la imagen no se usa de forma explícita en ningún otro lugar del código.

También puede ser útil descargar todas las imágenes de la página:

```
var imagenes = ['imgs/imagen1.png', ' imgs/imagen2.jpg', ' imgs/imagen3.jpg'];
var imgObj = [];
for (var i=0; i<imagenes.length; i++) {
    imgObj[i] = new Image();
    imgObj[i].src = imagenes[i];
}
```

Option/Select

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function eliminaOpc() {
      var x=document.getElementById("miSeleccion");
      x.remove(x.selectedIndex);
    }
  </script>
</head>
<body>
  <form>
    <select id="miSeleccion">
      <option>Manzana</option>
      <option selected="true">Piña</option>
      <option>Naranja</option>
    </select>
    <input type="button" onclick="eliminaOpc()" value="Elimina seleccion">
  </form>
</body>
</html>
```

Table

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function celda(){
      var x=document.getElementById('miTabla').rows[0].cells;
      alert(x[0].innerHTML);
    }
  </script>
</head>
<body>
  <table id="miTabla" border="1">
    <tr> <td>celda 1,1</td> <td>celda 1,2</td> </tr>
    <tr> <td>celda 2,1</td> <td>celda 2,2</td> </tr>
  </table>
  <br>
  <input type="button" onclick="celda()" value="Muestra la primera celda">
</body>
</html>
```

Otros selectores

querySelector

Se invoca sobre el objeto *document* y devuelve el primer elemento que concuerda con el grupo de selectores especificados entre paréntesis, independientemente del nivel de anidamiento en el que se encuentre. Los selectores se declaran usando comillas y con la misma sintaxis que CSS. Pueden especificarse varios grupos de selectores separados por comas.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <script src="micodigo.js"></script>
</head>
<body>
  <div id="principal">
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```

micodigo.js

```
function hacerclic(){
  document.querySelector("#principal p:first-child").onclick=mostraralerta;
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

querySelectorAll

Este selector se invoca sobre el objeto *document* y devuelve un array con todos los elementos que concuerdan con el grupo de selectores declarados entre paréntesis, independientemente del nivel de anidamiento en el que se encuentre. El orden en el array es el de aparición en el documento.

```
function hacerclic(){
  var lista=document.querySelectorAll("#principal p");
  for(var f=0; f<lista.length; f++){
    lista[f].onclick=mostraralerta;
  }
}
function mostraralerta(){
  alert('hizo clic!');
}
window.onload=hacerclic;
```

Es importante tener en cuenta que los selectores se pueden combinar. El ejemplo anterior se podría reescribir con *querySelectorAll* y *getElementById*:

```
function hacerclic(){
    var lista=document.getElementById('principal').querySelectorAll("p");
    lista[0].onclick=mostraralerta;
}
function mostraralerta(){
    alert('hizo clic!');
}
window.onload=hacerclic;
```

Lista completa de selectores

Selector	Ejemplo	Descripción del ejemplo
.class	.intro	elementos con class="intro"
#id	#firstname	elemento con id="firstname"
*	*	todos los elementos
element	p	elementos <p>
element,element	div,p	elementos <div> y <p>
element element	div p	elementos <p> dentro de <div>
element>element	div>p	elementos <p> con padre <div>
element+element	div+p	elementos <p> inmediatamente tras <div> (mismo padre)
[attribute]	[target]	elementos con a target atributo
[attribute=value]	[target=_blank]	elementos con target="_blank"
[attribute~=value]	[title~=flower]	elementos con atributo <i>title</i> contiene "flower"
[attribute =value]	[lang =en]	elementos con atributo <i>lang</i> comenzando con "en"
:link	a:link	enlaces no visitados
:visited	a:visited	Enlaces visitados
:active	a:active	enlaces activos
:hover	a:hover	enlaces con ratón encima
:focus	input:focus	elemento <i>input</i> con el foco
:first-letter	p:first-letter	primera letra de elementos <p>
:first-line	p:first-line	primera línea de elementos <p>
:first-child	p:first-child	todo elemento <p> que sea primer hijo
:before	p:before	inserta contenido antes del contenido de los <p>
:after	p:after	Insert content after every <p> elemento
:lang(language)	p:lang(it)	todo <p> con atributo lang igual a "it"
element1-element2	p-ul	todo elemento precedido por un <p>
[attribute^=value]	a[src^="https"]	todo <a> cuyo atributo <i>src</i> empieza por "https"
[attribute\$=value]	a[src\$=".pdf"]	todo <a> cuyo atributo <i>src</i> termina por ".pdf"
[attribute*=value]	a[src*="uvigo"]	todo <a> cuyo atributo <i>src</i> contiene "uvigo"
:first-of-type	p:first-of-type	todo <p> que sea el primer <p> de su padre
:last-of-type	p:last-of-type	todo <p> que sea el último <p> de su padre
:only-of-type	p:only-of-type	todo <p> que sea el único <p> de su padre
:only-child	p:only-child	todo <p> que sea el único hijo de su padre
:nth-child(n)	p:nth-child(2)	todo elemento <p> que sea el 2º hijo de su padre
:nth-last-child(n)	p:nth-last-child(2)	todo <p> 2º hijo de su padre desde el último
:nth-of-type(n)	p:nth-of-type(2)	todo <p> que sea el 2º <p> de su padre
:nth-last-of-type(n)	p:nth-last-of-type(2)	todo <p> que sea el 2º <p> de su padre desde el último.
:last-child	p:last-child	todo <p> que sea el último hijo de su padre

:root	:root	elemento raíz del documento HTML
:empty	p:empty	todo <p> sin hijos (nodos de texto incluidos)
:target	#news:target	El anchor #news actualmente activo (click en una URL que contiene ese nombre <i>anchor</i>)
:enabled	input:enabled	todo elemento <input> habilitado (<i>enabled</i>)
:disabled	input:disabled	todo elemento <input> dishabilitado (<i>disabled</i>)
:checked	input:checked	todo elemento <input> marcado (<i>checked</i>)
:not(selector)	:not(p)	todo elemento que no sea un <p>
::selection	::selection	selección del usuario con el cursor en el documento

addEventListener

La función `addEventListener` permite asociar una función de respuesta a evento a un objeto del DOM.

```
<html>
<head>
  <script>
    function mostraralerta(){ alert('hizo clic!'); } // no usa argumento evento
    function hacerclic(){
      var elemento=document.getElementsByTagName('p')[0];
      elemento.addEventListener('click', mostraralerta);
    }
    window.addEventListener('load', hacerclic);
  </script>
</head>
<body>
  <div id="principal">
    <p>Hacer Clic</p>
    <p>No puede hacer Clic</p>
  </div>
</body>
</html>
```

El primer *addEventListener* del ejemplo está dentro de una función para que se ejecute cuando ya se haya cargado completamente la página y evitar así que aún no se haya cargado la función (esa es la función del segundo *addEventListener*).

La propagación de eventos indica el orden en el que ocurren los eventos cuando hay anidamiento. Supongamos que tenemos un elemento (*elemento 1*) que contiene un segundo elemento (*elemento 2*). Si se hace click en el elemento 2, se está haciendo click también en el elemento 1 por anidamiento. Si el último argumento de la función *addEventListener* recibe el valor *false* (valor por defecto), se trata primero el evento del elemento interno. En caso contrario (valor *true*) se trata primero el evento del elemento externo.

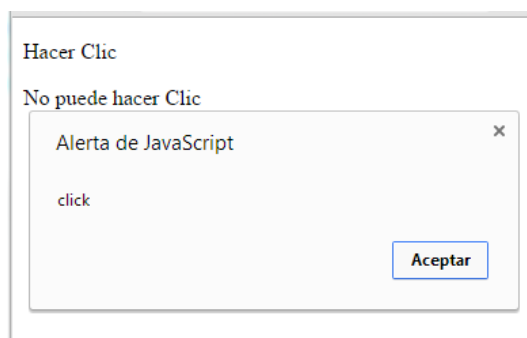
La propagación del evento se puede detener con la función *stopPropagation*:

```
function func1(evento) {
  alert("DIV 1");
  evento.stopPropagation();
}
```

La función de respuesta a un evento recibe como argumento un objeto con la información del evento como los atributos *timestamp* (ms.), *target* (objeto que generó el evento) o *type* ("click",

"mousedown", etc.). Cada tipo de evento tiene métodos y atributos específicos. Por ejemplo, el evento "click" devuelve las coordenadas del clic en *clientX* y *clientY*.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script>
      function mostraralerta(evento) {
        alert(evento.type);
      }
      function hacerclic() {
        var elemento = document.getElementsByTagName('p')[0];
        elemento.addEventListener('click', mostraralerta);
      }
      window.addEventListener('load', hacerclic);
    </script>
  </head>
  <body>
    <div id="principal">
      <p>Hacer Clic</p>
      <p>No puede hacer Clic</p>
    </div>
  </body>
</html>
```



En este ejemplo, al hacer clic en el primer párrafo (<p>), se mostrará una alerta indicando el tipo del evento disparado: click

Para pasar argumentos a la función de respuesta al evento, se tiene que utilizar funciones anónimas:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script>
      var p1 = 3; var p2 = 5;
      function hacerclic() {
        document.getElementById("boton").addEventListener("click", function() {
          multiplica(p1, p2); // usa los valores de p1 y p2 en el momento del
        });                  // disparo del evento (invocación de la función)
      }
      function multiplica(a, b) {
        var result = a * b;
        document.getElementById("parrafo").innerHTML = result;
      }
      window.addEventListener('load', hacerclic);
    </script>
  </head>
  <body>
    <p>paso de parámetros con la función addEventListener</p>
    <button id="boton">multiplica</button>
    <p id="parrafo"></p>
  </body>
</html>
```

Window

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function abre_ventanas() {
      window.open("http://www.uvigo.es/"); // acepta opciones de apertura
      window.open("http://momentum.uvigo.es/" , "_blank");
    }
  </script>
</head>
<body>
  <form>
    <input type="button" value="abre ventanas" onclick="abre_ventanas()">
  </form>
</body>
</html>
```

La palabra reservada *self* se refiere a la ventana actual. Por ello, `window.x` es equivalente a `self.x`

Imprimir la ventana

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function imprime(){
      window.print(); // abre el cuadro de impresión del sistema operativo
    }
  </script>
</head>
<body>
  <input type="button" value="Imprime esta página" onclick="imprime()" />
</body>
</html>
```

Otros métodos de *Window*

close (JS no puede cerrar ventanas que no haya abierto), *resizeBy*, *resizeTo*, *moveBy*, *moveTo*, *scrollTo*, *scrollBy*, etc.

History

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function retroceder() {
      window.history.back()
    }
  </script>
</head>
<body>
  <input type="button" value="retroceder" onclick="retroceder()" />
  <p>No actúa si es la primera página que se visita.</p>
</body>
</html>
```

Location

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function redirigir() {
      window.location.replace("http://momentum.uvigo.es")
    }
  </script>
</head>
<body>
  <input type="button" value="Redirige" onclick="redirigir()" />
</body>
</html>
```

Navigator

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <div id="ejemplo"></div>
  <script>
    txt= "<p> Nombre del navegador: " + navigator.appName + "</p>";
    txt+= "<p> Versión del navegador: " +
      navigator.appVersion + "</p>";
    txt+= "<p>Permite Cookies: " + navigator.cookieEnabled + "</p>";
    txt+= "<p>Plataforma: " + navigator.platform + "</p>"; // p. ej: Win32
    txt+= "<p>Cabecera: " + navigator.userAgent + "</p>"; // nombre+vers+plataf
    document.getElementById("ejemplo").innerHTML=txt;
  </script>
</body>
</html>
```

```
<html>
<head>
  <meta charset="UTF-8">
</head>
<body>
  <h3>Datos de tu pantalla:</h3>
  <script>
    // screen es un atributo de window: window.screen.width
    document.write("Ancho y alto totales: ");
    document.write(screen.width + " x " + screen.height);
    document.write("<br>");
    document.write("Profundidad de color: ");
    document.write(screen.colorDepth);
  </script>
</body>
</html>
```

Validación de formularios

Algunas comprobaciones típicas sobre formularios con JavaScript:

- ¿Ha dejado el usuario en blanco campos obligatorios?
- ¿Son válidas las direcciones de correo electrónico introducidas por el usuario?
- ¿Son válidas las fechas introducidas por el usuario?
- ¿Se ha introducido texto en un campo numérico?

Se debería comprobar también en el servidor (PHP, JEE, ASP...) por si en el navegador está deshabilitado el javascript.

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function validarFormulario() {
      var x=document.forms["miFormulario"]["nombre"].value
      if (x==null || x=="") {
        alert("Debe cubrirse el campo del nombre");
        return false;
      }
    }
  </script>
</head>
<body>
  <form name="miFormulario" action="procesa_form.php"
    onsubmit="return validarFormulario()" method="post">
    Nombre: <input type="text" name="nombre">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Como se verá más adelante, existe una forma alternativa de acceder a los elementos de la página: *document.anchors* (colección de elementos <a> con atributo *name*), *document.images* (colección de todos los elementos), *document.links* (colección de elementos <a> y <area> con atributo *href*), etc.

Esta función comprueba si la dirección de email contiene por lo menos una @ y un punto (.). Además, la @ no debe ser el primer carácter de la dirección y el punto debe aparecer tras la @ y por lo menos, dos caracteres antes del final:

```
<html>
<head>
  <meta charset="UTF-8">
  <script>
    function validarFormulario() {
      var x=document.forms["miFormulario"]["email"].value
      var posArroba=x.indexOf("@");
      var posPunto=x.lastIndexOf(".");
      if (posArroba<1 || posPunto < posArroba+2 || posPunto+2 >= x.length) {
        alert("No es una dirección de correo electrónico válida");
        return false;
      }
    }
  </script>
</head>
<body>
  <form name="miFormulario" action="procesa_form.php"
    onsubmit="return validarFormulario();" method="post">
    Email: <input type="text" name="email">
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Cookies

Escritura y borrado

Estructura: nombre=valor[;expires=fecha[;path=ruta[;domain=subdominio[;secure]]]]

Ejemplo: document.cookie = 'micookie=hola;expires=Fri, 17 Dec 2010 10:00:00 GMT';

```
function crearCookie (nombre,valor,dias) {
  if (dias) { // sin fecha, la cookie se borra cuando se cierra el navegador
    var fecha = new Date();
    fecha.setTime(fecha.getTime()+(dias*24*60*60*1000));
    var expira = "; expires="+fecha.toGMTString();
  }
  else var expira = "";
  document.cookie = nombre + "=" + valor + expira + "; path=/";
}

function borrarCookie(nombre) { // parar borrar se crea la cookie caducada
  crearCookie(nombre,"",-1);
}
```

Lectura

Instrucción: document.cookie

Devuelve todas las cookies del dominio: nombre4=valor4; nombre3=valor3; ... nombre1=valor1

```
function leerCookie(nombre) {
    var nombreEQ = nombre + "=";
    var ca = document.cookie.split(';'); // divide en array usando separador ';'
    for(var i=0;i < ca.length; i++) {
        var c = ca[i];
        c = c.trim(); // elimina espacios al inicio y al final
        if (c.indexOf(nombreEQ) == 0)
            return c.substring(nombreEQ.length,c.length); // (inicial, final+1)
    }
    return null;
}
```

Expresiones regulares

Las expresiones regulares permiten comprobar si una cadena de texto cumple un determinado formato, como por ejemplo, si todo son dígitos, si tiene formato de fecha y hora, etc. También permiten extraer partes de esa cadena de texto que cumplan ese formato, por ejemplo, si dentro de un texto más o menos largo hay una fecha, podemos extraerla fácilmente.

Expresiones simples

Una expresión regular se construye con una cadena de texto que representa el formato que debe cumplir otro texto. En javascript se puede hacer de dos formas: instanciando la clase RegExp pasando como parámetro la cadena de formato (poco recomendada), o bien poniendo directamente la cadena de formato delimitada por el carácter / en lugar de las comillas

```
// Son equivalentes
var reg = new RegExp("aa")
var reg = /aa/
```

Obtener un array con las ocurrencias de una determinada secuencia de letras o números fija es fácil usando el método *match()*:

```
// devuelve un array de 1 elemento ["javascript"], indicando
// que existe esa cadena dentro del texto:
var reg = /javascript/;
"hola javascript".match(reg);

// devuelve null: no existe 'javascript' dentro de "Hola Lara".
"Hola Lara".match(reg);
```

No es necesario definir la expresión regular antes:

```
// Devuelve ["javascript"]
"hola javascript".match(/javascript/);
```

Y para verificar si existe o no la cadena, se puede usar un if:

```
if ("hola javascript".match(/javascript/) {  
    ...  
}
```

Caracteres no alfabéticos ni numéricos

Algunos de los caracteres no numéricos ni alfabéticos tienen un significado especial (lo vemos más adelante), como por ejemplo [] { } () * . ^ \$ etc. No se pueden poner tal cual si forman parte de nuestro formato, debemos usar el carácter \ delante para indicar que no es un carácter de formato:

```
// Devuelve ["*"] indicando que existe un asterisco.  
"esto es un *".match(/\*/);
```

Conjunto opcional de caracteres

A veces es indiferente si una letra, por ejemplo, es mayúscula o minúscula, o se desea que sea una vocal, o un dígito. Cuando se quiere exigir que una de las letras del texto pueda ser una cualquiera de un conjunto de letras determinado, las ponemos entre [] en la expresión. Por ejemplo, si vale "Javascript" y "javascript", se puede poner la expresión como /[Jj]avascript/ para indicar que vale J mayúscula o j minúscula:

```
// Sí encuentra la cadena:  
"javascript con minúscula".match(/[Jj]avascript/);  
// También la encuentra:  
"Javascript con minúscula".match(/[Jj]avascript/);
```

Si los caracteres válidos son varios y van ordenados según el juego de caracteres, se puede poner el primero y el último separados por un guión (-). Por ejemplo, [a-z] vale para cualquier letra minúscula, [0-9] para cualquier dígito y [a-zA-Z] para cualquier letra mayúscula o minúscula

```
"aa2bb".match(/[0-9]/); // encuentra el 2, devolviendo ["2"].
```

Se puede hacer lo contrario, es decir, imponer que la letra no esté en el conjunto de caracteres, poniendo el juego de caracteres que no queremos entre [^ y]. Por ejemplo, para no dígitos pondríamos [^0-9]:

```
"22 33".match(/[^0-9]/); // encuentra el espacio en blanco, devolviendo [" "]
```

Metacaracteres

Hay varios conjuntos que se usan con frecuencia, como el de letras [a-zA-Z], el de dígitos [0-9] o el de espacios en blanco (espacio, tabulador, etc). Para estos conjuntos y otros caracteres especiales, se tienen los siguientes metacaracteres:

- \s cualquier espacio en blanco (espacios en blanco, tabuladores, etc.)
- \S cualquier carácter excepto un espacio en blanco de los representados por \s
- \w cualquier carácter alfanumérico, incluyendo el guión bajo (equivale a [a-zA-Z0-9_])
- \W cualquier carácter no alfanumérico ni guión bajo (equivale a [^a-zA-Z0-9_])
- \d cualquier dígito (equivale a [0-9])

<code>\D</code>	cualquier carácter excepto un dígito (equivalente a <code>[^0-9]</code>)
<code>\r</code>	retorno de carro
<code>\n</code>	alimentación de línea
<code>\t</code>	tabulador horizontal
<code>\v</code>	tabulador vertical
<code>\0</code>	carácter NUL
<code>[\b]</code>	carácter <i>backspace</i> (borrado)
<code>\b</code>	límite de palabra (la posición entre una palabra y el espacio)
<code>\B</code>	el complementario del anterior: <code>[^\b]</code>
<code>\cx</code>	carácter de control X (por ejemplo, <code>\cm</code> para control-M)
<code>\xh</code>	carácter con el código hexadecimal h (dos dígitos)
<code>\uh</code>	carácter con el código hexadecimal h (cuatro dígitos)

Por ejemplo:

```
"aa2bb".match(/\d/); // encuentra el 2, devolviendo ["2"]
```

Cuantificadores

Tomemos como ejemplo la siguiente expresión que permite buscar un conjunto de tres dígitos, repitiendo tres veces el metacarácter `\d`

```
"aa123bb".match(/\d\d\d/); // encuentra el 123, devolviendo ["123"]
```

Este enfoque es poco práctico si hay muchos caracteres. Las expresiones regulares permiten indicar un rango de veces que debe repetirse usando los caracteres `{}`.

Por ejemplo:

- Para buscar tres dígitos en la cadena: `/\d{3}/`
- Para buscar entre uno y cinco dígitos en la cadena: `/\d{1,5}/`
- Para buscar entre dos o más dígitos en la cadena: `/\d{2,}/`

Como resumen de los cuantificadores anteriores y tres más:

<code>{n}</code>	n veces
<code>{n,}</code>	n o más veces
<code>{n,m}</code>	entre n y m veces
<code>*</code>	cero o más veces; equivale a <code>{0,}</code>
<code>+</code>	una o más veces; equivale a <code>{1,}</code>
<code>?</code>	una o ninguna vez; equivale a <code>{0,1}</code>

Ejemplos

```
"1234".match(/\d{2}/); // encuentra 12
"1234".match(/\d{1,3}/); // encuentra 123
"1234".match(/\d{3,10}/); // encuentra 1234
"a2a".match(/a\d+a/); // encuentra a2a
"a234a".match(/a\d+a/); // encuentra a234a
```

Los cuantificadores `*` y `+` encuentran el máximo posible de caracteres. Por ejemplo, si el patrón es `/a+/` y la cadena es `"aaaaa"`, el resultado será toda la cadena:

```
"aaaaa".match(/a+/);    // devuelve aaaaa
```

Para que se encuentre lo menos posible, se pone un `?` detrás. Así por ejemplo, si la expresión regular es `/a+?/` y nuestra cadena es `"aaaaa"`, sólo se encontrará una `"a"`

```
"aaaaa".match(/a+?/);    //devuelve aaaaa
```

El comportamiento inicial se conoce como "greedy" o codicioso, en el que el patrón intenta coger la mayor parte de la cadena de texto posible. En segundo comportamiento se conoce como "nongreedy" o no codicioso, en el que el patrón coge lo menos posible de la cadena.

Alternativos

El carácter `|` permite la existencia de dos contenidos: por ejemplo, su uso en el patrón `/(a|b)a/` encaja con `'aa'` en `"contraatacar"` y `'ba'` en `"trabajar"`.

Capturas

A veces interesa no sólo saber si una cadena cumple un determinado patrón, sino extraer determinadas partes de él. Por ejemplo, si una fecha está en el formato `"27/11/2012"` puede interesar extraer los números. Una expresión regular que vale para esta cadena puede ser esta:

```
/\d{1,2}\d{1,2}\d{4}/
```

Si el día y el mes pueden tener un dígito y el año obligatoriamente cuatro:

```
"27/11/2012".match(/\d{1,2}\d{1,2}\d{4}/);
```

En este caso nos devuelve un array con un único elemento que es la cadena `"27/11/2012"`. Para extraer las partes, únicamente se debe poner entre paréntesis en la expresión regular aquellas partes que interesan:

```
/(\d{1,2})\d{1,2}\d{4}/
```

Si ahora se ejecuta el método `match()` con la misma cadena anterior, se obtendrá un array de cuatro cadenas: la primera será la cadena completa que cumple la expresión regular. Los otros tres elementos serán lo que cumple cada uno de los paréntesis.

```
"27/11/2012".match(/(\d{1,2})\d{1,2}\d{4}/);  
// devuelve el array: ["27/11/2012", "27", "11", "2012"]
```

Los paréntesis también sirven para agrupar una parte de la expresión y añadir detrás uno de los cuantificadores. Por ejemplo

```
"xyxyxyxy".match(/(xy)+/);    // se cumple, ya que xy aparece una o más veces.
```

Referencias (*backreferences*)

Las partes de la cadena que cumplen la parte de expresión regular entre paréntesis, se pueden reutilizar en la misma expresión regular. Se puede hacer referencia a estas partes indicando su posición precedida de una barra \. La numeración empieza en cero.

Por ejemplo, verificar que una cadena de texto va cerrada entre comillas del mismo tipo, es decir, buscar algo como 'esto' o "esto", pero no 'esto': `/(['"]).*\1/`

La expresión busca una comilla simple (') o una doble (") con `['"]`. Lo que se encuentre se guardará debido a que va entre paréntesis `(['"])` y a partir de ahí vale cualquier conjunto de caracteres terminados por lo extraído en el grupo 1 `(\1)`:

```
var s="'hola tú' ¿qué tal\n?".match(/(['"]).*\1/);
document.write(s); // muestra: 'hola tú','

var s='"hola tú' ¿qué tal\n?".match(/(['"]).*\1/);
document.write(s); // devuelve null: la cadena comienza con " y termina en '
```

Sin captura

A veces interesa encontrar una secuencia que se repita varias veces seguidas y la forma de hacerlo es con los paréntesis, por ejemplo, si ponemos `/(pe){2}/` estamos buscando "pepe". Para evitar que esos paréntesis guarden lo encontrado en `\1`, podemos poner `?:`, tal que así `/(?:pe){2}/`, de esta forma encontraremos "pepe", pero se nos devolverá el trozo "pe" encontrado ni lo tendremos disponible en `\1`:

```
"pepe".match(/(pe){2}/); // devuelve ["pepe", "pe"]
"pepe".match(/(?:pe){2}/); // devuelve ["pepe"]
```

Ubicación

Los siguientes caracteres también tienen un significado especial:

`^` representa el principio de cadena:

```
"hola Pedro".match(/^hola/); // devuelve ["hola"]
"pues hola Pedro".match(/^hola/); // devuelve null
```

`$` representa el final de la cadena:

```
"hola Pedro".match(/Pedro$/); // devuelve ["Pedro"]
"hola Pedro".match(/Pe$/); // devuelve null
```

`\b` representa una frontera de palabra, es decir, entre un caracter "letra" y cualquier otra cosa como espacios, fin o principio de línea, etc.

```
"java es divertido".match(/\bjava\b/); // devuelve ["java"]
"javascript es divertido".match(/\bjava\b/); // devuelve null
```


\B es lo contrario de \b, así por ejemplo, /\bjava\b/ buscará una palabra que empiece por "java", pero no sea sólo java sino que tenga algo más

```
"java es divertido".match(/\bjava\b/); // devuelve null
"javascript es divertido".match(/\bjava\b/); // devuelve ["java"]
```

Lookaheads

Detecta sólo si la subexpresión precedente va seguida del patrón, pero éste no es parte de la coincidencia.

(?=expresion) posiciona el resto de la expresión regular y busca antes o después. Por ejemplo si queremos buscar un número que vaya delante de Km, podemos hacer esto /\d+(?= Km) /. Es decir, uno o más dígitos seguidos de un espacio y las letras km. La diferencia con la expresión (/ \d+ km/) es que en el primer caso sólo casan con la expresión los números, mientras que en el segundo caso se incluye también el " Km"

```
"11 millas 10 Km".match(/\d+(?= Km) /); // devuelve ["10"]
"11 millas 10 Km".match(/\d+ Km/); // devuelve ["10 Km"]
```

Hay que tener cuidado si buscamos detrás, porque como la construcción (?=expresion) no se tiene en cuenta, sigue contando para el resto de la expresión. Por ejemplo, si queremos extraer la parte decimal de "11.22" podríamos pensar en hacer esto /(=?\.) \d+/, pero no funciona porque el punto decimal no se "consume" con (=?\.), así que debemos tenerlo en cuenta y ponerlo detrás:

```
"11.22".match(/(=?\.) \d+/); // devuelve null
"11.22".match(/(=?\.) \. \d+/); // devuelve [".22"]
```

(?!expresion) hace lo contrario que (?=expresion), es decir, busca una posición donde no se cumpla expresión. Por ejemplo, para sacar lo que no sean km de "11 km, 12 km, 14 m":

```
"11 km 12 km 14 m".match(/\d{2} (?! km) /); // devuelve ["14"]
```

Flags de opciones

Tras una expresión regular /expresion/ se pueden poner algunos caracteres llamados *flags* que cambian algo el comportamiento:

i sirve para ignorar mayúsculas y minúsculas

```
"hola".match(/HOLA/); // devuelve null
"hola".match(/HOLA/i); // devuelve ["hola"]
```

g permite buscar todas las veces posibles la expresión, no sólo una vez

```
"11 223 44 66 77".match(/\d+/); // devuelve ["11"]
"11 223 44 66 77".match(/\d+/g); // devuelve ["11", "223", "44", "66", "77"]
```

m busca en cadenas con retornos de carro \n considerando estos como inicios de línea ^ o fin \$

```
"hola\namigo".match(/^amigo/); // devuelve null
"hola\namigo".match(/^amigo/m); // devuelve ["amigo"]
"hola\namigo".match(/hola$/); // devuelve null
"hola\namigo".match(/hola$/m); // devuelve ["hola"]
```

Métodos de cadena

Para todos estos ejemplos se ha usado el método `match()` de la clase `String`, ya que devuelve un array con los elementos que se encuentran. Sin embargo, tanto `String` como `RegExp` tienen otros métodos útiles

search

`String.search(/expresion/)` devuelve la posición donde se encuentra la expresión dentro de la cadena, o `-1` si no se encuentra.

replace

`String.replace(/expresion/, cadena)` busca la parte de cadena que concuerda con la expresión y la reemplaza con lo que le pasemos en el argumento *cadena*. Este método tiene además un detalle interesante: cuando en la expresión regular hay paréntesis para extraer algunas partes de la cadena, la misma expresión regular recuerda qué ha encontrado. En el método `replace`, si en la cadena de segundo parámetro aparecen cosas como `$1`, `$2`, utilizará lo encontrado.

```
"ho3la".replace(/\d/, "X");           // devuelve "hoXla"
"ho3la".replace(/(\d)/, "-$1-");      // devuelve "ho-3-la"
```

match

`String.match(/expresion/)`: devuelve un array con los elementos que se encuentran.

split

`String.split(/expresion/)`: usa lo que sea que coincida con la expresión como separador y devuelve un array con la cadena dividida en partes por ese separador

```
"hola, dos tres; cuatro".split(/\W+/);
// devuelve ["hola", "dos", "tres", "cuatro"]
```

exec

`RegExp.exec()`: es similar a `match()` de `String`, pero sólo devuelve un resultado y hace que `RegExp` guarde la posición en la que lo ha encontrado. Sucesivas llamadas a `exec()`, nos irán devolviendo los siguientes resultados

```
var reg = new RegExp("\\d+");
reg.exec("11 22 33");      // devuelve ["11"]
reg.exec("11 22 33");      // vuelve a devolver ["11"], ya que no hay flag g

var reg = new RegExp("\\d+", "g");
reg.exec("11 22 33");      // devuelve ["11"]
reg.exec("11 22 33");      // vuelve a devolver ["22"], ya que hay flag g
reg.exec("11 22 33");      // vuelve a devolver ["33"], ya que hay flag g
reg.exec("11 22 33");      // devuelve null, ya que no hay más resultados
reg.exec("11 22 33");      // tras null reinicializa → devuelve ["11"] otra vez
```

test

`RegExp.test()`: similar a `exec()`, pero en vez de devolver lo encontrado, devuelve *true* si ha encontrado algo o *false* si no. Como la expresión regular recuerda las búsquedas anteriores, sucesivas llamadas a `test()` pueden devolver resultados distintos:

```
var reg = new RegExp("\\d+", "g");
reg.test("11 22 33"); // devuelve true, porque encuentra el ["11"]
reg.test("11 22 33"); // devuelve true porque encuentra ["22"] (hay flag g)
reg.test("11 22 33"); // devuelve true porque encuentra ["33"] (hay flag g)
reg.test("11 22 33"); // devuelve false porque ya no hay más resultados
reg.test("11 22 33"); // tras null reinicializa → devuelve true por el 1er 11
```

constructor

Además de crear las expresiones regulares con */expresion/flags*, podemos hacerlo con el operador *new* de la clase `RegExp`, por ejemplo *new RegExp("expresion", "flags")*.

```
var reg = new RegExp("\\d+", "g");
"11 22 33".match(reg); // devuelve ["11", "22", "33"]
```

Es imprescindible escapar las `\` con `\\`

VIDEO

Reproducción básica de vídeo

No todos los navegadores aceptan todos los formatos, por lo que se suele proporcionar los enlaces a vídeos en varios formatos: mp4, ogg, webm, 3gp ...

El elemento Vídeo es de tipo *block*.

```
<video id="video1" width="720" height="400" preload controls loop
      poster="http://momentum.uvigo.es/multimedia/poster.jpg">
  <source src="http://momentum.uvigo.es/multimedia/trailer.mp4" type="video/mp4">
  <source src="http://momentum.uvigo.es/multimedia/trailer.ogg" type="video/ogg">
  Este navegador no reproduce vídeo
</video>
```

El atributo *controls* muestra los controles de forma distinta según el navegador. En algunos no se reconoce esta etiqueta. Si se modifica *width* o *height*, el vídeo se redimensiona de acuerdo al menor de ellos, manteniendo la proporción original (4/3, 16/9, etc.)

Otros atributos:

- *type*: formato del archivo. Puede especificar los codecs:
`type="video/ogg; codecs=dirac, speex"`
- *autoplay* (se reproduce en cuanto es posible)
- *poster*: imagen inicial
- *loop*: reproducción en bucle
- *preload* : *auto* (cargar todo el vídeo cuando se carga la página), *metadata* (sólo carga los metadatos como dimensión, duración, primer cuadro...), *none* (no se precarga nada)

Propiedades métodos y eventos

Propiedades

volume, *pause*, *ended*, *duration*, *currentTime* (lectura y escritura), *error* (valor), *muted*, *played*, *buffered* ... Esta última contiene un campo *length* que indica cuántos intervalos hay disponibles y dos arrays con los inicios (*start*) y los fines (*end*) de los intervalos de tiempo disponibles:
`video1.buffered.start(0)`, p. ej.

Métodos

play (inicia o continua tras pausa), *pause*, *load*, *canPlayType* (formato soportado o no), *addTextTrack* (añade una nueva pista de texto);

Eventos

- **progress**: se dispara periódicamente para informar del progreso de descarga. La información de los tramos descargados se guarda en la propiedad *buffered*.
- **canplaythrough**: disparado cuando se estima, en función de la velocidad de descarga, que se puede reproducir totalmente.
- **canplay**: cuando ya se puede empezar a reproducir.
- **play**: se comenzó a reproducir.
- **waiting, seeked, ended, pause, error...**

Programación de un reproductor de vídeo propio

El objetivo es evitar las diferencias entre navegadores de los controles estándar que proporciona la etiqueta *video*

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Reproductor de vídeo</title>
  <link rel="stylesheet" href="reproductor.css">
  <script src="reproductor.js"></script>
</head>
<body>
  <section id="reproductor">
    <video id="video" width="720" height="400">
      <source src="http://momentum.uvigo.es/multimedia/trailer.mp4">
      <source src="http://momentum.uvigo.es/multimedia/trailer.ogg">
    </video>
    <nav>
      <div id="botones">
        <input type="button" id="reproducir" value="Reproducir"> <!-- tb pausa -->
        <input type="button" id="silencio" value="Silencio">
      </div>
      <div id="barra">
        <div id="progreso"></div>
      </div>
      <div id="control">
        <input type="range" id="volumen" min="0" max="1" step="0.1" value="0.6">
      </div>
      <div class="clear"></div>
    </nav>
  </section>
</body>
</html>
```

Fichero reproductor.css

```
#botones {
  float: left;
  width: 185px;
  height: 20px;
  padding-left: 5px;
}
```

```

#reproductor {
  width: 720px;
  margin: 20px auto;
  padding: 10px 5px 5px 5px;
  background: #999999;
  border: 1px solid #666666;
  border-radius: 10px;
}
#reproducir, #silencio {
  padding: 5px 10px;
  width: 90px;
  border: 1px solid #000000;
  background: #DDDDDD;
  font-weight: bold;
  border-radius: 10px;
}
nav {
  margin: 5px 0px;
}
#barra {
  float: left;
  width: 400px;
  height: 16px;
  padding: 2px;
  margin: 8px 5px;
  border: 1px solid #CCCCCC;
  background: #EEEEEE;
}
#progreso {
  width: 0px; /* se incrementará proporcionalmente a la reproducción */
  height: 16px;
  background: rgba(0,0,150,.2);
}
.clear {
  clear: both;
}

```

Fichero reproductor.js (con algunas versiones de Chrome da problemas el método *mover*)

```

// inicializamos la aplicación:
var maximo, video, reproducir, barra, progreso, silencio, volumen, bucle;

function inicializar() {
  video = document.getElementById("video");
  reproducir = document.getElementById("reproducir"); // botón
  barra = document.getElementById("barra"); // div contenedor de progreso
  progreso = document.getElementById("progreso"); // div dentro de barra
  silencio = document.getElementById("silencio"); // botón
  volumen = document.getElementById("volumen"); // botón

  reproducir.addEventListener("click", pulsar);
  silencio.addEventListener("click", silenciar);
  barra.addEventListener("click", mover);
  volumen.addEventListener("change", nivel);

  // obtenemos el valor de máximo (400) del CSS:
  maximo = parseInt(window.getComputedStyle(barra).getPropertyValue('width'));
}

```

```

// reproducir y pausar
function pulsar() {
    if (!video.paused && !video.ended) {
        video.pause();
        reproducir.value = "Reproducir";
        clearInterval(bucle);
    } else {
        video.play();
        reproducir.value = "Pausar";
        bucle = setInterval(estado, 1000);
    }
}

// actualizar la barra de progreso
function estado() {
    if (!video.ended) {
        var tamano = parseInt(video.currentTime * maximo / video.duration);
        progreso.style.width = tamano + "px";
    } else {
        progreso.style.width = "0px";
        reproducir.value = "Reproducir";
        clearInterval(bucle);
    }
}

// selección del instante de reproducción (no funciona en algunos navegadores)
function mover(event) {
    if (!video.paused && !video.ended) { // movemos si está reproduciendo
        var ratonX = event.offsetX - 2;    // 2 es el padding de la barra
        if (ratonX < 0) {
            ratonX = 0;
        } else if (ratonX > maximo) {
            ratonX = maximo;
        }
        var nuevotiempo = ratonX * video.duration / maximo;
        video.currentTime = nuevotiempo;
        progreso.style.width = ratonX + "px";
    }
}

// conmutar silencio
function silenciar() {
    if (silencio.value === "Silencio") {
        video.muted = true;
        silencio.value = "Sonido";
    } else {
        video.muted = false;
        silencio.value = "Silencio";
    }
}

// control del volumen
function nivel() {
    video.volume = volumen.value;
}

window.addEventListener("load", inicializar);

```

Subtítulos

Con el objeto *Track* es posible introducir pistas de subtítulos en los vídeos. En el siguiente ejemplo se proporcionan dos pistas de subtítulos, estableciendo la segunda como opción por defecto.

Los ficheros con los subtítulos suelen tener extensión *vtt* aunque no es obligatoria. Son ficheros de texto plano, que permiten formateo por medio de etiquetas HTML.

Las pistas no funcionan en local. Es decir, que la página tiene que descargarse de un servidor web.

Ejemplo básico

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Vídeo con subtítulos</title>
</head>
<body>
  <section>
    <video width="720" height="400" controls>
      <source src="http://momentum.uvigo.es/multimedia/trailer.mp4">
      <source src="http://momentum.uvigo.es/multimedia/trailer.ogg">
      <track label="English" kind="subtitles"
        src="subtitulos.en.vtt" srclang="en">
      <track label="Castellano" kind="subtitles"
        src="subtitulos.es.vtt" srclang="es" default>
    </video>
  </section>
</body>
</html>
```

Fichero subtitulos.es.vtt:

WEBVTT

```
00:02.000 --> 00:07.000
<i>Bienvenidos</i>
al elemento <track>
```

} *cue*

```
00:10.000 --> 00:15.000
Este es un ejemplo sencillo.
```

```
00:17.000 --> 00:22.000
Se pueden usar múltiples pistas simultáneamente ...
```

```
00:22.000 --> 00:25.000
para proporcionar texto en lenguajes diferentes
```

```
00:27.000 --> 00:30.000
<b>¡Hasta la vista!</b>
```


Acceso a la información de las pistas

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Pistas</title>
  <style>
    #reproductor, #info{
      float: left;
    }
  </style>
  <script>
    function inicializar() {
      var info = document.getElementById("info");
      var mipista = document.getElementById("mipista");
      var obj = mipista.track;
      var lista = "";
      lista += "<br>Tipo: " + obj.kind;
      lista += "<br>Etiqueta: " + obj.label;
      lista += "<br>Idioma: " + obj.language;
      info.innerHTML = lista;
    }
    window.addEventListener("load", inicializar);
  </script>
</head>
<body>
  <section id="reproductor">
    <video id="video" width="720" height="400" controls>
      <source src="trailer.mp4">
      <source src="trailer.ogg">
      <track id="mipista" kind="subtitles" label="Castellano"
        src="subtitulos.txt" srclang="es">
    </video>
  </section>
  <aside id="info"></aside>
</body>
</html>
```

Mostrar información de las cues

```
function inicializar() {
  var info = document.getElementById("info");
  var mipista = document.getElementById("mipista");
  var obj = mipista.track;
  var miscues = obj.cues;

  var lista = "";
  for (var f = 0; f < miscues.length; f++) {
    var cue = miscues[f];
    lista += cue.startTime + " - "; // size, textAlign, etc.
    lista += cue.text + "<br>";
  }
  info.innerHTML = lista;
}
window.addEventListener("load", inicializar);
```

Añadir pistas y cues desde javascript

```
function inicializar (){
    var cue;
    var cues = [
        { start: 2.000, end: 7.000, text: "Hola."},
        { start: 10.000, end: 15.000, text: "Este es un ejemplo"},
        { start: 15.001, end: 20.000, text: "de como añadir"},
        { start: 20.001, end: 25.000, text: "una pista nueva"},
        { start: 27.000, end: 30.000, text: "¡Hasta la vista!"},
    ];
    var video = document.getElementById("video");
    var nuevapista = video.addTextTrack("subtitulos");
    nuevapista.mode = "showing"; // disabled, hidden o showing
    for (var f = 0; f < cues.length; f++) {
        cue = new VTTCue(cues[f].start, cues[f].end, cues[f].text);
        nuevapista.addCue(cue);
    }
    video.play();
}
window.addEventListener("load", inicializar);
```

AUDIO

Reproducción básica de audio

Algunos navegadores necesitan el format *ogg* y otros *mp3*, por lo que se suele proporcionar los enlaces a vídeos en ambos formatos. Al igual que el video, el elemento audio es de tipo *block*.

Excepto *poster*, *width* y *height*, comparte atributos comunes con el objeto video: *src*, *controls*, *autoplay*, *loop*, *preload*. Las dimensiones se pueden fijar con CSS: `<audio style="width: 200px;">`

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Reproductor de Audio</title>
</head>
<body>
    <section id="reproductor">
        <audio id="video" controls>
            <source src="http://momentum.uvigo.es/multimedia/beach.mp3" type="audio/mpeg">
            <source src="http://momentum.uvigo.es/multimedia/beach.ogg" type="audio/ogg">
            Su navegador no soporta la etiqueta de audio
        </audio>
    </section>
</body>
</html>
```

Envío de un formulario desde Javascript

Se puede usar la función *submit()* sobre un formulario para provocar su envío.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    function iniciar() {
      var boton = document.getElementById("envio");
      boton.addEventListener("click", enviar);
    }
    function enviar() {
      var formulario = document.querySelector("form[name='informacion']");
      formulario.submit();
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p>
        <label>Email:
          <input type="email" name="miemail" id="miemail" required>
        </label>
      </p>
      <p><button type="button" id="envio">Enviar</button></p>
    </form>
  </section>
</body>
</html>
```

De forma alternativa, podemos obtener la referencia al formulario por medio de la propiedad *forms*. En el ejemplo anterior, la función *enviar()* cambiaría de este modo:

```
function enviar() {
  var lista = document.forms;
  var formulario = lista[0];
  formulario.submit();
}
```

Comprobando la validez del formulario

Si queremos comprobar la validez del formulario antes de enviarlo, se usaría la función *checkValidity()*. De este modo, la función *enviar* quedaría así:

```
function enviar() {
    var formulario = document.querySelector("form[name='informacion']");
    var valido = formulario.checkValidity();
    if (valido) {
        formulario.submit();
    } else {
        alert("Error: el formulario no puede enviarse");
    }
}
```

setCustomValidity()

setCustomValidity() se usa para establecer el mensaje de error que se mostrará cuando se intente enviar el formulario. Si el argumento no está vacío, el método asume que la validación ha fallado y muestra el argumento como mensaje al enviar el formulario. Si el argumento es una cadena vacía, se considera que el elemento ha superado la validación y se resetea el mensaje de error.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Formularios</title>
    <meta charset="UTF-8">
    <script>
        var nombre, apellidos;
        function iniciar() {
            nombre = document.getElementById("nombre");
            apellidos = document.getElementById("apellidos");
            // evento input: cuando cambia el contenido de un elemento input o textarea
            nombre.addEventListener("input", validacion);
            apellidos.addEventListener("input", validacion);
            validacion();
        }
        function validacion() {
            if(nombre.value === "" && apellidos.value === ""){
                // mensj sólo será visible cuando el usuario intente enviar el formulario:
                nombre.setCustomValidity('inserte nombre o apellidos');
                nombre.style.background='#FFDDDD';
                apellidos.style.background='#FFDDDD';
            } else {
                nombre.setCustomValidity(""); // si no hay problemas, reseteamos mensaje
                nombre.style.background='#FFFFFF';
                apellidos.style.background='#FFFFFF';
            }
        }
        window.addEventListener("load", iniciar);
    </script>
</head>
<body>
    <form name="registro" method="get">
        <p>Nombre: <input type="text" name="nombre" id="nombre"></p>
        <p>Apellido: <input type="text" name="apellidos" id="apellidos"></p>
        <input type="submit" id="envio" value="Entrada">
    </form>
</body>
</html>
```

El evento *invalid*

Se dispara al detectar un campo inválido al solicitar el envío o al llamar al método *checkValidity()* sobre el campo o su formulario. Se dispara para cada campo inválido en el formulario.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <script>
    var formulario;
    function iniciar() {
      var boton = document.getElementById("envio");
      boton.addEventListener("click", enviar);
      formulario = document.querySelector("form[name='informacion']");
      formulario.addEventListener("invalid", validacion, true);
    }
    function validacion (evento) {
      var elemento = evento.target;
      elemento.style.background = "#FFDDDD";
    }
    function enviar() {
      var valido = formulario.checkValidity();
      if (valido) {
        formulario.submit();
      }
    }
    window.addEventListener("load", iniciar);
  </script>
</head>
<body>
  <section>
    <form name="informacion" method="get" action="procesar.php">
      <p><label>Usuario: <input pattern="[A-Za-z]{3,}" name="usuario" id="usuario"
        maxlength="10" required></label></p>
      <p><label>Email: <input type="email" name="miemail"
        id="miemail" required></label></p>
      <p><button type="button" id="envio">Entrar</button></p>
    </form>
  </section>
</body>
</html>
```

El método *checkValidity()* da valor a los campos del objeto *.validity* y dispara el evento *invalid* si la validación falla. No existe el evento contrario (si la validación tiene éxito).

Validación de campos en tiempo real

Los atributos del objeto *.validity* (clase *ValidityState*) permiten trabajar de un modo más práctico. En el ejemplo anterior se añadiría la función *compruebaValor* y se añadiría una línea al final de la función *iniciar*:

```
function iniciar() {
    var boton = document.getElementById("envio");
    boton.addEventListener("click", enviar);
    formulario = document.querySelector("form[name='informacion']");
    formulario.addEventListener("invalid", validacion, true);
    formulario.addEventListener("input", compruebaValor);
}
function compruebaValor(evento) {
    var elemento = evento.target;
    if (elemento.validity.valid)
        elemento.style.background = "#FFFFFF";
    else
        elemento.style.background = "#FFDDDD";
}
```

Otros atributos de validity son: *valueMissing* (requerido pero falta), *typeMismatch* (p.ej. email sin @), *patternMismatch* (incoherente con campo *pattern*), *tooLong*, *rangeUnderflow* (valor menor que el indicado por el campo *min*), *rangeOverflow* (valor supera al indicado por el campo *max*), *stepMismatch* (valor no divisible por el campo *step*), *customError* (error definido con *setCustomValidity*)

Para conocer qué origina el error en el formulario, reemplazaríamos la función enviar por la siguiente versión:

```
function enviar(){
    var elemento = document.getElementById("usuario");
    var valido = formulario.checkValidity();
    if(valido) formulario.submit();
    else
        if(elemento.validity.patternMismatch || elemento.validity.valueMissing)
            alert('el nombre de usuario debe tener mínimo de 3 caracteres');
}
```

Usando pseudoclases

:valid e :invalid

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="utf-8">
    <title>Formularios</title>
    <style>
        input:valid{ background: #EEEEFF; }
        input:invalid{ background: #FFEEEE; }
    </style>
</head>
<body>
    <form name="formulario" method="get" action="procesar.php">
        <input type="email" name="miemail" required>
        <input type="submit" value="Enviar">
    </form>
</body>
</html>
```

:required y :optional

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <style>
    input:optional{ /* es optional cualquiera que no sea required */
      border: 2px solid #009999;
    }
    input:required{
      border: 2px solid #000099;
    }
  </style>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <p><input type="text" name="nombre"></p>
      <p><input type="text" name="apellidos" required></p>
      <p><input type="submit" value="Enviar"></p>
    </form>
  </section>
</body>
</html>
```

:in-range y :out-of-range

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Formularios</title>
  <style>
    input:in-range{
      background: #EEEEFF;
    }
    input:out-of-range{
      background: #FFEEEE;
    }
  </style>
</head>
<body>
  <section>
    <form name="formulario" method="get" action="procesar.php">
      <input type="number" name="numero" min="0" max="10">
      <input type="submit" value="Enviar">
    </form>
  </section>
</body>
</html>
```

willValidate

Se utiliza para saber si un campo del formulario tiene que ser verificado o no. Aunque el valor de un campo sea erróneo (texto en un input numérico, por ejemplo), si el navegador puede verificarlo, devolverá *true*.

```
<div id="uno"></div>
<input type="text" id="dos" />
<input type="text" id="tres" disabled />
<script>
    alert(document.getElementById('uno').willValidate); // muestra undefined
    alert(document.getElementById('dos').willValidate); // muestra true
    alert(document.getElementById('tres').willValidate); // muestra false
</script>
```

Canvas

Sintaxis básica del elemento Canvas

Permite dibujar, presentar gráficos y animaciones, procesar imágenes y texto. Es un elemento de tipo block.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Canvas API</title>
    <script src="canvas.js"></script>
</head>
<body>
    <section id="cajalienzo">
        <canvas id="lienzo" width="500" height="300"> <!-- dims obligatorias: -->
            Su navegador no soporta el elemento canvas <!-- muestra una caja vacía -->
        </canvas>
    </section>
</body>
</html>
```

El archivo canvas.js:

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d'); //crea un contexto de dibujo asociado al canvas
} // '2d': hay otros valores aún no estables: webgl, p.ej.
window.addEventListener("load", iniciar);
```


Dibujo de rectángulos con color

```
function iniciar(){ // origen de coordenadas: esquina superior izquierda
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');

    lienzo.fillStyle="#000099"; // color de relleno
    lienzo.strokeStyle="#990000"; // color de contorno

    lienzo.strokeRect(100,100,120,120); // rectángulo hueco (x, y, ancho, alto)
    lienzo.fillRect(110,110,100,100); // rectángulo sólido
    lienzo.clearRect(120,120,80,80); // borra área rectangular.
}

window.addEventListener("load", iniciar);
```

Los dibujos se solapan en el orden de creación. El color de fondo del canvas es el mismo que el del elemento que lo contiene.

La propiedad *globalAlpha* permite leer o establecer el nivel de transparencia entre 0 (transparente) y 1 (opaco, valor por defecto). En la función anterior, por ejemplo: `lienzo.globalAlpha = 0.2;`

Se pueden usar las funciones *rgb* y *rgba* de CSS para establecer colores:

```
lienzo.fillStyle = "rgba(32, 45, 21, 0.3)"; // r,g,b ∈ [0, 255]. Alpha: 1: opaco
```

El contexto contiene una referencia al canvas al que pertenece. Por ejemplo, para calcular las dimensiones del canvas a partir de contexto:

```
var ancho_canvas = lienzo.canvas.width;
var alto_canvas = lienzo.canvas.height;
```

Aplicando un gradiente lineal al lienzo

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d');

    var grd =lienzo.createLinearGradient(0,0,200,0); // línea (direc) del gradiente
    grd.addColorStop(0.2,"red"); // hasta el 20% sin degradado
    grd.addColorStop(1,"#FFFFFF"); // hasta el 100% blanco (colores tb. hexadecimal)
    // addColorStop indica un punto donde el color es puro
    // se asigna el grad como si fuera un color:
    lienzo.fillStyle = grd;
    lienzo.fillRect(10,10,150,80);
}

window.addEventListener("load", iniciar);
```



addColorStop indica dónde empieza el degradado. Además de *createLinearGradient*, también existe *createRadialGradient*. Se pueden definir degradados con más de dos colores. Para asignar un gradiente a las líneas, se usa *strokeStyle*.

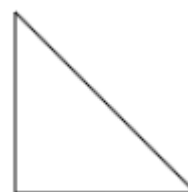
Es importante tener en cuenta que el gradiente se define con respecto a las coordenadas del canvas, por lo que la misma figura en dos lugares distintos puede mostrar gradientes diferentes.

Triángulo con trazados

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100,100); // va al pto sin dibujar el salto. Evita líneas continuas
  lienzo.lineTo(200,200); // línea recta desde pos. actual al pto (x,y) indicado
  lienzo.lineTo(100,200);
  lienzo.fill();           // dibuja el trazado como figura sólida
}
window.addEventListener("load", iniciar);
```

Si en lugar del triángulo sólido anterior se quisiera dibujar su perímetro, se usaría la siguiente versión:

```
function iniciar() {
  var elemento = document.getElementById('lienzo');
  lienzo = elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100, 100);
  lienzo.lineTo(200, 200);
  lienzo.lineTo(100, 200);
  lienzo.closePath(); // línea hasta el primer punto
  lienzo.stroke();
}
window.addEventListener("load", iniciar);
```



Uso del triángulo anterior como una máscara

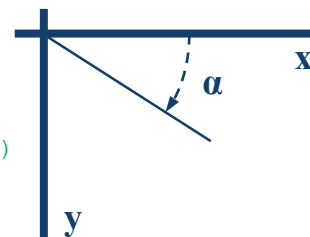
```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.moveTo(100,100);
  lienzo.lineTo(200,200);
  lienzo.lineTo(100,200);
  lienzo.clip();           // define la zona del canvas en la que se podrá dibujar:
  lienzo.beginPath(); // lo que esté fuera se omite. No eliminable salvo
  for(f=0; f<300; f=f+10){ // restaurando el contexto
    lienzo.moveTo(0,f);
    lienzo.lineTo(500,f);
  }
  lienzo.stroke(); // closePath, que traza una línea del último punto al primero
} // resetClip(): elimina las máscaras de recorte
window.addEventListener("load", iniciar);
```



Arcos

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');

  lienzo.beginPath();
  var radianes=Math.PI/180*45;    // 2 * Math.PI: círculo
  // arc: x, y, radio, ang inic, ang fin, false (sentido horario)
  lienzo.arc(100,100,50,0,radianes, false);
  lienzo.stroke();
}
window.addEventListener("load", iniciar);
```



Los ángulos siempre se miden en radianes y por defecto, el signo es el indicado en la figura anterior.

Curvas complejas

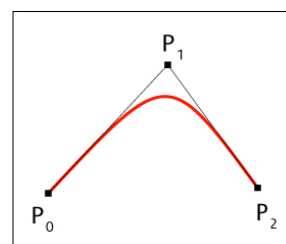
```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');

  lienzo.beginPath();
  lienzo.moveTo(50,50);
  lienzo.quadraticCurveTo(100,125, 50,200); // curva cuadrática de Bezier
  lienzo.moveTo(250,50);
  lienzo.bezierCurveTo(200,125, 300,125, 250,200); // cúbica: el pto del medio es
  lienzo.stroke();                                // de control, no de paso.
}
window.addEventListener("load", iniciar);
```



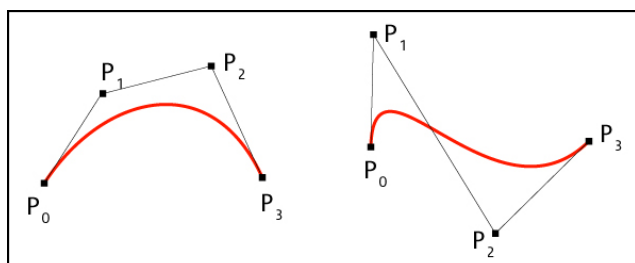
Una curva de Bezier cuadrática: usa tres puntos para definir un polinomio de segundo grado:

$$B(t) = P_0 (1-t)^2 + 2 P_1 t (1-t) + t^2 P_2 \quad \text{con } t \in [0,1]$$



Curva de Bezier cúbica: usa cuatro puntos, dos de ellos de control.

$$B(t) = P_0 (1-t)^3 + 3 P_1 t (1-t)^2 + 3 P_2 t^2 (1-t) + P_3 t^3 \quad \text{con } t \in [0,1]$$

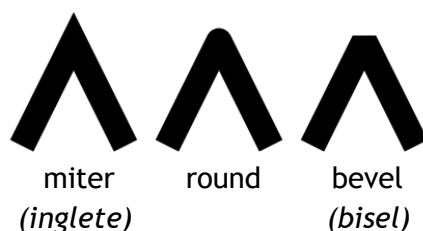


Diferentes estilos de línea

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.beginPath();
  lienzo.arc(200,150,50,0,Math.PI*2, false);
  lienzo.stroke();

  lienzo.lineWidth=10; // por defecto, el grosor de la línea es de 1 punto
  lienzo.lineCap="round"; // forma de terminación de la línea: butt, round, square:
  lienzo.beginPath();
  lienzo.moveTo(230,150);
  lienzo.arc(200,150,30,0,Math.PI, false);
  lienzo.stroke();

  lienzo.lineWidth=5;
  lienzo.lineJoin="miter"; // forma de conectar dos líneas: miter, bevel, round
  lienzo.beginPath(); // existe miterLimit: corte anticipado del miter
  lienzo.moveTo(195,135);
  lienzo.lineTo(215,155);
  lienzo.lineTo(195,155);
  lienzo.stroke();
}
window.addEventListener("load", iniciar);
```



Dibujando texto

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');

  lienzo.font="bold 24px verdana, sans-serif"; // mismos valores que en CSS
  lienzo.textAlign="start"; // start, end, left, center, right
  lienzo.fillText("Mi mensaje", 100,100); // texto sólido; strokeText: contorno.
} // Las coordenadas de fillText son de // la esquina inferior izq. del texto
window.addEventListener("load", iniciar);
```

El alineamiento *start* equivale a *left* en idiomas que se escriben de izquierda a derecha y a *right* en aquellos con escritura de derecha a izquierda. Por el mismo motivo, el valor *end*, equivale a *right* en los primeros y a *left* en los segundos.

En el canvas se puede indicar la dirección del texto:

```
<canvas id="lienzo" width="578" height="200" dir="rtl"></canvas>
```

Midiendo texto

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.font="bold 24px verdana, sans-serif";
  lienzo.textAlign="start";
  lienzo.textBaseline="bottom";           // alphabetic (por defecto), top,
  lienzo.fillText("Mi mensaje", 100,124); // hanging, middle e ideographic
  var tamano=lienzo.measureText("Mi mensaje"); // Devuelve objeto TextMetrics
  lienzo.strokeRect(100,100,tamano.width,24); // ancho del texto
}
window.addEventListener("load", iniciar);
```

Aplicando sombras

Bottom Middle Alphabetic Hanging
Top

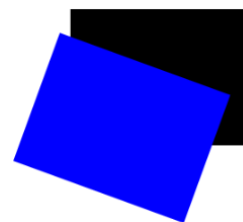
```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.shadowColor="rgba(0,0,0,0.5)"; // usa sintaxis CSS
  lienzo.shadowOffsetX=4;                // desplazamiento a dcha
  lienzo.shadowOffsetY=4;                // desplazamiento hacia abajo
  lienzo.shadowBlur=5;                  // difuminado de las sombras
  lienzo.font="bold 50px verdana, sans-serif";
  lienzo.fillText("Mi mensaje", 100,100);
}
window.addEventListener("load", iniciar);
```

Mi mensaje

Mover, rotar y escalar

Las transformaciones se aplican sobre el sistema de referencia, de modo que lo que está dibujado con anterioridad no se ve afectado.

```
var lienzo = document.getElementById("lienzo");
var lienzo= lienzo.getContext("2d");
lienzo.fillRect(80, 20, 200, 150);
lienzo.rotate(20 * Math.PI / 180); // radianes
lienzo.fillStyle="blue";
lienzo.fillRect(80, 20, 200, 150);
```



Las transformaciones son acumulativas:

```
var elemento=document.getElementById('lienzo');
lienzo=elemento.getContext('2d');
lienzo.font="bold 20px verdana, sans-serif";
lienzo.fillText("PRUEBA",50,20);
lienzo.translate(50,70);           // mueve el origen del sist de ref (tb negativos)
lienzo.rotate(Math.PI/180*45);    // rota los ejes de referencia l lienzo
lienzo.fillText("PRUEBA",0,0);
lienzo.rotate(-Math.PI/180*45);
lienzo.translate(0,100);
lienzo.scale(2,2);                 // amplía las divisiones del sist. de referencia
lienzo.fillText("PRUEBA",0,0);
```

PRUEBA

PRUEBA

PRUEBA

Transformaciones acumulativas sobre la matriz

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.transform(3,0,0,1,0,0); // transformación afín
  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA",20,20);
  lienzo.transform(1,0,0,10,0,0);
  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA",100,20);
}
window.addEventListener("load", iniciar);
```

La función *setTransform* resetea y aplica una nueva matriz (modifica el sistema de referencia).

Transformación afín con *transform (a,b,c,d,e,f)*:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Traslación:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Rotación:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Guardando y restaurando el estado

El estado del lienzo incluye la matriz de transformación actual, la región actual de recorte (clip) y los valores de algunos atributos (*strokeStyle*, *fillStyle*, *globalAlpha*, *lineWidth*, *lineCap*, *lineJoin*, *miterLimit*, *shadowOffsetX*, *shadowOffsetY*, *shadowBlur*, *shadowColor*, *globalCompositeOperation*, *font*, *textAlign* y *textBaseline*). Los métodos *save* y *restore* guardan y recuperan este estado respectivamente. El método *save* no cambia el estado, sólo lo guarda. Estos métodos tampoco modifican el dibujo.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.save(); // guarda el estado del contexto actual
  lienzo.translate(50,70);
  lienzo.font="bold 20px verdana, sans-serif";
  lienzo.fillText("PRUEBA1",0,30);
  lienzo.restore();
  lienzo.fillText("PRUEBA2",0,30);
}
window.addEventListener("load", iniciar);
```

Dado que las transformaciones afectan al sistema de coordenadas, los métodos *save* y *restore* son útiles si se quiere borrar todo el canvas:

```

contexto.save(); // se guarda el estado
contexto.setTransform(1, 0, 0, 1, 0, 0); // se elimina todas las transformaciones
contexto.clearRect(0, 0, canvas.width, canvas.height); // se borra todo
contexto.restore(); // se restaura el estado: transformaciones, area de recorte...

```

Propiedad globalCompositeOperation

Esta propiedad define como se posiciona una figura sobre las anteriores.

```

function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  lienzo.fillStyle="#990000";
  lienzo.fillRect(100,100,300,100);

  // sólo muestra la parte de la imagen previa que se solapa sobre la nueva:
  lienzo.globalCompositeOperation="destination-atop";
  lienzo.fillStyle="#AAAAFF";
  lienzo.font="bold 80px verdana, sans-serif";
  lienzo.textAlign="center";
  lienzo.textBaseline="middle";
  lienzo.fillText("PRUEBA",250,110);
}
window.addEventListener("load", iniciar);

```



Otros valores de la propiedad son: *source-over*, *source-atop*, *source-in*, *source-out*, *destination-over*, *destination-atop*, *destination-in*, *destination-out*, *lighter*, *copy* y *xor*.

Trabajando con imágenes

Mostrar una imagen en el Canvas

```

function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  var imagen=new Image(); // tb se puede cargar imagen existente con getElementById
  imagen.src="http://momentum.uvigo.es/multimedia/snow.jpg"; // carga, no muestra
  // cuando la imagen está totalmente cargada, se dispara el evento load: mostrar
  imagen.addEventListener("load",function(){lienzo.drawImage(imagen,20,20)});
}
window.addEventListener("load", iniciar);

```

El método *drawImage* es el único método que existe para dibujar imágenes, canvas o vídeo . Puede tener hasta nueve argumentos, todos opcionales excepto el primero (la imagen a mostrar). En los siguientes ejemplos se verá cómo usar esta función para recortar y ampliar o reducir la imagen mostrada.

Argumentos de *drawImage(img, a, b, c, d, e, f, g, h)*:

- a, b: inicio del recorte (opcionales, posición superior izquierda)
- c, d: ancho y alto del recorte: opcionales
- e, f: posición de la imagen
- g, h: ancho y alto de la imagen (opcionales, permiten ampliar o reducir la imagen)

Cuando se pasan cinco argumentos, los dos últimos se referirán al tamaño de la imagen.

Ajustar la imagen al tamaño del Canvas

```
lienzo.drawImage(imagen, 0, 0, elemento.width, elemento.height)
```

Extraer, cambiar tamaño y dibujar

En este ejemplo se mostrará una imagen desde el origen (0,0), con unas dimensiones de 200x200 píxeles. La imagen mostrada es un recorte de 50x50 px de la original, comenzando en el píxel (135,50):

```
lienzo.drawImage(imagen,135,30,50,50,0,0,200,200)
```

Procesando imágenes

En este ejemplo se accederá a un trozo de una imagen y se mostrará en negativo.

```
function iniciar(){
    var elemento=document.getElementById('lienzo');
    lienzo=elemento.getContext('2d'); // variable global (para mayor velocidad)
    var imagen=new Image();
    imagen.crossOrigin = "anonymous"; // Cross-origin: permite imgs de otros sitios
    imagen.src="snow.jpg";
    imagen.addEventListener("load", modificarimagen);
}
function modificarimagen(e){
    var imagen=e.target; // el evento contiene una ref a la fuente
    lienzo.drawImage(imagen,0,0); // dibuja la imagen original
    var info=lienzo.getImageData(0,0,175,262); // extrae una parte del canvas
    var pos; // correspondiente a la mitad
    for(x=0;x<=175;x++){ // izquierda de la imagen
        for(y=0;y<=262;y++){
            pos=(info.width*4*y)+(x*4); // imagen como array de 4 pos por pixel:
            info.data[pos]=255-info.data[pos]; // (r, g, b, a). Guarda por filas
            info.data[pos+1]=255-info.data[pos+1]; // cada pos. toma valores en [0, 255]
            info.data[pos+2]=255-info.data[pos+2]; // resta de 255 para efecto negativo
        }
    }
    lienzo.putImageData(info,0,0); // dibuja una imagen guardada en un objeto
    // como el info anterior
}
window.addEventListener("load", iniciar);
```

El método *getImageData* extrae un trozo rectangular del canvas y lo devuelve en un objeto con tres campos: width, height y data. Los argumentos del método son: posición x, posición y, ancho y alto.

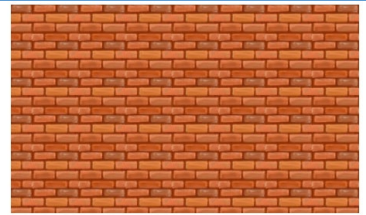
Por razones de seguridad, no es posible extraer información del canvas tras dibujar en el una imagen tomada desde una fuente externa. Este método sólo funcionará adecuadamente cuando el documento y la imagen proceden de la misma URL (en local tampoco funciona).

El método *putImageData* dibuja una imagen guardada en un objeto con la misma estructura que el *info* del ejemplo. Los otros dos parámetros indican la posición de la imagen a dibujar (x, y). El método *createImage* crea una imagen con todos los píxeles a cero (negro transparente). El método *toDataURL(tipo)* devuelve una imagen con el formato indicado (por defecto, PNG). Permite asignar el contenido de la imagen en el campo dirección.

Patrones (texturas)

El funcionamiento de los patrones es similar al de los gradientes.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  var imagen=new Image();
  imagen.src="http://momentum.uvigo.es/multimedia/bricks.jpg";
  imagen.addEventListener("load", modificarimagen);
}
function modificarimagen(e){
  var imagen=e.target;
  var patron=lienzo.createPattern(imagen, 'repeat'); // crea patrón
  lienzo.fillStyle=patron;                          // asigna el patrón
  lienzo.fillRect(0,0,500,300);
}
window.addEventListener("load", iniciar);
```



El segundo argumento del método *createPattern* permite controlar la repetición mediante los valores *repeat-x*, *repeat-y*, *repeat* y *no-repeat*.

Animación básica

El siguiente ejemplo dibujará dos ojos mirando hacia el puntero del ratón.

```
function iniciar(){
  var elemento=document.getElementById('lienzo');
  lienzo=elemento.getContext('2d');
  window.addEventListener('mousemove', animacion, false);
}
function animacion(e){
  lienzo.clearRect(0,0,300,500); // limpia el lienzo
  var xraton=e.clientX;           // obtiene la posición del puntero
  var yraton=e.clientY;
  var xcentro=220;                // establece centro del ojo izquierdo
  var ycentro=150;
  // cálculos pupila
  var angulo=Math.atan2(yraton-ycentro, xraton-xcentro); // ángulo puntero-ojo
  var x=xcentro+Math.round(Math.cos(angulo)*10); // centro de la pupila
  var y=ycentro+Math.round(Math.sin(angulo)*10); //10: dist centros ojo y pupila
  // dibuja ojos con radio = 50
  lienzo.beginPath();
  lienzo.arc(xcentro,ycentro,20,0,Math.PI*2, false); // primer ojo
  lienzo.moveTo(xcentro+70,150);
  lienzo.arc(xcentro+50,150,20,0,Math.PI*2, false); // segundo ojo
  lienzo.stroke();
  // dibuja pupilas
  lienzo.beginPath();
  lienzo.moveTo(x+10,y);
  lienzo.arc(x,y,10,0,Math.PI*2, false); // dibuja primera pupila
  lienzo.moveTo(x+60,y);
  lienzo.arc(x+50,y,10,0,Math.PI*2, false); // dibuja segunda pupila
  lienzo.fill(); // rellena en negro el path actual. fillStyle=color/gradiente
}
window.addEventListener("load", iniciar);
```



Si hay que redibujar las mismas cosas muy a menudo, en general suele ser más rápido y trabajar con imágenes.

Mostrar vídeo en el Canvas

El siguiente ejemplo muestra el vídeo y su imagen especular, esta última sobre un canvas. Para ello utiliza la función *drawImage*.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="utf-8">
  <title>Video sobre Canvas</title>
  <style>
    section {
      float: left;
    }
  </style>
  <script>
    var canvas, video;
    function inicializar() {
      var elemento = document.getElementById("canvas");
      canvas = elemento.getContext("2d");
      video = document.getElementById("medio");

      canvas.translate(483, 0);    // desplaza a la derecha en el ancho del vídeo
      canvas.scale(-1, 1);        // invierte las coordenadas x
      setInterval(procesaMarcos, 33);
    }
    function procesaMarcos() {
      canvas.drawImage(video, 0, 0); // muestra la instantánea del vídeo
    }
    window.addEventListener("load", inicializar);
  </script>
</head>
<body>
  <section>
    <video id="medio" width="483" height="272" autoplay>
      <source src="trailer.mp4">
      <source src="trailer.ogg">
    </video>
  </section>
  <section>
    <canvas id="canvas" width="483" height="272"></canvas>
  </section>
</body>
</html>
```

Ejemplo básico de arrastrar y soltar

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <title>Arrastrar y soltar</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="dragdrop.css">
    <script src="dragdrop.js"></script>
  </head>
  <body>
    <section id="cajasoltar">
      Arrastre y suelte la imagen aquí
    </section>
    <section id="cajaimagenes">
      
    </section>
  </body>
</html>
```

El fichero dragdrop.css

```
#cajasoltar{
  float: left;
  width: 500px;
  height: 300px;
  margin: 10px;
  border: 1px solid #999999;
}
#cajaimagenes{
  float: left;
  width: 320px;
  margin: 10px;
  border: 1px solid #999999;
}
#cajaimagenes > img{
  float: left;
  padding: 5px;
}
```

El fichero dragdrop.js

```
function iniciar(){
  var origen1=document.getElementById('imagen');
  origen1.addEventListener('dragstart', arrastrado);
  var destino=document.getElementById('cajasoltar');
  destino.addEventListener('dragenter',function(e){ e.preventDefault(); });
  destino.addEventListener('dragover', function(e){ e.preventDefault(); });
  destino.addEventListener('drop', soltado);
}
function arrastrado(e){
  var codigo='';
  e.dataTransfer.setData('Text', codigo);
}
```

```
function soltado(e){
    e.preventDefault(); // a veces, al arrastrar un enlace, por defecto lo abre
    e.target.innerHTML=e.dataTransfer.getData('Text');
}
window.addEventListener('load', iniciar);
```

Los eventos del origen son: *dragstart*, *drag* y *dragend*. Los del destino (o destino potencial): *dragenter*, *dragover*, *drop* y *dragleave* (*drag* y *dragover* se disparan varias veces por segundo)

El método *preventDefault* impide que se ejecute la acción normal del navegador ante ese evento. Si por ejemplo se aplica en una respuesta al evento *click* en un enlace, impediría que se mostrara la página correspondiente a dicho enlace.

El objeto *dataTransfer* está compartido por los eventos de origen y destino y sus métodos son *getData(tipo)*, *clearData(tipo)* y *setData (tipo, dato)*. El tipo puede ser *text/plain*, *text/html*, *URL* y *text* (algunos navegadores no los soportan todos). El último (*text*) es el más compatible.

Eventos del destino *dragenter*, *dragleave* y *dragend*

```
function iniciar(){
    var origen1=document.getElementById('imagen');
    origen1.addEventListener('dragstart', arrastrado);
    origen1.addEventListener('dragend', finalizado);
    var soltar=document.getElementById('cajasoltar');
    soltar.addEventListener('dragenter', entrando);
    soltar.addEventListener('dragleave', saliendo);
    soltar.addEventListener('dragover', function(e){ e.preventDefault(); });
    soltar.addEventListener('drop', soltado);
}
function arrastrado(e){ // dragstart (origen)
    var codigo='';
    e.dataTransfer.setData('Text', codigo);
}
function finalizado(e){ // dragend (origen)
    elemento=e.target;
    elemento.style.visibility='hidden';
}
function entrando(e){ // dragenter (destino)
    e.preventDefault(); // cambia del fondo del los elementos por los que pasa:
    e.target.style.background='rgba(0,150,0,.2)';
}
function saliendo(e){ // dragleave (destino)
    e.preventDefault(); // cambia del fondo del los elementos por los que pasa
    e.target.style.background='#FFFFFF';
}
function soltado(e){ // drop (destino)
    e.preventDefault();
    e.target.style.background='#FFFFFF';
    e.target.innerHTML=e.dataTransfer.getData('Text');
}
window.addEventListener('load', iniciar);
```

Enviando el valor de atributo *id*

Mover cualquiera de las cuatro imágenes, excepto la cuarta.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Arrastrar y soltar</title>
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
    Arrastre y suelte las imágenes aquí
  </section>
  <section id="cajaimagenes">
    
    
    
    
  </section>
</body>
</html>
```

El fichero dragdrop.js

```
function iniciar(){
  var imagenes=document.querySelectorAll('#cajaimagenes > img');
  for(var i=0; i<imagenes.length; i++){
    imagenes[i].addEventListener('dragstart', arrastrado);
  }

  soltar=document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){ e.preventDefault(); });
  soltar.addEventListener('dragover', function(e){ e.preventDefault(); });
  soltar.addEventListener('drop', soltado);
}

function arrastrado(e){          // dragstart (origen)
  elemento=e.target;
  e.dataTransfer.setData('Text', elemento.getAttribute('id'));
}

function soltado(e){             // drop (destino)
  e.preventDefault();
  // lee id de la imag transmitida, lee origen y dibuja
  var id=e.dataTransfer.getData('Text');
  if(id!="imagen4"){ // no se permite arrastrar la imagen4
    var src=document.getElementById(id).src;
    soltar.innerHTML='';
  }
  else{
    soltar.innerHTML='la imagen no es admitida';
  }
}
window.addEventListener('load', iniciar);
```

El método setdragimage

En este ejemplo, se mostrará una imagen durante el desplazamiento

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>Arrastrar y soltar</title>
    <link rel="stylesheet" href="dragdrop.css">
    <script src="dragdrop.js">
    <meta charset="UTF-8">
</head>
<body>
    <section id="cajasoltar"><canvas id="lienzo" width="500" height="300"></canvas>
</section>
<section id="cajaimagenes">
    
    
    
    
</section>
</body>
</html>
```

El fichero dragdrop.js

```
function iniciar(){
    var imagenes=document.querySelectorAll('#cajaimagenes > img');
    for(var i=0; i<imagenes.length; i++){
        imagenes[i].addEventListener('dragstart', arrastrado);           // origen
        imagenes[i].addEventListener('dragend', finalizado);           // origen
    }
    soltar=document.getElementById('lienzo');
    lienzo=soltar.getContext('2d');
    imgArrastre = new Image();
    imgArrastre.src = "http://momentum.uvigo.es/multimedia/Positive.png";
    soltar.addEventListener('dragenter',function(e){ e.preventDefault(); });
    soltar.addEventListener('dragover', function(e){ e.preventDefault(); });
    soltar.addEventListener('drop', soltado);
}
function arrastrado(e){ // (dragstart en el origen)
    elemento=e.target;
    e.dataTransfer.setData('Text', elemento.getAttribute('id'));
    e.dataTransfer.setDragImage(imgArrastre, 0, 0); // pos img respecto al puntero
}
function finalizado(e){ // (dragend en el origen) para ocultar la imagen original
    elemento=e.target;
    elemento.style.visibility='hidden';
}
function soltado(e){ // (drop en el destino)
    e.preventDefault();
    var id=e.dataTransfer.getData('Text');
    var elemento=document.getElementById(id);
    var posx=e.pageX-soltar.offsetLeft; // pageX y pageY: posición del cursor
    var posy=e.pageY-soltar.offsetTop; // respecto al origen del documento
    lienzo.drawImage(elemento,posx,posy); // offsetLeft y offsetTop: desplazamiento
    // respecto a su elemento padre
}
window.addEventListener('load', iniciar);
```

Procesando los datos con la propiedad *files*

En el siguiente ejemplo se verá cómo arrastrar archivos desde el sistema operativo u otras aplicaciones. Por seguridad, no es posible arrastrar archivos fuera del navegador.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title> Arrastrar y soltar </title>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="dragdrop.css">
  <script src="dragdrop.js"></script>
</head>
<body>
  <section id="cajasoltar">
    Arrastre y suelte archivos en este espacio
  </section>
</body>
</html>
```

El fichero dragdrop.js

```
var soltar;

function iniciar(){
  soltar=document.getElementById('cajasoltar');
  soltar.addEventListener('dragenter', function(e){ e.preventDefault(); });
  soltar.addEventListener('dragover' , function(e){ e.preventDefault(); });
  soltar.addEventListener('drop', soltado);
}

function soltado(e){
  e.preventDefault();
  var archivos=e.dataTransfer.files; // devuelve objeto FileList (colección)
  var lista="";
  for(var f=0;f<archivos.length;f++){ // indexando FileList devuelve obj. File
    lista+='Archivo: '+archivos[f].name+' '+archivos[f].size+'<br>';
  }
  soltar.innerHTML=lista;
}
window.addEventListener('load', iniciar);
```

El objeto *FileList* también se usa para gestionar la lista de ficheros obtenidos con la etiqueta HTML `<input type="file" multiple>`. El objeto *File* obtenido de la indexación, se puede usar para leer su contenido, como se verá más adelante.

Se puede comprobar la disponibilidad de la geolocalización con: `if(navigator.geolocation)`

Obteniendo la localización del usuario

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Geolocalización</title>
  <script src="geolocation.js"></script>
</head>
<body>
  <section id="ubicacion">
    <button id="obtener">Obtener mi Ubicación</button>
  </section>
</body>
</html>
```

El fichero geolocation.js

```
function iniciar(){
  var boton=document.getElementById('obtener');
  boton.addEventListener('click', obtener);
}
function obtener(){
  navigator.geolocation.getCurrentPosition(mostrar); // se le pasa la función que
                                                    // procesará la info
}
function mostrar(posicion){
  var ubicacion=document.getElementById('ubicacion');
  var datos = 'Latitud: ' + posicion.coords.latitude+'<br>';
  datos += 'Longitud: ' + posicion.coords.longitude+'<br>';
  datos += 'Exactitud: ' + posicion.coords.accuracy+' m.<br>';
  ubicacion.innerHTML = datos;
}
window.addEventListener('load', iniciar);
```

La propiedad *coords* dispone de los siguientes campos: *latitude* (número decimal), *longitude* (número decimal), *altitude* (en metros), *heading* (en grados en sentido horario desde el norte), *speed* (en m/s), *accuracy* (en metros), *altitudeAccuracy* y *timestamp* (del tipo *DOMTimeStamp*, que se puede convertir a un objeto *Date*: `new Date(domTimeStamp)`)

Mensajes de error

```
function iniciar(){
  var boton=document.getElementById('obtener');
  boton.addEventListener('click', obtener);
}
function obtener(){
  navigator.geolocation.getCurrentPosition(mostrar, errores);
}
```



```
function mostrar(posicion){
    var ubicacion=document.getElementById('ubicacion');
    var datos = 'Latitud: ' + posicion.coords.latitude+'<br>';
    datos += 'Longitud: ' + posicion.coords.longitude+'<br>';
    datos += 'Exactitud: ' + posicion.coords.accuracy+' m.<br>';
    ubicacion.innerHTML=datos;
}
function errores(error){
    alert('Error: ' + error.code + ' ' + error.message);
}
window.addEventListener('load', iniciar);
```

Los valores posibles del error son: PERMISSION_DENIED, POSITION_UNAVAILABLE, TIMEOUT y UNKNOWN_ERROR.

Opciones

```
function iniciar(){
    var boton=document.getElementById('obtener');
    boton.addEventListener('click', obtener);
}
function obtener(){
    var geoconfig={ enableHighAccuracy: true,      // GPS
                    timeout: 10000,                // ms
                    maximumAge: 60000               // ms. Si supera pide nueva pos.
    };
    navigator.geolocation.getCurrentPosition(mostrar, errores, geoconfig);
}
function mostrar(posicion){
    var ubicacion=document.getElementById('ubicacion');
    var datos = "";
    var datos = 'Latitud: ' + posicion.coords.latitude+'<br>';
    datos += 'Longitud: ' + posicion.coords.longitude+'<br>';
    datos += 'Exactitud: ' + posicion.coords.accuracy+' m.<br>';
    ubicacion.innerHTML=datos;
}
function errores(error){ alert('Error: ' + error.code + ' ' + error.message); }
window.addEventListener('load', iniciar);
```

El dispositivo puede devolver un valor guardado en su memoria caché (en lugar de calcular una nueva ubicación), siempre que dicho valor haya sido solicitado en un plazo inferior a los milisegundos indicados por el atributo *maximumAge*.

El método watchposition

Devuelve de forma periódica los datos a medida que el dispositivo se mueve. Si no hay cambio, no se obtiene una nueva lectura.

```
function iniciar(){
    var boton=document.getElementById('obtener');
    boton.addEventListener('click', obtener);
}
```

```

function obtener(){
    var geoconfig={
        enableHighAccuracy: true,
        maximumAge: 60000 // período de muestreo en milisegundos
    };
    control=navigator.geolocation.watchPosition(mostrar, errores, geoconfig);
}
function errores(error){
    alert('Error: '+error.code+' '+error.message);
}
function mostrar(posicion){
    var ubicacion=document.getElementById('ubicacion');
    var datos = "";
    var datos = 'Latitud: ' + posicion.coords.latitude+'<br>';
    datos += 'Longitud: ' + posicion.coords.longitude+'<br>';
    datos += 'Exactitud: ' + posicion.coords.accuracy+' m.<br>';
    ubicacion.innerHTML = datos;
}
window.addEventListener('load', iniciar);

```

El método *clearWatch* detiene el funcionamiento de *watchPosition*. El atributo *maximumAge* contiene el período de muestreo de la posición.

Mostrando la ubicación en google maps

Se puede mostrar una imagen estática de la ubicación usando el servicio de Google Maps. Para ello es imprescindible obtener previamente una clave.

```

var claveAPI = "xyz"; // clave para el acceso a los mapas de google
function iniciar(){
    var boton=document.getElementById('obtener');
    boton.addEventListener('click', obtener);
}
function obtener(){
    navigator.geolocation.getCurrentPosition(mostrar, errores);
}
function mostrar(posicion){
    var ubicacion=document.getElementById('ubicacion');
    var mapurl = "https://maps.googleapis.com/maps/api/staticmap?center=" +
        posicion.coords.latitude + ',' + posicion.coords.longitude +
        "&zoom=14&size=400x300&sensor=false&key=" + claveAPI;
}
function errores(error){
    alert('Error: '+error.code+' '+error.message);
}
window.addEventListener('load', iniciar);

```

El objeto *sessionStorage* mantiene la información (pares clave/valor de **texto**) por pestaña o ventana y el *localStorage* almacena la información de modo permanente. En algunos navegadores los siguientes ejemplos no funcionan si no se ejecutan desde un servidor real. Para ver si el navegador soporta el almacenamiento web:

```
if (typeof(Storage) !== "undefined") { ... }
```

Almacenando y leyendo datos con *sessionStorage*

La información guardada con *sessionStorage* se mantiene aunque se recargue la página. Sin embargo, a diferencia de las *cookies*, si se cierra la pestaña o ventana y se vuelve a abrir de nuevo la página, la información se habrá borrado.

```
<!DOCTYPE html>
<html lang="es">
<head>
<title>Web Storage API</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="storage.css">
  <script src="storage.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p>Clave:<br><input type="text" name="clave" id="clave"></p>
      <p>Valor:<br><textarea name="text" id="texto"></textarea></p>
      <p><button type="button" id="grabar">Grabar</button></p>
    </form>
  </section>
  <section id="cajadatos"> No hay información disponible </section>
</body>
</html>
```

El fichero *storage.css*

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto{ width: 200px; }
#cajadatos > div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

El fichero storage.js

```
function iniciar(){
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem); // 3er argumento: false por defecto
}

function nuevoitem(){
    var clave=document.getElementById('clave').value;
    var valor=document.getElementById('texto').value;
    sessionStorage.setItem(clave,valor); // tb vale: sessionStorage[clave]=valor;
    mostrar(clave);
}

function mostrar(clave){
    var cajadatos=document.getElementById('cajadatos');
    var valor=sessionStorage.getItem(clave); // tb: var valor=sessionStorage[clave];
    cajadatos.innerHTML = '<div>' + clave + ' - ' + valor + '</div>';
}

window.addEventListener('load', iniciar);
```

Es posible usar la notación `sessionStorage.clave`, para leer y escribir, sin necesidad de usar `setItem` ni `getItem`. Por ejemplo: `sessionStorage.clave = valor`;

Listar y eliminar datos

```
function iniciar(){
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem, false);
    mostrar();
}

function nuevoitem(){
    var clave=document.getElementById('clave').value;
    var valor=document.getElementById('texto').value;
    sessionStorage.setItem(clave,valor);
    document.getElementById('clave').value = "";
    document.getElementById('texto').value = "";
    mostrar();
}

function mostrar(){
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML='<div><button onclick="eliminarTodo()">' +
        'Eliminar Todo</button></div>';

    for(var f=0;f<sessionStorage.length;f++){ // listar recorriendo todas las claves
        var clave=sessionStorage.key(f);
        var valor=sessionStorage.getItem(clave);
        cajadatos.innerHTML+='<div>' + clave + ' - ' + valor +
            '<br><button onclick="eliminar(\'' + clave + '\')">Eliminar</button></div>';
    }
}

function eliminar(clave){
    if(confirm('¿Está Seguro?')){
        sessionStorage.removeItem(clave); // eliminar par clave/valor
        mostrar();
    }
}
```

```
function eliminarTodo(){
    if(confirm('¿Está Seguro?')){
        sessionStorage.clear();    // eliminar todos los pares clave/valor
        mostrar();
    }
}
window.addEventListener('load', iniciar);
```

Uso de localStorage

Funciona de modo similar a *sessionStorage*, pero los datos son permanentes (sin fecha de caducidad). La reserva típica de espacio es de 5 MB o más. A diferencia de las *cookies*, su información **no se envía al servidor**. El acceso se permite únicamente a las páginas del mismo subdominio.

```
function iniciar(){
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem);
    mostrar();
}
function nuevoitem(){
    var clave=document.getElementById('clave').value;
    var valor=document.getElementById('texto').value;
    localStorage.setItem(clave,valor);    // único cambio con respecto a sessionStorage
    mostrar();
    document.getElementById('clave').value = "";
    document.getElementById('texto').value = "";
}
function mostrar(){
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML = "";
    for(var f=0;f<localStorage.length;f++){
        var clave=localStorage.key(f);
        var valor=localStorage.getItem(clave);
        cajadatos.innerHTML+=<div>'+clave+' - '+valor+'</div>';
    }
}
window.addEventListener('load', iniciar);
```

Evento storage

Se dispara en cada cambio en el espacio de almacenamiento (tanto *localStorage* como *sessionStorage*), independientemente de cuál sea la ventana o pestaña que haya producido el cambio.

```
function iniciar(){
    var boton=document.getElementById('grabar');
    boton.addEventListener('click', nuevoitem);
    window.addEventListener("storage", mostrar);
    mostrar();
}
function nuevoitem(){
    var clave=document.getElementById('clave').value;
    var valor=document.getElementById('texto').value;
    localStorage.setItem(clave,valor);
    mostrar();
    document.getElementById('clave').value = "";
    document.getElementById('texto').value = "";
}
```

```
function mostrar(){
  var cajadatos=document.getElementById('cajadatos');
  cajadatos.innerHTML = "";
  for(var f=0;f<localStorage.length;f++){
    var clave=localStorage.key(f);
    var valor=localStorage.getItem(clave);
    cajadatos.innerHTML += '<div>' + clave + ' - ' + valor + '</div>';
  }
}
window.addEventListener('load', iniciar);
```

IndexedDB

Proporciona una estructura muy básica, aunque con espacio ilimitado, con el objetivo de construir sobre ella bibliotecas de funciones a medida. No hay posibilidad de establecer permisos diferentes para usuario ni restricciones de acceso. Sólo se puede acceder a la base desde el mismo dominio (subdominio) que la creó. Sólo es necesario el nombre y la versión. El elemento básico es el *objectStore* que cumple la misma función que las tablas en las bases de datos relacionales.

Para ver si el navegador soporta esta funcionalidad:

```
if (!window.indexedDB) window.alert("Su navegador no soporta indexedDB");
```

Abrir una base de datos

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>IndexedDB API</title>
  <meta charset="UTF-8">
  <link rel="stylesheet" href="indexed.css">
  <script src="indexed.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="clave">Clave:</label><br>
      <input type="text" name="clave" id="clave"><br>
      <label for="texto"> Título: </label><br>
      <input type="text" name="texto" id="texto"><br>
      <label for="fecha"> Año:</label><br>
      <input type="text" name="fecha" id="fecha"><br>
      <input type="button" name="grabar" id="grabar" value="Grabar"><br>
    </form>
  </section>
  <section id="cajadatos"> No hay información disponible </section>
</body>
</html>
```

El archivo indexed.css

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 400px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
#clave, #texto{
  width: 200px;
}
#cajadatos > div{
  padding: 5px;
  border-bottom: 1px solid #999999;
}
```

El archivo indexed.js

```
var cajadatos, bd;
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  var boton=document.getElementById('grabar');
  boton.addEventListener('click', agregarobjeto);

  window.indexedDB =      window.indexedDB || window.mozIndexedDB ||
                           window.webkitIndexedDB || window.msIndex
  window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||
                           window.msIDBTransaction;
  window.IDBKeyRange =    window.IDBKeyRange || window.webkitIDBKeyRange ||
                           window.msIDBKeyRange;

  var solicitud = window.indexedDB.open('mibase', 1); // abre y si no existe, crea
  solicitud.addEventListener('upgradeneeded', creabd);
  solicitud.addEventListener('error', errores);
  solicitud.addEventListener('success', abrir);
}
```

La versión de la base de datos siempre debe ser un valor entero (por defecto, cero). Cuando en el método *open* se pasa un número de versión superior al actual de la base de datos, se dispara el evento *upgradeneeded*. El mismo evento se dispara la primera vez que se solicita la creación de la base de datos. En un cambio de versión no se deben crear los *objectStores* previos que sigan siendo válidos. Se puede eliminar o crear *objectStores*, pero no modificar. Solo existe una versión de la base de datos en cada momento.

Creación del esquema

```
function errores(e){ alert('Error: ' + e.code + " " + e.message); }
function abrir(e) {
    bd = e.target.result; // bd: conector de la BDD
}
function creabd (e) {
    var base = e.target.result;
    // es en esta función donde se deben crear los objectStores nuevos y eliminar
    // los que ya no hagan falta para la nueva versión:
    var almacen=base.createObjectStore('peliculas',{keyPath:'id'}); //id: clave prim.
    // índice búsqueda. 1º argumento: nombre del índice y campo sobre el que se crea
    almacen.createIndex('BuscarFecha', 'fecha',{unique: false});
}
```

Si se define un campo del *objectStore* como *keyPath*, además de establecer ese campo como clave primaria, podrá contener objetos JavaScript. Esos objetos deben tener una propiedad con el mismo valor que el campo *keyPath* del *objectStore*. Si no existe ningún campo *keyPath* en el *objectStore*, sólo podrá contener valores primitivos, como números o strings. Se puede añadir también un campo *autoincrement* con valores *true* o *false* (en este caso no se indica la clave primaria cuando se inserta un nuevo objeto)

Sólo se pueden realizar búsquedas sobre índices o sobre el *keyPath* (sobre el cual se crea un índice por defecto)

Agregar objetos

Para realizar cualquier operación sobre un *objectStore*, es necesario crear una transacción.

```
function agregarobjeto(){
    var clave = document.getElementById('clave').value;
    var titulo = document.getElementById('texto').value;
    var fecha = document.getElementById('fecha').value;

    var transaccion=bd.transaction(['peliculas'], "readwrite"); // crea la transac.
    transaccion.addEventListener('complete', function(){ mostrar(clave) });
    transaccion.addEventListener('error', function(){ alert("Error") });

    var almacen=transaccion.objectStore('peliculas'); // devuelve objectStore a usar
    almacen.add({id: clave, nombre: titulo, fecha: fecha}); // add: inserta
                                                            // put: inserta o actual.

    document.getElementById('clave').value = "";
    document.getElementById('texto').value = "";
    document.getElementById('fecha').value = "";
}
```

La transacción (conjunto atómico de operaciones) se solicita indicando todas las *objectStore* que va a abarcar (*array*) y posteriormente se van haciendo todas las operaciones. En función de las operaciones que se realizarán en la transacción, ésta debe solicitarse indicando un modo de acceso: *readonly* (valor defecto), *readwrite* o *versionchange* (para cambios en el esquema). Este último, es el único modo de borrar y añadir *objectStores* e índices.

Pueden coexistir varias transacciones si los ámbitos (*objectStores* que usan) no se solapan. Incluso en este caso, pueden coexistir múltiples transacciones de tipo *readonly*, aunque sólo una *readwrite*.

Las transacciones hacen *autocommit* cuando termina la operación solicitada (*add* o *put*, por ejemplo). Esto ocurre cuando termina la ejecución de la función de respuesta a los eventos de éxito o fracaso. Para abortar (hacer un *rollback*) la transacción: `transaccion.abort()`.

Se recomienda que sean lo más rápidas posible.

Leer y mostrar un objeto almacenado (sin índices)

```
function mostrar(clave){
    var transaccion = bd.transaction(['peliculas']); // por defecto modo readonly
    var almacen = transaccion.objectStore('peliculas');
    var solicitud = almacen.get(clave); // clave o rango (IDBKeyRange)
    solicitud.addEventListener('success', mostrarlista);
}
function mostrarlista(e){
    var resultado = e.target.result;
    cajadatos.innerHTML = '<div>' + resultado.id + ' - ' + resultado.nombre +
                        ' - ' + resultado.fecha + '</div>';
}
```

Iniciar la aplicación

```
window.addEventListener('load', iniciar);
```

Cursores

Las funciones *iniciar*, *errores*, *abrir*, *creabd* y *agregarobjeto* no cambian. Los cursores permiten acceder a todos los elementos sin conocer sus claves.

```
function mostrar(){
    cajadatos.innerHTML = "";
    var transaccion = bd.transaction(['peliculas']);
    var almacen = transaccion.objectStore('peliculas');
    var cursor = almacen.openCursor();
    cursor.addEventListener('success', mostrarlista);
}

function mostrarlista(e){
    var cursor = e.target.result; // peculiaridad: el resultado es el cursor
    if(cursor){ // cuando no quedan más datos, vale undefined
        cajadatos.innerHTML += '<div>' + cursor.value.id + ' - ' + cursor.value.nombre
                            + ' - ' + cursor.value.fecha + '</div>';
        cursor.continue();
    }
}

window.addEventListener('load', iniciar);
```

El método *continue* dispara el evento *success* incluso cuando ya no hay elementos que recorrer.

El método *openCursor* se invoca sobre *objectStores* o índices.

Índices y orden de recorrido

```
function mostrar(){
    cajadatos.innerHTML = "";
    var transaccion = bd.transaction(['peliculas']);
    var almacen = transaccion.objectStore('peliculas');

    var indice = almacen.index('BuscarFecha'); // campo de búsqueda
    var cursor = indice.openCursor(null, IDBCursor.PREV); // 2º arg: tb. vale "prev"

    cursor.addEventListener('success', mostrarlista);
}
```

El primer campo del *openCursor* permite introducir el rango del recorrido por medio de un objeto *IDBKeyRange*. El segundo indica el orden del recorrido:

- NEXT: recorrido creciente, que incluye registros duplicados.
- PREV: recorrido decreciente, que incluye registros duplicados.
- NEXTUNIQUE: recorrido creciente, sin registros duplicados (solo el primero encontrado).
- PREVUNIQUE: recorrido decreciente, sin registros duplicados (solo el primero encontrado).

Eliminación de objetos

```
function mostrarlista(e){
    var cursor = e.target.result;
    if(cursor){
        cajadatos.innerHTML += '<div>' + cursor.value.id + ' - ' +
            cursor.value.nombre + ' - ' + cursor.value.fecha +
            ' <button onclick="eliminar(' + cursor.value.id + ')">' +
            'Eliminar</button></div>';
        cursor.continue();
    }
}

function eliminar(clave){
    if(confirm('¿Está Seguro?')){
        var transaccion = bd.transaction(['peliculas'], "readwrite");
        var almacen = transaccion.objectStore('peliculas');
        var solicitud = almacen.delete(clave);
        solicitud.addEventListener('success', mostrar);
    }
}
```

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>IndexedDB API</title>
  <link rel="stylesheet" href="indexed.css">
  <script src="indexed.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="fecha"> Buscar Película por Año: </label><br>
      <input type="search" name="fecha" id="fecha"></p>
      <p><input type="button" name="buscar" id="buscar" value="Buscar"></p>
    </form>

  </section>
  <section id="cajadatos">
    No hay información disponible
  </section>
</body>
</html>
```

El fichero indexed.js

Las funciones *errores*, *abrir* y *creabd* no cambian.

```
var cajadatos, bd;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('buscar');
  boton.addEventListener('click', buscarobjetos);

  window.indexedDB =      window.indexedDB || window.mozIndexedDB ||
                           window.webkitIndexedDB || window.msIndex
  window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||
                           window.msIDBTransaction;
  window.IDBKeyRange =    window.IDBKeyRange || window.webkitIDBKeyRange ||
                           window.msIDBKeyRange;
  var solicitud = window.indexedDB.open('mibase', 1); // solicita creación/apertura
  solicitud.addEventListener('upgradeneeded', creabd);
  solicitud.addEventListener('error', errores);
  solicitud.addEventListener('success', abrir);
}

function buscarobjetos(){
  cajadatos.innerHTML = "";
  var buscar      = document.getElementById('fecha').value;
  var transaccion = bd.transaction(['películas']);
  var almacen     = transaccion.objectStore('películas');
  var índice      = almacen.index('BuscarFecha');
  var rango       = IDBKeyRange.only(buscar); // rango de coincidencia exacta
  var cursor      = índice.openCursor(rango);
  cursor.addEventListener('success', mostrarlista);
}
```

```
function mostrarlista(e){
    var cursor = e.target.result;
    if(cursor){
        cajadatos.innerHTML += '<div>' + cursor.value.id + ' - ' + cursor.value.nombre
                                + ' - ' + cursor.value.fecha+'</div>';
        cursor.continue();
    }
}
window.addEventListener('load', iniciar);
```

Para definir los rangos del *openCursor*, además del método *only*, se dispone de los siguientes:

- bound (bajo, alto, bajoIgnoradoBool, altoIgnoradoBool)
- lowerbound (valor, ignorarBool)
- upperbound (valor, ignorarBool)

Transacción con más de un *objectStore*

```
tx = db.transaction(['st1', 'st2'], 'readwrite');
ob1 = tx.objectStore('st1');
ob2 = tx.objectStore('st2');
ob1.openCursor().onsuccess = function(e1) {
    obj2.openCursor().onsuccess = function(e2) {
        e2.result.put(e1.result.value);
    }
}
```

FILE

Lectura de un archivo de texto

```
<!DOCTYPE html>
<head>
    <title>File API</title>
    <link rel="stylesheet" href="file.css">
    <script src="file.js"></script>
</head>
<body>
    <section id="cajaformulario">
        <form name="formulario">
            <label for="archivos">Archivos:</label>
            <input type="file" name="archivos" id="archivos"></p>
        </form>
    </section>
    <section id="cajadatos">No se seleccionaron archivos</section>
</body>
</html>
```

El fichero file.css

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 500px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
.directorio{
  color: #0000FF;
  font-weight: bold;
  cursor: pointer;
}
```

El fichero file.js

```
var cajadatos;

function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  var archivos=document.getElementById('archivos');
  archivos.addEventListener('change', procesar);
}

function procesar(e){
  var archivos=e.target.files;      // devuelve colección FileList
  var archivo=archivos[0];          // File
  var lector=new FileReader();
  lector.addEventListener("load", mostrar);
  lector.readAsText(archivo);
}

function mostrar(e){
  var resultado=e.target.result;
  cajadatos.innerHTML = resultado;
}

window.addEventListener('load', iniciar);
```

Cargar imágenes

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var archivos=document.getElementById('archivos');
    archivos.addEventListener('change', procesar);
}
function procesar(e){
    var archivos=e.target.files;
    cajadatos.innerHTML = "";
    var archivo=archivos[0];

    // .. cq carácter excepto nueva línea *: 0 ó más veces i: case insensitive
    if(!archivo.type.match(/image.*/i)){
        alert('Error: seleccione una imagen');
    }else{
        // cada objeto devuelto por <input> en la propiedad files:
        cajadatos.innerHTML += 'Nombre: ' + archivo.name + '<br>';
        cajadatos.innerHTML += 'Tamaño: ' + archivo.size + ' bytes<br>';
        // tb archivo.type: (tipos MIME) image.jpeg, image.gif ...
        var lector=new FileReader();
        lector.addEventListener("load", mostrar);
        lector.readAsDataURL(archivo);
        // data:url → formato q permite introducir datos en línea como si fuera URL
    }
}
function mostrar(e){
    var resultado=e.target.result;
    cajadatos.innerHTML += '';
}
window.addEventListener('load', iniciar);
```

Trabajo con blobs

Los blobs permiten procesar (subir, por ejemplo) archivos por trozos.

El método *slice* devuelve un blob a partir de otro o un fichero. Los argumentos que reciben son la posición del primer byte, the posición del último byte, y un tipo MIME opcional (por ejemplo, "text/plain") que se le aplicará al blob (por defecto, el mismo que el blob original).

```
var cajadatos;
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var archivos=document.getElementById('archivos');
    archivos.addEventListener('change', procesar);
}
function procesar(e){
    cajadatos.innerHTML = "";
    var archivos = e.target.files;
    var archivo=archivos[0];
    var lector=new FileReader();
    lector.addEventListener("load", function(e){ mostrar(e, archivo)} );
    // define como un blob lo q hay q leer y luego ordena la lectura:
    var blob=archivo.slice(0,1000); // mozSlice, webKitSlice
    lector.readAsBinaryString(blob); // para mostrar como img eliminar línea anterior
}
```

```
function mostrar(e, archivo){
    var resultado = e.target.result;
    cajadatos.innerHTML = 'Nombre: ' + archivo.name + '<br>';
    cajadatos.innerHTML += 'Tipo: ' + archivo.type + '<br>';
    cajadatos.innerHTML += 'Tamaño: ' + archivo.size+' bytes<br>';
    cajadatos.innerHTML += 'Tamaño Blob: ' + resultado.length + ' bytes<br>';
    cajadatos.innerHTML += 'Blob: ' + resultado;
    // para mostrar el blob como una imagen como imagen:
    // var imagen = document.createElement("img");
    // imagen.src = URL.createObjectURL(archivo);
    // cajadatos.appendChild(imagen);
}
window.addEventListener('load', iniciar);
```

Control de la lectura por medio de eventos

Las etapas de la carga se pueden detectar por medio de los eventos que se disparan:

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var archivos=document.getElementById('archivos');
    archivos.addEventListener('change', procesar);
}
function procesar(e){
    cajadatos.innerHTML = "";
    var archivos=e.target.files;
    var archivo=archivos[0];
    var lector=new FileReader();
    lector.addEventListener("loadstart", comenzar); // inicia la carga
    lector.addEventListener("progress", estado); // periódicamente
    lector.addEventListener("loadend",function(){ mostrar(archivo); });
    // tb existen los eventos: abort, error, load (completada con éxito)
    lector.readAsBinaryString(archivo);
}
function comenzar(e){
    cajadatos.innerHTML='<progress value="0" max="100">0%</progress>';
}
function estado(e){
    var por=parseInt(e.loaded/e.total*100);
    cajadatos.innerHTML='<progress value="'+por+'" max="100">'+por+'%</progress>';
}
function mostrar(archivo){
    cajadatos.innerHTML='Nombre: '+archivo.name+'<br>';
    cajadatos.innerHTML+='Tipo: '+archivo.type+'<br>';
    cajadatos.innerHTML+='Tamaño: '+archivo.size+' bytes<br>';
}
window.addEventListener('load', iniciar);
```

Esta nueva especificación de AJAX permite acceder a información de múltiples servidores, transferencia de datos binarios y monitorizar el progreso de la transferencia por medio de eventos.

Lectura de un archivo en el servidor

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Ajax</title>
  <link rel="stylesheet" href="ajax.css">
  <script src="ajax.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p><button type="button" id="boton">Aceptar</p>
    </form>
  </section>
  <section id="cajadatos"></section>
</body>
</html>
```

El archivo ajax.css:

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 500px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
```

El archivo ajax.js

```
var cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  var boton = document.getElementById('boton');
  boton.addEventListener('click', leer);
}
function leer(){
  var url = "texto.txt";
  var solicitud = new XMLHttpRequest();
  solicitud.addEventListener('load',mostrar); // evento load: descarga ok
  solicitud.open("GET", url, true); // GET/POST, url, asincr (bool), usuario, clave
  solicitud.send(null);
}
function mostrar(e){ cajadatos.innerHTML = e.target.responseText; }
window.addEventListener('load', iniciar);
```


Lectura de una imagen del servidor

La función iniciar no cambiaría. Las funciones leer y mostrar serían así:

```
function leer() {
    var url = "imagen.jpg";
    var solicitud = new XMLHttpRequest();
    solicitud.responseType = "blob";
    solicitud.addEventListener("load", mostrar);
    solicitud.open("GET", url, true);
    solicitud.send(null); // null: no se envía información junto con la petición
}
function mostrar(e) {
    var datos = e.target;
    if (datos.status == 200) { // 200: ok, 404: no encontrada
        var imagen = URL.createObjectURL(datos.response);
        cajadatos.innerHTML = '';
    }
}
window.addEventListener("load", initiate);
```

Monitorización del progreso de la solicitud

```
function iniciar(){
    cajadatos = document.getElementById('cajadatos'); // variable global
    document.getElementById('boton').addEventListener('click', leer);
}
function leer(){
    var url="trailer.ogg";
    var solicitud = new XMLHttpRequest();
    solicitud.addEventListener('loadstart', comenzar);
    solicitud.addEventListener('progress', estado);
    solicitud.addEventListener('load', mostrar);
    solicitud.open("GET", url, true);
    solicitud.send(null);
}
function comenzar(){
    var progreso = document.createElement("progress");
    progreso.value = 0;
    progreso.max = 100;
    progreso.innerHTML = "0%";
    cajadatos.appendChild(progreso);
}
function estado(e){ // e: evento de tipo progress
    if(e.lengthComputable){ // (bool) indica si se puede calcular progreso
        var porc = Math.ceil(e.loaded/e.total*100);
        var barraprogreso = cajadatos.querySelector("progress");
        barraprogreso.value = porc;
        barraprogreso.innerHTML = porc + '%';
    }
}
function mostrar(e){
    cajadatos.innerHTML='Terminado';
}
window.addEventListener('load', iniciar);
```

Otros eventos son: *abort*, *error* y *loadend* (tanto éxito como fallo). Si se establece la propiedad *timeout* del objeto XMLHttpRequest (en milisegundos), se disparará el evento *timeout* cuando se cumpla el período y se termine la conexión.

Envío de un formulario virtual

```
var cajadatos;
function iniciar(){
    cajadatos = document.getElementById('cajadatos');
    var boton = document.getElementById('boton');
    boton.addEventListener('click', enviar);
}
function enviar(){
    var datos = new FormData();
    datos.append('nombre', 'Juan');
    datos.append('apellido', 'Paz');
    var url = "procesar.php";
    var solicitud = new XMLHttpRequest();
    solicitud.addEventListener('load', mostrar);
    solicitud.open("POST", url, true);
    solicitud.send(datos); // DOMString, Document, FormData, Blob, File o ArrayBuffer
}
function mostrar(e){
    var datos = e.target;
    if (datos.status == 200) {
        cajadatos.innerHTML = datos.responseText; // e.target: tipo XMLHttpRequest
    }
}
window.addEventListener('load', iniciar);
```

El tipo de respuesta (*solicitud.responseType*) puede tomar los valores "DOMString" (valor por defecto), "text", "arraybuffer" (imágenes por defecto), "blob", "json" o "document".

Respuesta simple a una solicitud post (procesar.php)

```
<?PHP
    print('Su nombre es: '.$_POST['nombre'].'<br>');
    print('Su apellido es: '.$_POST['apellido']);
?>
```

Autorizar solicitudes de orígenes múltiples

En AJAX nivel 2, es posible trabajar con orígenes múltiples.

```
<?PHP
    header('Access-Control-Allow-Origin: *');
    print('Su nombre es: '.$_POST['nombre'].'<br>');
    print('Su apellido es: '.$_POST['apellido']);
?>
```

La cabecera con la directiva *Access-Control-Allow-Origin* permite indicar un único origen permitido. El asterisco representa cualquier origen. Por ejemplo: 'Access-Control-Allow-Origin: http://momentum.uvigo.es'

En lugar de especificarlo en la cabecera de la página de destino, se puede indicar en la configuración del servidor.

Subir archivos con FormData

El objeto XMLHttpRequestUpload permite el acceso a todas las propiedades, métodos y eventos de XMLHttpRequest y además, controlar el proceso de subida.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>Ajax</title>
  <link rel="stylesheet" href="ajax.css">
  <script src="ajax.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <label for="archivos"> Archivo a Subir: </label>
      <input type="file" name="archivos" id="archivos">
    </form>
  </section>
  <section id="cajadatos"></section>
</body>
</html>
```

El archivo ajax.js

```
var cajadatos;
function iniciar(){
  cajadatos = document.getElementById('cajadatos');
  document.getElementById('archivos').addEventListener('change', subir);
}

function subir(e){
  var archivos = e.target.files;
  var archivo = archivos[0];
  var datos = new FormData();
  datos.append('archivo', archivo); // (par nombre/valor) Tb vale sin XHRU
  var url = "procesar.php";
  var solicitud=new XMLHttpRequest();
  solicitud.addEventListener('loadstart',comenzar); // con él como si fuera
  solicitud.addEventListener('load',mostrar);
  var xmlhttp=solicitud.upload; // obtiene el XMLHttpRequestUpload y trabaja
  xmlhttp.addEventListener('progress',estado); // el XMLHttpRequest
  solicitud.open("POST", url, true);
  solicitud.send(datos);
}

function comenzar(){
  var progreso = document.createElement("progress");
  progreso.value = 0;
  progreso.max = 100;
  progreso.innerHTML = "0%";
  cajadatos.appendChild(progreso);
}
```

```

function estado(e){
    if(e.lengthComputable){
        var porc = parseInt(e.loaded/e.total*100);
        var barraprogreso = cajadatos.querySelector("progress");
        barraprogreso.value = porc;
        barraprogreso.innerHTML = porc + '%';
    }
}
function mostrar(e){
    if (e.target.status == 200) cajadatos.innerHTML = "Terminado";
}
window.addEventListener('load', iniciar);

```

Subir archivos uno por uno

```

<!DOCTYPE html>
<html lang="es">
<head>
    <title>Ajax Level 2</title>
    <link rel="stylesheet" href="ajax.css">
    <script src="ajax.js"></script>
</head>
<body>
    <section id="cajados">
        <p>Suelta los archivos aquí</p>
    </section>
</body>
</html>

```

El archivo ajax.js

```

var cajadatos;
function iniciar(){
    cajadatos = document.getElementById('cajados');
    cajadatos.addEventListener('dragenter',function(e){e.preventDefault();});
    cajadatos.addEventListener('dragover', function(e){e.preventDefault();});
    cajadatos.addEventListener('drop', soltado);
}
function soltado(e){
    e.preventDefault();
    var archivos = e.dataTransfer.files;
    if(archivos.length){

        var lista = "";
        for(var f = 0; f <archivos.length; f++){
            var archivo = archivos[f];
            lista += '<div>Archivo: '+archivo.name;
            lista += '<br><span><progress value="0" max="100">0%</progress></span>';
            lista += '</div>'; // blockquote: citas (no necesariamente comillas)
        }
        cajadatos.innerHTML = lista;
    }
}

```

```

var cuenta = 0;
var subir = function(){
    var archivo = archivos[cuenta];
    var datos = new FormData();
    datos.append('archivo', archivo);
    var url = "procesar.php";
    var solicitud = new XMLHttpRequest(); // un XHRU para cada fichero
    var xmlupload = solicitud.upload;
    xmlupload.addEventListener(
        'progress',
        function(e){
            if(e.lengthComputable){
                var hijo = cuenta + 1;
                var porc = parseInt(e.loaded / e.total * 100);
                var barraprogreso = cajadatos.querySelector(
                    "div:nth-child(" + hijo + ") > span > progress");
                barraprogreso.value = porc;
                barraprogreso.innerHTML = porc + '%';
            }
        }
    );
    xmlupload.addEventListener(
        'load',
        function(){
            var hijo = cuenta + 1;
            var elemento = cajadatos.querySelector(
                "div:nth-child(" + hijo + ") > span");
            elemento.innerHTML = 'Terminado';
            cuenta++;
            if(cuenta<archivos.length){
                subir();
            } // sube el siguiente
        }
    );
    solicitud.open("POST", url, true);
    solicitud.send(datos);
}
subir();
}
window.addEventListener('load', iniciar);

```

Envío de mensajes al servidor

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>WebSocket</title>
  <link rel="stylesheet" href="websocket.css">
  <script src="websocket.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p>Comando:<br><input type="text" name="comando" id="comando"></p>
      <p><input type="button" name="boton" id="boton" value="Enviar"></p>
    </form>
  </section>
  <section id="cajadatos"></section>
</body>
</html>
```

El fichero websocket.css

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;          width: 500px;
  height: 350px;        overflow: auto;
  margin-left: 20px;    padding: 20px;
  border: 1px solid #999999;
}
```

El fichero websocket.js

```
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  var boton=document.getElementById('boton');
  boton.addEventListener('click', enviar);
  socket=new WebSocket("ws://echo.websocket.org/");
  socket.addEventListener('message', recibido);
}
function recibido(e){
  var lista=cajadatos.innerHTML;
  cajadatos.innerHTML='Recibido: ' + e.data + '<br>' + lista;
}
function enviar(){
  var comando=document.getElementById('comando').value;
  socket.send(comando); // sólo strings
}
window.addEventListener('load', iniciar);
```

Informar del estado de la conexión

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var boton=document.getElementById('boton');
    boton.addEventListener('click', enviar);
    socket=new WebSocket("ws://www.dominio.com:12345/server.php");
    socket.addEventListener('open', abierto);
    socket.addEventListener('message', recibido);
    socket.addEventListener('close', cerrado);
    socket.addEventListener('error', errores);
}
function abierto(){
    cajadatos.innerHTML='CONEXION ABIERTA<br>';
    cajadatos.innerHTML+='Estado: ' + socket.readyState;
}
function recibido(e){
    var lista=cajadatos.innerHTML;
    cajadatos.innerHTML='Recibido: ' + e.data + '<br>' + lista;
}
function cerrado(){
    var lista=cajadatos.innerHTML;
    cajadatos.innerHTML='CONEXION CERRADA<br>' + lista;
    var boton=document.getElementById('boton');
    boton.disabled=true;
}
function errores(){
    var lista=cajadatos.innerHTML;
    cajadatos.innerHTML='ERROR<br>' + lista;
}
function enviar(){
    var comando=document.getElementById('comando').value;
    if(comando=='cerrar') socket.close();
    else socket.send(comando);
}
}
window.addEventListener('load', iniciar);
```

Webworkers

Se ejecutan en segundo plano para que las páginas no queden bloqueadas por scripts lentos. No pueden acceder al DOM no tener el suyo propio. Tampoco pueden acceder a los scripts de la página. Solo pueden acceder a algunas propiedades de la ventana (objeto window).

Ejemplo básico de uso del api

```
<!DOCTYPE html>
<html lang="es">
<head>
  <title>WebWorkers</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="webworkers.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p>Nombre:<br><input type="text" name="nombre" id="nombre"></p>
      <p><input type="button" name="boton" id="boton" value="Enviar"></p>
    </form>
  </section>
  <section id="cajadatos"></section>
</body>
</html>
```

El archivo webworkers.css

```
#cajaformulario{
  float: left;
  padding: 20px;
  border: 1px solid #999999;
}
#cajadatos{
  float: left;
  width: 500px;
  margin-left: 20px;
  padding: 20px;
  border: 1px solid #999999;
}
```

El archivo webworkers.js

```
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  var boton=document.getElementById('boton');
  boton.addEventListener('click', enviar);

  trabajador=new Worker('trabajador.js');
  trabajador.addEventListener('message', recibido);
}
function enviar(){
  var nombre=document.getElementById('nombre').value;
  trabajador.postMessage(nombre); // string u objeto JSON
}
function recibido(e){
  cajadatos.innerHTML = e.data;
}
window.addEventListener('load', iniciar);
```


Comprobación de API soportada

```
if(typeof(Worker) !== "undefined") {  
    // código  
} else {  
    // código si API no soportada  
}
```

Código del *trabajador* (trabajador.js)

```
addEventListener('message', recibido);  
function recibido(e){  
    var respuesta='Su nombre es ' + e.data;  
    postMessage(respuesta);  
}
```

Uso del evento error

```
function iniciar(){  
    cajadatos=document.getElementById('cajadatos');  
    var boton=document.getElementById('boton');  
    boton.addEventListener('click', enviar);  
  
    trabajador=new Worker('trabajador.js');  
    trabajador.addEventListener('error', errores);  
}  
function enviar(){  
    var nombre=document.getElementById('nombre').value;  
    trabajador.postMessage(nombre);  
}  
function errores(e){  
    cajadatos.innerHTML='ERROR: ' + e.message + '<br>';  
    cajadatos.innerHTML+='Archivo: ' + e.filename + '<br>'; // fichero js  
    cajadatos.innerHTML+='Línea: ' + e.lineno;  
}  
window.addEventListener('load', iniciar);
```

Llamada a una función inexistente para provocar un error

```
addEventListener('message', recibido); // código del trabajador  
function recibido(e){  
    prueba();  
}
```

Detener al trabajador desde el código principal

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var boton=document.getElementById('boton');
    boton.addEventListener('click', enviar);
    trabajador=new Worker('trabajador.js');
    trabajador.addEventListener('message', recibido);
}
function enviar(){
    var nombre=document.getElementById('nombre').value;
    if(nombre=='cerrar1'){
        trabajador.terminate();
        cajadatos.innerHTML = 'Trabajador Detenido';
    }else{
        trabajador.postMessage(nombre);
    }
}
function recibido(e){
    cajadatos.innerHTML = e.data;
}
window.addEventListener('load', iniciar);
```

El trabajador se detiene a sí mismo

Para que el cierre de un trabajador no sea abrupto y queden tareas pendientes, la mejor estrategia es notificar al trabajador que debe terminar.

```
addEventListener('message', recibido);
function recibido(e){
    if(e.data=='cerrar2'){
        postMessage('Trabajador Detenido');
        close();
    }else{
        var respuesta='Su nombre es ' + e.data;
        postMessage(respuesta);
    }
}
```

Cargar códigos javascript para el trabajador desde archivos externos

```
importScripts('mascodigos.js');
addEventListener('message', recibido);
function recibido(e){
    prueba();
}
```

Trabajadores compartidos: conectar desde el documento principal

Cualquier script ejecutándose en la misma página puede comunicarse con ellos. Restricción: no funciona sobre fichero ni localhost. Se comunica a través de puertos.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <title>WebWorkers</title>
  <link rel="stylesheet" href="webworkers.css">
  <script src="webworkers.js"></script>
</head>
<body>
  <section id="cajaformulario">
    <form name="formulario">
      <p>Nombre:<br><input type="text" name="nombre" id="nombre"></p>
      <p><input type="button" name="boton" id="boton" value="Enviar"></p>
    </form>
  </section>
  <section id="cajadatos">
    <iframe id="iframe" src="iframe.html" width="500" height="350"></iframe>
  </section>
</body>
</html>

```

Fichero para el iframe (iframe.html)

```

<!DOCTYPE html>
<html lang="es">
<head>
  <title>iframe</title>
  <script src="iframe.js"></script>
</head>
<body>
  <section id="cajadatos">
  </section>
</body>
</html>

```

El archivo webworkers.js

```

function iniciar(){
  var boton=document.getElementById('boton');
  boton.addEventListener('click', enviar);
  trabajador=new SharedWorker('trabajador.js');// 2º arg opc: nombre del trabajador
  trabajador.port.addEventListener('message', recibido);
  trabajador.port.start();
}
function recibido(e){
  alert(e.data);
}
function enviar(){
  var nombre=document.getElementById('nombre').value;
  trabajador.port.postMessage(nombre);
}
window.addEventListener('load', iniciar);

```

Conectar desde el iframe (iframe.js)

```

function iniciar(){ // conecta con el mismo trabajador pq es la misma URL:
  trabajador=new SharedWorker('trabajador.js');
  trabajador.port.addEventListener('message', recibido);
  trabajador.port.start();
} // los (trabajadores se identifican por su fichero js)

```

```
function recibido(e){
    var cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML = e.data;
}
window.addEventListener('load', iniciar);
```

Código para el trabajador compartido (trabajador.js)

```
puertos=new Array();
addEventListener('connect', conectar);
function conectar(e){
    puertos.push(e.ports[0]); // guarda el puerto de la nueva conexión
    e.ports[0].onmessage=enviar; // nuevo mensaje -> llama a función que
} // envía un nuevo mensaje a todos
function enviar(e){
    for(f=0; f < puertos.length; f++){
        puertos[f].postMessage('Su nombre es ' + e.data);
    }
}
```

History

Generar una nueva URL y nuevo contenido

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title>History API</title>
    <link rel="stylesheet" href="history.css">
    <script src="history.js"></script>
</head>
<body>
    <section id="contenido">
        Este contenido nunca es actualizado<br>
        <span id="url">página 2</span>
    </section>
    <aside id="cajadatos"></aside>
</body>
</html>
```

El archivo history.css

```
#contenido{
    float: left;
    padding: 20px;
    border: 1px solid #999999;
}
#cajadatos{
    float: left;
    width: 500px;
    margin-left: 20px;
    padding: 20px;
    border: 1px solid #999999;
}
```

```
#contenido span{
  color: #0000FF;
  cursor: pointer;
}
```

El archivo history.js

```
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  url=document.getElementById('url');
  url.addEventListener('click', cambiar);
}

function cambiar(){
  cajadatos.innerHTML='La URL es pagina2';
  window.history.pushState(null, null, 'pagina2.html');
}

window.addEventListener('load', iniciar);
```

Controlar la ubicación del usuario

```
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  url=document.getElementById('url');
  url.addEventListener('click', cambiar);
  window.addEventListener('popstate', nuevaurl);
  window.history.replaceState(1, null);
}

function cambiar(){
  mostrar(2);
  window.history.pushState(2, null, 'pagina2.html');
}

function nuevaurl(e){
  mostrar(e.state);
}

function mostrar(actual){
  cajadatos.innerHTML='La URL es página '+actual;
}

window.addEventListener('load', iniciar);
```

Manipular el historial

```
<!DOCTYPE html>
<head>
  <title>History API</title>
  <link rel="stylesheet" href="history.css">
  <script src="history.js"></script>
</head>
```

```

<body>
  <section id="contenido">
    Este contenido nunca es actualizado <br>
    <span id="url1">imagen 1</span> -
    <span id="url2">imagen 2</span> -
    <span id="url3">imagen 3</span> -
    <span id="url4">imagen 4</span> -
  </section>
  <aside id="cajadatos">
    
  </aside>
</body>
</html>

```

El archivo history.js

```

function iniciar(){
  for(var f=1;f<5;f++){
    url=document.getElementById('url'+f);
    url.addEventListener('click',
      function(x){return function(){ cambiar(x);}}(f),
    );
  }
  window.addEventListener('popstate', nuevaurl);
  window.history.replaceState(1, null, 'pagina1.html');
}
function cambiar(pagina){
  mostrar(pagina);
  window.history.pushState(pagina, null, 'pagina'+pagina+'.html');
}
function nuevaurl(e){
  mostrar(e.state);
}
function mostrar(actual){
  if(actual!=null){
    imagen=document.getElementById('imagen');
    imagen.src='http://momentum.uvigo.es/multimedia/monster' + actual + '.gif';
  }
}
window.addEventListener('load', iniciar);

```

El evento *popstate* ocurre cada vez que cambia la entrada al historial.

replaceState() modifica la entrada al historial actual, asignando un objeto cualquiera (con pocas restricciones), un título (normalmente ignorado) y (opcionalmente) una URL. No recarga la página, y si se pulsa el botón de retroceso en el navegador, no nos lleva a la página original sino a la anterior.

pushState() actúa como el anterior, pero creando una nueva entrada, por lo que el botón de retroceso del navegador sí que nos llevaría a la página actual.

Otros métodos y atributos de *window.history* son *forward()*, *back()*, *length* y *go(n)*. Este último, toma un argumento entero que también puede ser negativo.

Se recarga la caché de la aplicación si el usuario la limpia, se actualiza por programa o se modifica el archivo de manifiesto.

Estados de la caché: **UNCACHED** (un objeto de la caché no está completamente inicializado) , **IDLE** (la caché no está siendo actualizada), **CHECKING** (se está buscando el manifiesto y comprobando si hay actualizaciones), **DOWNLOADING** (se están descargando recursos para añadirlos a la caché, debido a un cambio en los recursos del manifiesto), **UPDATEREADY** (está disponible una nueva versión de la caché) y **OBSOLETE**.

Eventos: *checking* (manifiesto revisado), *downloading* (descargado), *chached* (caché lista), *updateready* (nueva versión disponible), *obsolete* (caché obsoleta) y *noupdate* (manifiesto actualizado)

Archivo de manifiesto

```
CACHE MANIFEST
cache.html
cache.css
cache.js
```

Declarando archivos por categoría

```
CACHE MANIFEST

CACHE:
cache.html
cache.css
cache.js
NETWORK:
chat.html
FALLBACK:
noticias.html sinnoticias.html
```

Nuevo comentario para informar sobre actualizaciones

```
CACHE MANIFEST
CACHE:
cache.html
cache.css
cache.js
NETWORK:
chat.html
FALLBACK:
noticias.html sinnoticias.html
# fecha 2015/08/10
```

Cargar el archivo manifiesto

```
<!DOCTYPE html>
<html lang="es" manifest="micache.manifest">
<head>
  <title>Offline API</title>
  <link rel="stylesheet" href="cache.css">
  <script src="cache.js"></script>
</head>
<body>
  <section id="cajadatos">
    Aplicación para trabajar sin conexión
  </section>
</body>
</html>
```

Control de errores

```
function iniciar(){
  var cache=window.applicationCache;
  cache.addEventListener('error', errores);
}
function errores(){
  alert('error');
}
window.addEventListener('load', iniciar);
```

El archivo cache.css

```
#cajadatos{
  width: 500px;
  height: 300px;
  margin: 10px;
  padding: 10px;
  border: 1px solid #999999;
}
```

Control del estado de la conexión

```
function iniciar(){
  cajadatos=document.getElementById('cajadatos');
  window.addEventListener('online', function(){ estado(1); });
  window.addEventListener('offline', function(){ estado(2); });
}
function estado(valor){
  switch(valor){
    case 1:
      cajadatos.innerHTML+="
```


Control de la conexión

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    cache=window.applicationCache;
    cache.addEventListener('checking', function(){ mostrar(1); });
    cache.addEventListener('downloading', function(){ mostrar(2); });
    cache.addEventListener('cached', function(){ mostrar(3); });
    cache.addEventListener('updateready', function(){ mostrar(4); });
    cache.addEventListener('obsolete', function(){ mostrar(5); });
}
function mostrar(valor){
    cajadatos.innerHTML += '<br>Estado: ' + cache.status + ' | Evento: ' + valor;
}
window.addEventListener('load', iniciar);
```

Control de la descarga

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    cajadatos.innerHTML='<progress value="0" max="100">0%</progress>';
    cache=window.applicationCache;
    cache.addEventListener('progress', progreso);
    cache.addEventListener('cached', mostrar);
    cache.addEventListener('updateready', mostrar);
    cache.addEventListener('noupdate', mostrar);
}
function progreso(e){
    if(e.lengthComputable){
        var por=parseInt(e.loaded/e.total*100);
        var barraprogreso=cajadatos.querySelector("progress");
        barraprogreso.value=por;
        barraprogreso.innerHTML=por+'%';
    }
}
function mostrar(){ cajadatos.innerHTML='Terminado'; }
window.addEventListener('load', iniciar);
```

Actualizar la caché y comprobando la versión actual

```
<!DOCTYPE html>
<html lang="es" manifest="micache.manifest">
  <head>
    <title>Offline API</title>
    <link rel="stylesheet" href="cache.css">
    <script src="cache.js"></script>
  </head>
  <body>
    <section id="cajadatos"> Aplicación para trabajar sin conexión </section>
    <button id="actualizar">Actualizar Caché</button>
    <button id="prueba">Verificar</button>
  </body>
</html>
```

El archivo cache.js

```
function iniciar(){
    cajadatos=document.getElementById('cajadatos');
    var actualizar=document.getElementById('actualizar');
    actualizar.addEventListener('click', actualizarcache);
    var prueba=document.getElementById('prueba');
    prueba.addEventListener('click', probarcache);
    cache=window.applicationCache;
    cache.addEventListener('updateready', function(){ mostrar(1); });
    cache.addEventListener('noupdate', function(){ mostrar(2); });
}
function actualizarcache(){
    cache.update(); // fuerza la actualización
}
function probarcache(){
    cajadatos.innerHTML+='\n<br>cambiar este mensaje';
}
function mostrar(valor){
    switch(valor){
        case 1:
            cajadatos.innerHTML+='\n<br>Actualización Lista'; break;
        case 2:
            cajadatos.innerHTML+='\n<br>Actualización No Disponible'; break;
    }
}
window.addEventListener('load', iniciar);
```

Modernizr

Detectando la disponibilidad de estilos css para generar sombras

```
<!DOCTYPE html>
<html lang="es">
<head>
    <script src="modernizr.min.js"></script>
    <script src="modernizr.js"></script>
</head>
<body>
    <section id="cajadatos">    contenido    </section>
</body>
</html>
```

El archivo modernizr.js

```
function iniciar(){
    var cajadatos=document.getElementById('cajadatos');
    if(Modernizr.boxshadow){
        cajadatos.innerHTML='Box Shadow está disponible';
    }else{
        cajadatos.innerHTML='Box Shadow no está disponible';
    }
}
window.addEventListener('load', iniciar);
```

jQuery

Introducción

jQuery es una biblioteca de funciones JavaScript y tiene como objetivo simplificar el uso de JavaScript, proporcionando funciones que realizan de modo transparente muchas tareas comunes cuando se programa con JavaScript, incluyendo AJAX y el uso de DOM: manipulación HTML/DOM, manipulación CSS, gestión de eventos HTML, animaciones y efectos visuales, AJAX y utilidades varias. jQuery se comporta del mismo modo en los principales navegadores de Internet, ocultando así las peculiaridades de cada uno de ellos.

Además de lo anterior, jQuery tiene plugins para multitud de tareas (formularios, animaciones, interfaz de usuario...)

Existen frameworks alternativos, pero jQuery es el más popular y extensible. La mayoría de las grandes empresas en la Web usan jQuery: Google, Microsoft, Netflix, IBM, etc.

Cómo usar jQuery en páginas web

Hay varias formas de usar jQuery en páginas web:

- Descargando la biblioteca jQuery de jquery.com
- Incluyendo jQuery desde un CDN (*Content Delivery Network*), como por ejemplo Google

Descarga de jQuery

Hay dos versiones de jQuery disponibles:

- Versión de producción: versión minimizada y comprimida
- Versión de desarrollo: legible y sin comprimir para comprobaciones y desarrollo.

Ambas versiones se pueden descargar de jquery.com

La biblioteca jQuery es un único fichero JavaScript, a la que se hace referencia desde HTML por medio de la etiqueta `<script>` (que a su vez debe estar dentro de la sección `<head>`). En la etiqueta `<script>` no es necesario incluir `type="text/javascript"`, porque es JavaScript es la opción por defecto en la mayoría de los navegadores modernos. En HTML5, también es el lenguaje de script por defecto.

Por ejemplo:

```
<head>
  <script src="jquery-1.11.3.min.js"></script>
</head>
```

El fichero descargado debe guardarse en el mismo directorio que las páginas que lo usen.

Inclusión desde un CDN (*Content Delivery Network*)

Si no se desea descargar, la biblioteca jQuery se puede incluir desde un CDN (Content Delivery Network): Google, Microsoft, jQuery, CDNJS y jsDelivr

Google:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
</script>
```

Microsoft:

```
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.11.3.min.js"></script>
```

La mayor ventaja de usar Google o Microsoft como CDN para jQuery es que cuando alguien visita nuestra página, si ya tiene descargada jQuery por haber visitado otros sitios, se utilizará la versión guardada en la caché, en lugar de volver a descargarla. Esto supone un ahorro de tiempo considerable.

Sintaxis de jQuery

La sintaxis de jQuery permite seleccionar elementos HTML y realizar acciones sobre ellos. jQuery usa sintaxis CSS para seleccionar elementos.

La sintaxis básica es: `$(selector).accion()`

- El signo \$ para definir o acceder a jQuery
- (*selector*) permite indicar los elementos HTML a usar

accion() indica la acción a realizar sobre los elementos HTML seleccionados

Por ejemplo:

```
$( "p" ).hide()      // oculta todos los elementos <p>
$( ".test" ).hide()  // oculta todos los elementos de la clase "test" (class="test")
$( "#test" ).hide()  // oculta todos los elementos cuyo atributo id="test"
```

El evento de documento listo

```
$(document).ready(function() {
    // aquí van los métodos jQuery ...
});
```

Esto evita que se ejecute el código jQuery antes de que el documento haya acabado de cargarse, como por ejemplo, si trata de leer el tamaño de una imagen que no se haya cargado todavía.

Existe un método más corto para el evento de documento preparado:

```
$(function() {
    // inicializaciones
});
```

Selectores jQuery

Los selectores permiten seleccionar y manipular los elementos HTML, usando sus identificadores (id), clases (class), tipos, atributos, valores de atributos, etc. Como ya se ha comentado, jQuery se basa en los selectores de CSS y añade algunos selectores propios.

Todos los selectores en jQuery, comienzan con el signo del dolar y paréntesis: `$()`.

Selector de elementos

El selector de elemento de jQuery selecciona elementos basándose en su nombre de etiqueta. Por ejemplo, se pueden seleccionar todos los elementos `<p>` de una página web así: `$("p")`

Por ejemplo, cuando un usuario hace click en un botón, todos los elementos `<p>` se ocultarán:

```
$(document).ready(function() {
    $("button").click(function() {
        $("p").hide();
    });
});
```

El selector #id

El selector `#id` utiliza el atributo *id* de una etiqueta HTML para encontrar el elemento.

Para localizar un elemento a partir de su id, se debe escribir el carácter `#` antes del id. Por ejemplo: `$("#test")`

Por ejemplo, cuando un usuario haga click en un botón, se ocultará el elemento con `id="test"`.

```
$(document).ready(function() {
    $("button").click(function() {
        $("#test").hide();
    });
});
```

El selector .class

El selector de clase (`.class`) localiza elementos de una clase específica. Para ello, se escribe un punto seguido del nombre de la clase: `$(".test")`

Por ejemplo, cuando un usuario pulsa un botón, se ocultarán los elementos con el atributo `class="test"`

```
$(document).ready(function() {
    $("button").click(function() {
        $(".test").hide();
    });
});
```

Selectores en jQuery

<code>\$("*")</code>	Selecciona Los elementos
<code>\$("#final")</code>	El elemento con id="final"
<code>\$(".intro")</code>	Los elementos con el atributo class="intro"
<code>\$(".intro,.demo")</code>	Los elementos con el atributo class = "intro" o "demo"
<code>\$("p")</code>	Los elementos <p>
<code>\$("h1,div,p")</code>	Los elementos <h1>, <div> y <p>
<code>\$("p:first")</code>	El primer elemento <p>
<code>\$("p:last")</code>	El último elemento <p>
<code>\$("tr:even")</code>	Los elementos <tr> pares
<code>\$("tr:odd")</code>	Los elementos <tr> impares
<code>\$("p:first-child")</code>	Los elementos <p> que sean el primer hijo de su padre
<code>\$("p:first-of-type")</code>	Los elementos <p> que sean los primeros <p> hijos de su padres
<code>\$("p:last-child")</code>	Los elementos <p> que sean los últimos hijos de sus padres
<code>\$("p:last-of-type")</code>	Los elementos <p> que sean los últimos <p> hijos de su padres
<code>\$("p:nth-child(n)")</code>	Los elementos <p> que sean los n-ésimos hijos de sus padres
<code>\$("p:nth-last-child(n)")</code>	Los elementos <p> que sean el n-ésimos, contando desde el final
<code>\$("p:nth-of-type(n)")</code>	Los elementos <p> que sean los n-ésimos elementos <p> de sus padres
<code>\$("p:nth-last-of-type(n)")</code>	Los elementos <p> que sean los n-ésimos elementos <p>, contando desde el final
<code>\$("p:only-child")</code>	Los elementos <p> que sean los únicos hijos de sus padres
<code>\$("p:only-of-type")</code>	Los elementos <p> que sean los únicos hijos <p> de sus padres
<code>\$("div > p")</code>	Los elementos <p> que sean hijos directos de un elemento <div>
<code>\$("div p")</code>	Los elementos <p> que sean descendientes de un elemento <div>
<code>\$("div + p")</code>	Los elementos <p> que estén a continuación de un elemento <div>
<code>\$("div ~ p")</code>	Los elementos <p> que sean hermanos de un elemento <div>
<code>\$("ul li:eq(i)")</code>	El n-ésimo elemento de una lista (el índice empieza en 0)
<code>\$("ul li:gt(n)")</code>	Elementos de lista con un índice mayor de n
<code>\$("ul li:lt(n)")</code>	Elementos de lista con un índice menor de n
<code>\$("input:not(:empty)")</code>	Los elementos <i>input</i> que no estén vacíos
<code>\$(":header")</code>	Los elementos <i>header</i> <h1>, <h2> ...
<code>\$(":animated")</code>	Los elementos animados
<code>\$(":focus")</code>	El elemento que actualmente tiene el foco
<code>\$(":contains('ABC')")</code>	Los elementos que contengan el texto "ABC"
<code>\$("div:has(p)")</code>	Los elementos <div> que contengan un elemento <p>
<code>\$(":empty")</code>	Los elementos que estén vacíos
<code>\$(":parent")</code>	Los elementos que sean padres de otro elemento
<code>\$("p:hidden")</code>	Los elementos <p> ocultos
<code>\$("table:visible")</code>	Las tablas visibles
<code>\$(":root")</code>	El elemento raíz del documento
<code>\$("p:lang(de)")</code>	Los elementos <p> con un atributo <i>lang</i> con valor que comience por el texto "de"
<code>\$("[href]")</code>	Los elementos con un atributo <i>href</i>
<code>\$("[href='abc.html']")</code>	Los elementos con un valor del atributo <i>href</i> igual "abc.html"
<code>\$("[href!='abc.html']")</code>	Los elementos con un valor del atributo <i>href</i> distinto de "abc.html"
<code>\$("[href\$.jpg]")</code>	Los elementos con un valor del atributo <i>href</i> que termine en ".jpg"
<code>\$("[xyz]='ABC'")</code>	Los elementos con un atributo <i>xyz</i> con valor 'ABC' o comenzando por 'ABC'
<code>\$("[xyz^='ABC'")</code>	Los elementos con un atributo <i>xyz</i> con valor que empieza por "ABC"
<code>\$("[xyz~='ABC'")</code>	Los elementos con un atributo <i>xyz</i> que contenga la 'ABC' ('123 ABC KLM', p. ej.)
<code>\$("[xyz*='ABC'")</code>	Los elementos con un atributo <i>xyz</i> que contenga el texto 'ABC' ('1ABC2', p. ej.)
<code>\$(":input")</code>	Los elementos <i>input</i>
<code>\$(":text")</code>	Los elementos <i>input</i> con type="text"
<code>\$(":password")</code>	Los elementos <i>input</i> con type="password"
<code>\$(":radio")</code>	Los elementos <i>input</i> con type="radio"
<code>\$(":checkbox")</code>	Los elementos <i>input</i> con type="checkbox"
<code>\$(":submit")</code>	Los elementos <i>input</i> con type="submit"
<code>\$(":reset")</code>	Los elementos <i>input</i> con type="reset"
<code>\$(":button")</code>	Los elementos <i>input</i> con type="button"
<code>\$(":image")</code>	Los elementos <i>input</i> con type="image"

<code>\$(":file")</code>	Los elementos <i>input</i> con <code>type="file"</code>
<code>\$(":enabled")</code>	Los elementos <i>input</i> habilitados (<i>enabled</i>)
<code>\$(":disabled")</code>	Los elementos <i>input</i> inhabilitado (<i>disabled</i>)
<code>\$(":selected")</code>	Los elementos <i>input</i> seleccionados
<code>\$(":checked")</code>	Los elementos <i>input</i> marcados (<i>checked</i>)

Funciones en un archivo separado

Aunque en este documento se suelen escribir las funciones directamente en la sección `<head>` para simplificar la explicación, si el sitio web contiene muchas páginas, las funciones de jQuery serán más fáciles de mantener si se ubican en un archivo `.js` separado. Para ello se usa el atributo `src` para hacer referencia al fichero `.js`. Por ejemplo:

```
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script src="funciones_jquery_abc.js"></script>
</head>
```

Eventos

Un evento representa cualquier acción que el usuario realiza al interactuar con la página, como por ejemplo pasar el cursor sobre un elemento, seleccionar un botón de opción en un formulario o hacer click en un elemento. Es importante recordar que jQuery siempre propaga los eventos en dirección ascendente (bubbling) .

Sintaxis de jQuery para los métodos de respuesta a eventos

Muchos de los eventos DOM tienen un método equivalente de jQuery. Por ejemplo, para asignar un evento *click* a todos los párrafos de una página, se podría usar: `$("p").click()`

El siguiente paso sería definir las acciones que ejecutarían cuando se dispare el evento. Para ello se debe pasar una función al evento:

```
$("p").click(function() {
  // aquí las instrucciones
});
```

Métodos de evento más habituales en jQuery

`$(document).ready():` permite ejecutar una función cuando el documento se haya cargado completamente. Se puede escribir usando sólo el `$`:

```
$(function() {
  // código de inicialización
});
```

click(): asocia una función de manejador de evento al elemento HTML, que se ejecutará cuando el usuario haga click en el elemento. Por ejemplo, para ocultar el elemento <p> actual cuando se haga click sobre el elemento:

```
$( "p" ).click(function() {  
    $(this).hide();  
});
```

this representa el elemento sobre que está ejecutando el código (el elemento sobre el que se disparó el evento). En el ejemplo anterior, se define una función de respuesta al evento *click* de todos los párrafos. Cuando se ejecuta la función de respuesta al evento, *this* representa la párrafo sobre el que se hizo clic.

dblclick(): asocia una función manejadora de evento a un elemento HTML que se ejecutará cuando el usuario haga doble clic en ese elemento. Por ejemplo:

```
$( "p" ).dblclick(function() {  
    $(this).hide();  
});
```

mouseenter(): actúa cuando el puntero entra en un elemento HTML. Por ejemplo:

```
$( "#p1" ).mouseenter(function() {  
    alert("Has entrado en p1!");  
});
```

mouseleave(): actúa cuando el puntero sale de un elemento HTML. Por ejemplo:

```
$( "#p1" ).mouseleave(function() {  
    alert("Has salido de p1!");  
});
```

mousedown(): actúa cuando se baja el botón izquierdo del ratón sobre un elemento HTML. Por ejemplo:

```
$( "#p1" ).mousedown(function() {  
    alert("botón izquierdo del ratón pulsado sobre p1!");  
});
```

mouseup(): actúa cuando se suelta el botón izquierdo del ratón sobre un elemento HTML. Por ejemplo:

```
$( "#p1" ).mouseup(function() {  
    alert("botón izquierdo del ratón liberado sobre p1!");  
});
```

hover(): Este método necesita dos funciones y es la combinación de los métodos *mouseenter()* y *mouseleave()*. La primera función se ejecuta cuando se entra en el elemento HTML y la segunda cuando sale. Por ejemplo:

```
$( "#p1" ).hover(  
    function() {alert("has entrado en p1!");},  
    function() {alert("has salido de p1!");}  
);
```

focus(): la función asociada se ejecuta cuando el campo del formulario obtiene el foco. Por ejemplo:

```
$( "input" ).focus(function() {  
    $(this).css("background-color", "#cccccc");  
});
```


blur(): la función asociada se ejecuta cuando el campo del formulario pierde el foco. Por ejemplo:

```
$( "input" ).blur( function() {  
    $(this).css("background-color", "#ffffff");  
});
```

Además de los comentados en esta sección existen más eventos que se usan de modo similar a los presentados: *keypress*, *keydown*, *keyup*...

Objeto *Event* en jQuery

Todas las funciones de respuesta a eventos reciben un argumento con múltiples campos que proporcionan información acerca del evento (es un envoltorio del evento original de JS):

```
$( "p" ).dblclick( function( evento ) {  
    $(this).hide();  
    alert ( "coordenadas: " + evento.pageX + "-" + evento.pageY );  
});
```

Algunos campos interesantes de los objetos de eventos son (dependiendo del evento, algunas pueden no recibir valor):

- **target**: elemento DOM que disparó el evento.
- **relatedTarget**: elemento en el que se entra o sale por el movimiento del cursor
- **pageX** y **pageY**: coordenadas del cursor con respecto a la esquina superior izquierda del documento).
- **which**: código de la tecla o botón de ratón pulsada en el evento. Para obtener el carácter a partir del código: `String.fromCharCode(evt.which)`
- **altKey**, **ctrlKey**, **shiftKey**, **metaKey**: teclas *Alt*, *Ctrl*, mayúsculas o *Meta* pulsadas o no.
- **data**: objeto opcional con información pasada al método del evento cuando se enlaza el manejador actual.
- **timeStamp** en milisegundos.
- **type**: texto con el nombre del evento ("click", "mouseup", etc.).

Entre los métodos del objeto Event, se dispone **preventDefault()** para evitar que el navegador responda al evento (equivalente a hacer *return false* en la función manejadora), **stopPropagation()** para evitar la propagación del evento (*bubbling*), **trigger()** para disparar un evento (cuyo nombre es pasado como argumento) de los elementos seleccionados.

```
$( "#abc" ).click( function( evento ) {  
    evento.stopPropagation();  
    // resto de instrucciones  
});
```

Métodos de enlazado de eventos

Método *on*

El modo genérico de enlazar una función manejadora a uno o más eventos es por medio del método *on()*. Con este método se pueden capturar todos los eventos DOM (de window, de formulario, de teclado, etc.) Este método sustituye al desaconsejado método *bind*.

```
$(document).ready(function(){ // dos eventos
  $("p").on("mouseover mouseout",function(){
    $("p").toggleClass("intro");
  });
});
```

Sintaxis del método *on()*: `$(selector).on(evento, selectorHijos, info, funcion, mapa)`

- **selectorHijos**: el manejador se aplica a los hijos especificados por este selector (opcional)
- **info**: información adicional que se le pasa a la función manejadora del evento (opcional)
- **mapa**: permite indicar uno o más eventos y sus funciones manejadoras, que se asociarán al elemento seleccionado. Formato: {evento:funcion, evento:funcion, ... } (opcional)

```
$("#div1").on("click", "p", function(){ // ejemplo de argumento selector de hijos
  // se aplica a los <p> interiores a #div1:
  $(this).css("background-color", "pink");
});
```

```
$("p").on({ // ejemplo de mapa
  mouseover:function(){ $("body").css("background-color", "lightgray");},
  mouseout:function(){ $("body").css("background-color", "lightblue");},
  click:function(){ $("body").css("background-color", "yellow");}
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <script
      src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
    </script>
    <script>
      function manejador(evento) { // ejemplo de argumento "mapa"
        alert(evento.data.mensaje);
      }
      $(document).ready(function(){
        $("p").on("click", {mensaje: "¡clic!"}, manejador)
      });
    </script>
  </head>
  <body>
    <p>Haz clic aquí</p>
  </body>
</html>
```

Método *off*

Para desligar las funciones manejadoras de los eventos, se debe usar el método *off()*. Este método sustituye al desaconsejado método *unbind()*.

```
$("#button").click(function() {  
    $("#p").off("click");  
});
```

Método *one*

Si se quiere especificar una función de respuesta a eventos de un solo uso, se usa el método *one()*:

```
$("#p").one("click", function() { // clic en un párrafo ejecuta la animación  
    $(this).animate({fontSize:"+=6px"}); // sólo se ejecutará una vez para párrafo  
});
```

Efectos

hide() y show()

Ocultan y muestran elementos. La distribución del resto de los elementos se modifica, ya que estos métodos ocupan o liberan el espacio de los elementos afectados. Si se quiere evitar esta redistribución, habría que usar `$(selector).css('visibility', 'hidden');`

Sintaxis: `$(selector).hide(velocidad, respuesta);`
`$(selector).show(velocidad, respuesta);`

Parámetros opcionales:

- **velocidad:** velocidad de ocultación o muestra. Puede tomar los valores "slow", "fast", o *milisegundos*
- **respuesta:** función que se ejecutará tras completar el método *hide()* o *show()*

Ejemplos

```
$("#hide").click(function() {  
    $("#p").hide();  
});
```

```
$("#show").click(function() {  
    $("#p").show();  
});
```

```
$("#button").click(function() {  
    $("#p").hide(1000);  
});
```

Fundidos (fade)

Con jQuery se puede conseguir que los elementos se muestren y desaparezcan de forma gradual (*fade in* y *fade out*). Para ello se dispone de cuatro métodos: `fadeIn()`, `fadeOut()`, `fadeToggle()` y `fadeTo()`. La distribución del resto de los elementos se ve modificada, ya que estos métodos ocupan o liberan el espacio de los elementos afectados.

fadeIn()

Muestra gradualmente un elemento oculto.

Sintaxis: `$(selector).fadeIn(velocidad, respuesta);`

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("button").click(function() {
        $("#div1").fadeIn();
        $("#div2").fadeIn("slow");
        $("#div3").fadeIn(3000);
      });
    });
  </script>
</head>
<body>
  <p>Demostración de fadeIn() con distintos parámetros.</p>
  <button>Haz click</button><br><br>
  <div id="div1" style="width:80px;height:80px;display:none;
    background-color:red;"></div><br>
  <div id="div2" style="width:80px;height:80px;display:none;
    background-color:green;"></div><br>
  <div id="div3" style="width:80px;height:80px;display:none;
    background-color:blue;"></div>
</body>
</html>
```

fadeOut()

Oculto gradualmente un elemento visible.

Sintaxis: `$(selector).fadeOut(velocidad, respuesta);`

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script> $(document).ready(function() {
                                $("button").click(function() {
                                    $("#div1").fadeOut();
                                    $("#div2").fadeOut("slow");
                                    $("#div3").fadeOut(3000);
                                });
  </script>
</head>
<body>
  <p> Demostración de fadeOut () con distintos parámetros.</p>
  <button> Haz click</button><br><br>
  <div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
  <div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
  <div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
</body>
</html>
```

Para conseguir que se desvanecieran pero ocuparan espacio, las funciones anteriores quedarían así:

```
<script>
  function ocupa() {
    $(this).css('display', 'block').css('visibility', 'hidden');
  }
  $(document).ready(function() {
    $("button").click(function() {
      $("#div1").fadeOut(ocupa);
      $("#div2").fadeOut("slow", ocupa);
      $("#div3").fadeOut(3000, ocupa);
    });
  });
</script>
```

fadeToggle()

Alterna los comportamientos entre fadeIn() y fadeOut(): los elementos ocultos los muestra como fadeIn() y los visibles los oculta como fadeOut().

Sintaxis: \$(selector).**fadeToggle**(velocidad, respuesta);

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
$(document).ready(function() {
    $("button").click(function() {
        $("#div1").fadeToggle();
        $("#div2").fadeToggle("slow");
        $("#div3").fadeToggle(3000);
    });
});
```

fadeTo()

Este método hace que un elemento visible tenga un determinado nivel de opacidad.

Sintaxis: `$(selector).fadeTo(velocidad, opacidad, respuesta);`

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *opacidad*: nivel de opacidad de 0 (oculto/transparente) a 1 (sin transparencia)
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
    </script>
    <script> $(document).ready(function() {
        $("button").click(function() {
            $("#div1").fadeTo("slow", 0.15);
            $("#div2").fadeTo("slow", 0.4);
            $("#div3").fadeTo("slow", 0.7);
        });
    });
</script>
</head>
<body>
    <p>Demostración de fadeTo() con distintos parámetros.</p>
    <button>Haz click</button><br><br>
    <div id="div1" style="width:80px;height:80px;background-color:red;"></div><br>
    <div id="div2" style="width:80px;height:80px;background-color:green;"></div><br>
    <div id="div3" style="width:80px;height:80px;background-color:blue;"></div>
</body>
</html>
```

Deslizamientos (*slides*)

jQuery permite deslizar elementos hacia arriba (mostrar) o hacia abajo (ocultar). Para ello, dispone de los métodos: `slideDown()`, `slideUp()` y `slideToggle()`. La distribución del resto de los elementos se ve modificada, ya que estos métodos ocupan o liberan el espacio de los elementos afectados.

slideDown()

Desliza un elemento hacia abajo (muestra).

Sintaxis: `$(selector).slideDown(velocidad, respuesta);`

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("#flip").click(function() {
        $("#panel").slideDown("slow");
      });
    });
  </script>

  <style type="text/css">
    #panel, #flip {
      padding: 5px;
      text-align: center;
      background-color: #e5eccc;
      border: solid 1px #c3c3c3;
    }
    #panel {
      padding: 50px;
      display: none;
    }
  </style>
</head>
<body>
  <div id="flip">Haz click</div>
  <div id="panel">HOLA</div>
</body>
</html>
```

slideUp()

Deplaza hacia arriba un elemento (oculta).

Sintaxis: `$(selector).slideUp(velocidad, respuesta);`

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("#flip").click(function() {
        $("#panel").slideUp("slow");
      });
    });
  </script>
  <style type="text/css">
    #panel, #flip {
      padding: 5px;
      text-align: center;
      background-color: #e5eccc;
      border: solid 1px #c3c3c3;
    }
    #panel { padding: 50px; }
  </style>
</head>
<body>
  <div id="flip">Haz clic</div>
  <div id="panel">HOLA</div>
</body>
</html>
```

slideToggle()

Alterna su comportamiento entre slideDown() y slideUp(): si el elemento ha sido deslizado hacia abajo, se desliza hacia arriba y viceversa.

Sintaxis: \$(selector).**slideToggle**(velocidad, respuesta);

Argumentos opcionales:

- *velocidad*: duración del efecto. Puede tomar los valores "slow", "fast", o indicar milisegundos
- *respuesta*: función que se ejecutará tras finalizar el efecto.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("#flip").click(function() {
        $("#panel").slideToggle("slow");
      });
    });
  </script>
```



```

<style type="text/css">
    #panel,#flip {
        padding:5px;
        text-align:center;
        background-color:#e5eccc;
        border:solid 1px #c3c3c3;
    }
    #panel {
        padding:50px;
        display:none;
    }
</style>
</head>
<body>
    <div id="flip">Haz click</div>
    <div id="panel">HOLA</div>
</body>
</html>

```

Animaciones

animate()

Permite crear animaciones a medida.

Sintaxis: `$(selector).animate({parametros}, velocidad, respuesta);`

Parámetros:

- *parámetros*: parámetros obligatorios que definen las propiedades CSS que se animarán.
- *velocidad*: parámetro opcional que especifica la duración del efecto.
- *respuesta*: función que se ejecutará tras completarse la animación.

Ejemplo:

Mover un element `<div>` a la izquierda hasta que la propiedad *left* haya alcanzado un valor de 250px:

```

$("button").click(function() {
    $("div").animate({left:'250px'});
});

```

Por defecto, todos los elementos HTML tienen una posición estática y no se pueden mover. Para manipular la posición, se debe establecer primero la propiedad de CSS *position* del elemento a *relative*, *fixe*, o *absolute*.

Es posible animar **múltiples propiedades** a la vez:

```

$("button").click(function() {
    $("div").animate({left:'250px',
                      opacity:'0.5',
                      height:'150px',
                      width:'150px'
    });
});

```

Con el método *animate()* es posible manipular casi todas las propiedades CSS. Sin embargo, se debe recordar que todos los nombres de las propiedades deben estar en notación "camello" cuando se usen con este método: *paddingLeft* en lugar de *padding-left*, por ejemplo.

El método *animate()* sólo puede trabajar con propiedades numéricas. Por ese motivo, la animación del color no está incluido en la librería básica de jQuery. Para poder animar colores es necesario instalar el plugin <https://github.com/jquery/jquery-color>

También es posible definir **valores relativos** (el valor será entonces, relativo al valor actual del elemento). Esto se consigue añadiendo += o -= delante del valor.

Ejemplo:

```
$( "button" ).click(function() {  
    $( "div" ).animate({  
        left: '250px',  
        height: '+=150px',  
        width: '+=150px'  
    });  
});
```

También es posible especificar **valores de propiedad de animación** como "show", "hide", o "toggle":

```
$( "button" ).click(function() {  
    $( "div" ).animate({  
        height: 'toggle' // 0 ó valor inicial  
    });  
});
```

Si se escriben múltiples llamadas consecutivas a *animate()*, jQuery crea una cola interna con estas llamadas, que se ejecutarán una por una.

Ejemplos

```
$( "button" ).click(function() {  
    var div=$( "div" );  
    div.animate ({height: '300px', opacity: '0.4'}, "slow");  
    div.animate ({width: '300px', opacity: '0.8'}, "slow");  
    div.animate ({height: '100px', opacity: '0.4'}, "slow");  
    div.animate ({width: '100px', opacity: '0.8'}, "slow");  
});
```

Mover el elemento <div> a la derecha y después incrementar el tamaño del texto.

```
$( "button" ).click(function() {  
    var div=$( "div" );  
    div.animate ({left: '100px'}, "slow");  
    div.animate ({fontSize: '3em'}, "slow");  
});
```

stop()

Se usa para detener una animación o efecto antes de su final. Funciona con todas las funciones de efecto, incluyendo deslizamientos (*sliding*), desvanecimientos (*fading*) y animaciones a medida (*custom animations*)

Sintaxis: `$(selector).stop(detenerTodos, irAlFinal);`

Parámetros opcionales:

- *detenerTodos*: especifica si también se debe limpiar la cola de ejecución o no. Por defecto, su valor es *false*, lo que significa que solo la animación activa se detendrá, permitiendo que las demás animaciones pendientes se ejecuten a continuación.
- *irAlFinal*: indica si se debe completar la animación actual de modo instantáneo. Por defecto vale *false*.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script> $(document).ready(function() {
    $("#start").click(function() {
      $("#div").animate({left: '100px'}, 5000);
      $("#div").animate({fontSize: '3em'}, 5000);
    });
    $("#stop").click(function() {      $("#div").stop();                });
    $("#stop2").click(function() {    $("#div").stop(true);          });
    $("#stop3").click(function() {    $("#div").stop(true, true);    });
  });
</script>
</head>
<body>
  <button id="start">Iniciar</button>
  <button id="stop">Parar</button>
  <button id="stop2">Parar todo</button>
  <button id="stop3">Parar al terminar</button>
  <p>El botón "Iniciar" comienza la animación.</p>
  <p>El botón "Parar" detiene la animación en curso pero permite que
    continúen las siguientes de la cola.</p>
  <p>El botón "Parar todo" detiene la animación en curso y
    vacía la cola de animación, de modo que se detienen todas
    las animaciones sobre el elemento.</p>
  <p>"Parar al terminar" termina la animación en curso y luego finaliza.</p>
  <div style="background:#98bf21;height:100px;width:200px;position:absolute;">
    HOLA
  </div>
</body>
</html>
```

Funciones de respuesta a la finalización de eventos (*callback*)

Las sentencias JavaScript se ejecutan línea por línea. Sin embargo, con los efectos, la siguiente línea de código puede ser ejecutada incluso si el efecto aún no ha terminado. Esto puede ser problemático. Para evitarlo, podemos crear una función de respuesta (*callback*) que se ejecute tras finalizar el efecto.

Sintaxis típica: `$(selector).hide(velocidad, respuesta);`

Ejemplo con respuesta:

```
$("#button").click(function() {  
    $("#p").hide("slow", function() {  
        alert("El párrafo está oculto");  
    });  
});
```

Ejemplo sin respuesta

La ventana de alerta se mostrará antes de que termine la ocultación:

```
$("#button").click(function() {  
    $("#p").hide(1000);  
    alert("El párrafo está oculto");  
});
```

Encadenamiento de métodos.

Hasta ahora hemos escrito siempre las sentencias jQuery una tras otra. Sin embargo, hay una técnica llamada encadenamiento que nos permite ejecutar multiples comandos jQuery, uno tras otro sobre los mismos elementos. De este modo, los navegadores solo tienen que localizar los elementos una vez. Cada acción espera hasta que la anterior haya acabado.

Para encadenar una acción, simplemente se añade a la acción previa. Por ejemplo:

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

Si la línea resultante es muy larga, podemos incluir cambios de línea e indentaciones sin problema:

```
$("#p1").css("color", "red")  
    .slideUp(2000)  
    .slideDown(2000);
```

Iteración implícita

Si un selector devuelve varias coincidencias, las funciones que se apliquen sobre aquel, se aplicarán sobre cada uno de los elementos descritos por el selector:

Por ejemplo, la sentencia `$('img').hide()` ocultará todas las imágenes.

El método *each* define una función que se aplicará a cada elemento de la iteración. El elemento concreto al que se aplica la función está referenciado por el selector *this*: `$(this)`

```
$("#li").each(function() {  
    alert($(this).text())  
});
```

La función suministrada a *each* puede tener dos parámetros. El primero indica la posición (índice que comienza en 0) del elemento en la iteración y el segundo, la referencia al elemento procesado (equivalente al `$(this)`)

```
$("#li").each(function(indice, elemento) {  
    alert(indice + " -> " + $(elemento).text()); // posición y texto de cada <li>  
})
```

Manipulación del DOM con jQuery

jQuery proporciona métodos que simplifican el acceso y manipulación de elementos y atributos del DOM.

text(), html() y val()

- **text()**: asigna o devuelve el contenido de texto de los objetos seleccionados sin las etiquetas HTML.
- **html()**: asigna o devuelve el contenido de los objetos seleccionados, incluso con las etiquetas HTML. No devuelven los elementos internos, sólo el texto (*innerHTML*)
- **val()** - Establece o devuelve el valor de campos del formulario.

Ejemplos:

```
$("#btn1").click(function() {
    alert("Texto: " + $("#test").text());
});

$("#btn2").click(function() {
    alert("HTML: " + $("#test").html());
});

$("#btn1").click(function() {
    alert("Valor: " + $("#test").val());
});

$("#btn1").click(function() {
    $("#test1").text(";Hola mundo!");
});

$("#btn2").click(function() {
    $("#test2").html("<b>;Hola mundo!</b>");
});

$("#btn3").click(function() {
    $("#test3").val("Hola");
});
```

attr()

Permite establecer, asignar y leer valores de atributos.

Ejemplos

```
$("button").click(function() {
    alert($("#cursos").attr("href"));
});

$("button").click(function() {
    $("#cursos").attr({
        "href" : "http://momentum.uvigo.es ",
        "title" : "Cursos de programación"
    });
});
```

Los métodos *text()*, *html()* y *val()* también pueden recibir como argumento una función. Lo que devuelva la función será lo que se asigne al elemento. La función tiene dos parámetros: el índice del elemento actualmente recorrido en la lista de elementos seleccionados y el valor original.

De modo similar, el método *attr()* también puede recibir una función con dos argumentos: el índice del elemento actual en la lista de elementos seleccionados y el valor original del atributo. Igualmente, la *string* devuelta por la función será el nuevo valor del atributo.

Ejemplos

```
$("#btn1").click(function(){
    $("#test1").text(function(i, origText){
        return "Texto viejo: " + origText +
            " Texto nuevo: Hola a todos (índice: " + i + ")";
    });
});

$("#btn2").click(function(){
    $("#test2").html(function(i, origText){
        return "HTML viejo: " + origText + " HTML nuevo: Hola <b>a todos</b>
        (index: " + i + ")";
    });
});

$("#button").click(function(){
    $("#momentum").attr("href", "http://momentum.uvigo.es");
});

$("#button").click(function(){
    $("#w3s").attr("href", function(i, origValue){
        return origValue + "/jquery";
    });
});
```

Añadir nuevo contenido HTML

Los siguientes métodos permiten añadir nuevo contenido:

- **append()**: Inserta contenido al final de los elementos seleccionados
- **prepend()**: Inserta contenido al principio de los elementos seleccionados
- **after()**: Inserta contenido tras los elementos seleccionados
- **before()**: Inserta contenido antes de los elementos seleccionados

append()

Añade contenido dentro del elemento seleccionado, tras el último contenido:

```
$("#p").append("texto añadido al final");
```

prepend()

Añade contenido dentro del elemento seleccionado, antes del primer contenido:

```
$("#p").prepend("texto añadido al principio");
```

after()

Inserta contenido tras los elementos HTML seleccionados (no se modifican sus contenidos):

```
$("#img").after("texto añadido tras el elemento");
```

before()

Inserta contenido ante los elementos HTML seleccionados (no se modifican sus contenidos):

```
$("#img").before("texto añadido antes del elemento");
```

Los métodos *append()*, *prepend()*, *after()* y *before()* pueden tomar un número ilimitado de elementos nuevos como parámetros. Estos nuevos elementos se pueden generar con texto/HTML (como se ha hecho en los ejemplos anteriores), con jQuery o con código JavaScript y elementos DOM.

Ejemplos

```
function appendText() {
    var txt1="<p>Texto.</p>"; // Crea elemento con HTML
    var txt2=$("#<p></p>").text("Texto."); // Crea con jQuery
    var txt3=document.createElement("p"); // Crea con DOM
    txt3.innerHTML="Texto.";
    $("#p").append(txt1,txt2,txt3); // Añade los elementos nuevos
}
function afterText() {
    var txt1="<b>Me </b>"; // Crea elemento con HTML
    var txt2=$("#<i></i>").text("gusta "); // Crea con jQuery
    var txt3=document.createElement("big"); // Crea con DOM
    txt3.innerHTML="jQuery!";
    $("#img").after(txt1,txt2,txt3); // Inserta los elementos tras los img
}
```

clone()

Clona un elemento:

```
$("#div1 span").clone();
```

insertAfter(), insertBefore()

Insertan elementos después o antes:

```
$("#div1 span").clone().insertBefore("#div2");
$("#div1 span").clone().insertAfter("#div2");
```

appendTo()

Insertan elementos después o antes:

```
$("#div1").appendTo("#div2");
```

Existen más funciones para añadir o mover nodos en el DOM, como por ejemplo *replaceWith()*, *wrap()*, etc.

Eliminación de elementos y contenido

Para eliminar elementos y contenido hay, principalmente dos métodos: `remove()` y `empty()`.

`remove()`

Elimina los elementos seleccionados y sus elementos "hijo":

```
$("#div1").remove();
```

Este método también acepta un parámetro que permite filtrar los elementos a eliminar. Este parámetro puede ser cualquier selector jQuery. Por ejemplo, para eliminar todos los elementos `<p>` con `class="italic"`:

```
$("p").remove(".italic");
```

`empty()`

Elimina los elementos "hijo" de los elementos seleccionados:

```
$("#div1").empty();
```

`removeAttr()`

Elimina el atributo indicado:

```
$("#div1").removeAttr('height');
```

Manipulación de CSS

Algunos de los métodos disponibles para la manipulación de CSS (CSS3 incluido) son los siguientes:

- `addClass()`: añade una o más clases a los elementos seleccionados
- `removeClass()`: elimina una o más clases de los elementos seleccionados
- `toggleClass()`: alterna entre añadir y eliminar clases de los elementos seleccionados
- `css()`: asigna o devuelve el atributo de estilo ("style")

La siguiente hoja de estilo se usará para todos los ejemplos de este apartado:

```
.importante {
    font-weight:bold;
    font-size:xx-large;
}
.azul {
    color:blue;
}
```

`addClass()`

```
$("button").click(function() {
    $("h1,h2,p").addClass("azul");
    $("div").addClass("importante");
});
```



```
$("button").click(function() {  
    $("#div1").addClass("importante azul");  
});
```

removeClass()

```
$("button").click(function() {  
    $("h1,h2,p").removeClass("azul");  
});
```

toggleClass()

```
$("button").click(function() {  
    $("h1,h2,p").toggleClass("azul");  
});
```

Devolver el valor de una propiedad CSS

Sintaxis: `css("nombreDeLaPropiedad");`

El siguiente ejemplo devolverá el valor del **primer** elemento que encaje con el selector:

```
$("p").css("background-color");
```

Asignar valor a una propiedad CSS:

Sintaxis: `css("nombreDeLaPropiedad ","valor");`

El siguiente ejemplo dará valor a todos los elementos que encajen con el selector:

```
$("p").css("background-color", "yellow");
```

Asignar valor a multiples propiedades CSS

Sintaxis: `css({"nombreDeLaPropiedad ":"valor"," nombreDeLaPropiedad ":"valor",...});`

El siguiente ejemplo establecerá el valor del color de fondo y el tamaño de fuente para todos los elementos que encajen con el selector:

```
$("p").css({"background-color":"yellow","font-size":"200%"});
```

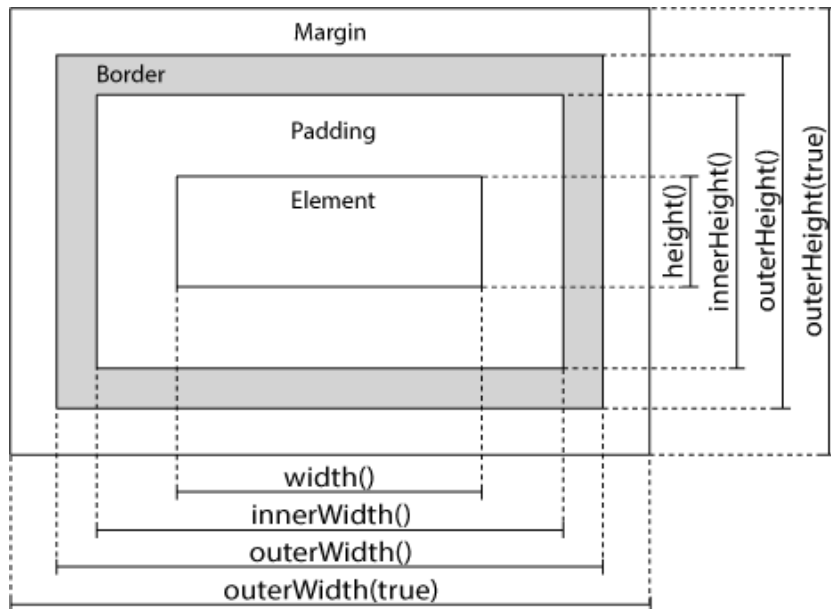
Este método también funciona con propiedades CSS3.

Métodos de dimensiones

jQuery dispone de métodos para trabajar con dimensiones: `width()`, `height()`, `innerWidth()`, `innerHeight()`, `outerWidth()` y `outerHeight()`.

width() y height()

Asignan o devuelven el ancho y el alto respectivamente, de un elemento (no incluyen "padding", ni "border" ni "margin").



```

$("button").click(function() {
    var txt="";
    txt+="Width: " + $("#div1").width() + "<br>";
    txt+="Height: " + $("#div1").height();
    $("#div1").html(txt);
});

$("button").click(function() {
    $("#div1").width(500).height(500);
});

```

El siguiente ejemplo incluye la ventana del navegador:

```

$("button").click(function() {
    var txt="";
    txt+="Ancho y alto del documento: " + $(document).width();
    txt+="x" + $(document).height() + "\n";
    txt+="Ancho y alto de la ventana: " + $(window).width();
    txt+="x" + $(window).height();
    alert(txt);
});

```

offset() y position()

El método *offset()* devuelve la posición de un elemento con relación al documento, por medio de un objeto con campos *top* y *left*. No funciona sobre elementos ocultos y no tiene en cuenta bordes, rellenos o márgenes.

Si se le pasa un argumento que sea un objeto con atributos *top* y *left* (desplazamientos con respecto al documento), cambiará la ubicación de los elementos. Si el posicionamiento del elemento era estático, pasará a ser relativo.

```

$("p:last").offset({ top: 10, left: 30 });

```

El método *position()* no acepta argumentos y funciona de modo parecido a *offset()*: devuelve las coordenadas del elemento con respecto a su elemento padre. Esta información la devuelve como un

objeto con los campos *top* y *left*. Al igual que el método *offset*, no funciona sobre elementos ocultos y no tiene en cuenta bordes, rellenos o márgenes.

innerWidth() e innerHeight()

Devuelven el ancho y el alto respectivamente de un elemento, incluyendo el relleno (*padding*).

```
$("#button").click(function() {
    var txt="";
    txt+="ancho interior: " + $("#div1").innerWidth() + "<br>";
    txt+="alto interior: " + $("#div1").innerHeight();
    $("#div1").html(txt);
});
```

outerWidth() y outerHeight()

Devuelven el ancho y el alto respectivamente de un elemento, incluyendo el "padding" y el "border"

```
$("#button").click(function() {
    var txt="";
    txt+="Ancho y alto de: " + $("#div1").outerWidth() + "<br>";
    txt+="ancho exterior: " + $("#div1").outerHeight();
    $("#div1").html(txt);
});
```

Si les pasamos un parámetro con valor *true*, también incluirán el "margin":

```
$("#button").click(function() {
    var txt="";
    txt+="ancho exterior (+margen): " + $("#div1").outerWidth(true) + "<br>";
    txt+="alto exterior (+margen): " + $("#div1").outerHeight(true);
    $("#div1").html(txt);
});
```

Recorridos (*traversing*)

jQuery dispone de multitud de funciones para seleccionar elementos HTML basándose en su relación con otros elementos. Para ellos se utilizan los conceptos de padre, ancestro, hijo, hermano o descendiente.

Ascestros

parent()

Devuelve el padre del elemento seleccionado. El siguiente ejemplo muestra un borde rojo alrededor del padre de cada elemento *span*:

```
$("#span").parent().css({"color":"red","border":"2px solid red"});
```

parents()

Devuelve todos los ancestros de los elementos seleccionados. El siguiente ejemplo muestra un borde rojo alrededor de cada ancestro de cada elemento *span*:

```

<script>
    $(document).ready(function() {
        $(".ex .hide").click(function() {$(this).parents(".ex").hide("slow");});
    });
</script>
<style> div.ex {
    background-color:#e5eecd;
    padding:7px;
    border:solid 1px #c3c3c3;
}
</style>

```

parentsUntil()

Devuelve todos los ancestros entre dos argumentos suministrados (ambos excluidos). El siguiente ejemplo devuelve todos los ancestros entre un elemento *span* y un *div*:

```

<html>
<head>
    <style>
        .ancestors * {
            display: block;
            margin: 15px;
            border: 2px solid lightgrey;
            color: lightgrey;
            padding: 5px;
        }
    </style>
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
    </script>
    <script>
        $(document).ready(function() {
            $("span").parentsUntil("div").css({"color":"red","border":"2px solid red"});
        });
    </script>
</head>
<body class="ancestors"> body (tatarabuelo)
    <div style="width:500px;">div (bisabuelo)
        <ul>ul (abuelo)
            <li>li (padre)
                <span>span</span>
            </li>
        </ul>
    </div>
</body>
</html>

```

body (tatarabuelo)



Como se ve en este último ejemplo, la mayoría de los métodos de recorridos permiten proporcionar un argumento de filtrado.

Descendientes

children()

Devuelve los hijos (ni nietos, ni bisnietos, etc.) del elemento seleccionado. El siguiente ejemplo muestra un borde rojo alrededor de cada hijo de cada elemento *div*:

```
$("div").children().css({"color":"red","border":"2px solid red"});
```

Es posible añadir un parámetro para filtrar los hijos. El siguiente ejemplo muestra un borde rojo alrededor de cada hijo de cada elemento *div*, a condición de que dicho hijo sea un elemento *p* de la clase "a":

```
$("div").children("p.a").css({"color":"red","border":"2px solid red"});
```

find()

Este método actúa del mismo modo que *children()*, pero devolviendo todos los descendientes, en lugar de sólo los hijos. También puede tener un argumento de filtrado.

Hermanos

siblings()

Devuelve todos los hermanos de los elementos seleccionados. Se le puede pasar un parámetro de filtrado. Por ejemplo, los hermanos de cada *h2* que sean elementos *p*:

```
$("h2").siblings("p").css({"color":"red","border":"2px solid red"});
```

next()

Devuelve el siguiente elemento de cada uno de los seleccionados. En el ejemplo siguiente, si todos los hermanos son *h2* se cambiaría el estilo de todos excepto el primero:

```
$("h2").next().css({"color":"red","border":"2px solid red"});
```

nextAll()

Devuelve todos los hermanos de los elementos seleccionados a condición de que aparezcan a continuación. Por ejemplo:

```
$("h2").nextAll().css({"color":"red","border":"2px solid red"});
```

nextUntil()

Similar al anterior, pero sólo se devuelven hasta alcanzar un elemento seleccionado por el argumento (este último excluido). En el siguiente ejemplo, sólo se devolverán los hermanos de los *h2* que estén a continuación de estos y hasta alcanzar el primer *h6*.

```
$("h2").nextUntil("h6").css({"color":"red","border":"2px solid red"});
```

prev(), prevAll() y prevUntil()

Funcionan de modo similar a los tres anteriores, pero devolviendo los elementos previos en lugar de los siguientes.

Filtrado

Permite acceder a elementos en función de su posición en un grupo.

first()

Devuelve el primero de los elementos seleccionados. En el siguiente ejemplo se cambia a amarillo el color de fondo del primer elemento *p* que esté dentro de un elemento *div*:

```
<!DOCTYPE html>
```

```
<html>
<head>
  <meta charset="UTF-8">
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.2.0/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("div p").first().css("background-color", "yellow");
    });
  </script>
</head>
<body>
  <p>Primer párrafo en el body </p>
  <div style="border: 1px solid black;">
    <p>Primer párrafo en un div</p>
    <p>Último párrafo en un div</p>
  </div><br>
  <div style="border: 1px solid black;">
    <p>Primer párrafo en otro div</p>
    <p>Último párrafo en otro div</p>
  </div>
  <p>Último párrafo en el body</p>
</body>
</html>
```

Primer párrafo en el body

Primer párrafo en un div

Último párrafo en un div

Primer párrafo en otro div

Último párrafo en otro div

Último párrafo en el body

last()

Actúa de forma similar al anterior, pero devolviendo el último en lugar del primer elemento seleccionado.

eq()

Devuelve el elemento de la selección que ocupa la posición indicada por el argumento entero suministrado. La primera posición corresponde al índice 0.

Primer párrafo en el body

Primer párrafo en un div

Último párrafo en un div

Primer párrafo en otro div

Último párrafo en otro div

Último párrafo en el body

Si en el ejemplo del método *first()* se sustituyera el selector *first* por *eq(1)*, el resultado sería el mostrado en la figura:

```
$(document).ready(function() {  
    $("div p").eq(1).css("background-color", "yellow");  
});
```

filter()

Este método permite especificar un criterio de modo que los que no lo cumplan, se eliminan de la selección. El siguiente ejemplo mostrará con fondo amarillo todos los elementos *p* que sean de la clase "intro":

```
$("p").filter(".intro").css("background-color", "yellow");
```

not()

Este método es similar a *filter()* pero eliminando los elementos de la selección que cumplen el criterio.

Otros métodos de recorrido del DOM

Además de los ya tratados, se dispone de más métodos para el recorrido del DOM: *add()*, *addBack()*, *closest()*, *contents()*, *each()*, *end()*, *has()*, *is()*, *map()*, *offsetParent()* y *slice()*.

AJAX

AJAX (*Asynchronous JavaScript And XML*), permite la descarga y visualización de información sin la necesidad de recargar toda la página. Gmail, Google Maps, Youtube y Facebook son ejemplos de uso de AJAX.

jQuery simplifica el uso de AJAX proporcionándonos varios métodos. Con ellos, podremos obtener los datos de un servidor remoto en distintos formatos (texto, HTML, XML o JSON) usando los comandos GET y POST de HTTP, y guardar la información externa directamente en los elementos HTML de la página seleccionados. Se puede hacer la solicitud a un servidor de destino distinto al de la página HTML, siempre que lo permita de forma explícita (con la directiva *Access-Control-Allow-Origin*, por ejemplo).

load()

Carga datos de un servidor y los guarda en el elemento HTML seleccionado. Esta última característica lo diferencia de los métodos *get()* y *load()* que se verán a continuación. Usará el método POST si se le pasa un argumento de datos y GET en caso contrario.

Sintaxis: `$(selector).load(URL, datos, respuesta);`

Argumentos:

- *URL*: es obligatorio y contendrá la ubicación duración del elemento a descargar.
- *datos*: es opcional y especifica un conjunto de pares clave/valor para enviar con la solicitud
- *respuesta*: función que se ejecutará tras finalizar la descarga.

```

<!DOCTYPE html>
<html>
<head>
  <script src="//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js">
  </script>
  <script>
    $(document).ready(function() {
      $("button").click(function() {
        $("#div1").load("demo_test.txt");
      });
    });
  </script>
</head>

<body>
  <div id="div1"><h2>jQuery con AJAX cambia este texto</h2></div>
  <button>descarga contenido externo</button>
</body>
</html>

```

También es posible añadir un selector jQuery al parámetro URL. Por ejemplo, el siguiente código carga el contenido del elemento con id="p1" del fichero "demo_test.txt", en un elemento <div> concreto: `$("#div1").load("demo_test.txt #p1");`

El parámetro opcional *respuesta* puede tener a su vez, tres parámetros distintos:

- **respuesta:** contenido resultante si se ejecuta con éxito (texto)
- **estado:** texto con los valores: *success*, *notmodified*, *error*, *timeout* o *parsererror*
- **xhr:** contiene el objeto XMLHttpRequest

Ejemplo:

Muestra una ventana de alerta tras completarse el método load(). Si éste ha tenido éxito, muestra contenido descargado. En caso contrario, muestra un mensaje de error:

```

$("button").click(function() {
  $("#div1").load("demo_test.txt", function(respuesta, estado, xhr) {
    if(estado=="success")
      alert("contenido descargado");
    if(estado=="error")
      alert("Error: "+xhr.status+": "+xhr.statusText);
  });
});

```

El segundo parámetro, permite de forma opcional suministrar información con la llamada, en forma de objeto:

```

$( "#feeds" ).load( "feeds.php", { limite: 25 }, function() {
  alert( "Se han cargado las últimas 25 entradas del feed" );
});

```


Solicitudes HTTP: Get y Post

Cuando un cliente solicita información a un servidor, las dos órdenes más habituales son GET y POST.

GET se usa básicamente para obtener simplemente información desde el servidor. Hay que tener en cuenta que esta información puede provenir de una memoria cache. POST nunca devuelve información guardada en memoria cache y se suele usar para enviar datos junto con la petición.

`$.get()`

Pide información del servidor por medio de una solicitud GET

Sintaxis: `$.get(URL, datos, respuesta, tipo);`

Argumentos:

- *URL*: es obligatorio y contendrá la ubicación duración del elemento a descargar
- *datos*: información adicional con el mismo formato que en el método *load()*
- *respuesta*: función opcional que se ejecutará si la solicitud tiene éxito. Igual a la del método *load()*.
- *tipo*: texto opcional que indica el tipo de datos que se espera recibir (*xml*, *html*, *text*, *script*, *json* o *jsonp*)

Ejemplo:

```
$.get("test.php", { 'colores[]' : ["Rojo","Verde","Azul"] }); // datos:n vector
$("#button").click(function() {
    $.get("test.php",function(datos, estado) {
        alert("Datos: " + datos + "\nEstado: " + estado);
    });
});
```

Es posible asociar una función respuesta ante errores:

```
$.get(url, datos, funcionOK).error(funcionError):
```

`$.post()`

Pide información del servidor por medio de una solicitud POST. Los argumentos y la respuesta son idénticos a los del método *get()*.

Sintaxis: `$.post(URL, datos, respuesta);`

Argumentos:

- *URL*: es obligatorio y contendrá la ubicación duración del elemento a descargar
- *datos*: información adicional con el mismo formato que en el método *load()*
- *respuesta*: función opcional que se ejecutará si la solicitud tiene éxito. Igual a la del método *load()*.
- *tipo*: texto opcional que indica el tipo de datos que se espera recibir (*xml*, *html*, *text*, *script*, *json* o *jsonp*)

Resumen de métodos AJAX en jQuery

Además de los métodos expuestos de AJAX, jQuery proporciona los siguientes:

<code>\$.ajax()</code>	Petición AJAX más configurable que <i>get()</i> , <i>post()</i> o <i>load()</i>
<code>\$.ajaxPrefilter()</code>	Permite modificar las opciones de AJAX (configurar) para la siguiente llamada
<code>\$.ajaxSetup()</code>	Establece los valores por defecto de las siguientes llamadas AJAX
<code>\$.ajaxTransport()</code>	Crea un objeto de uso excepcional para gestionar la transmisión (enviar y abortar)
<code>\$.getJSON()</code>	Carga información de un servidor en formato JSON usando una petición GET
<code>\$.getScript()</code>	Carga y ejecuta código Javascript de un servidor usando una petición GET
<code>ajaxComplete()</code>	Especifica una función que se ejecutará cuando la petición se complete
<code>ajaxError()</code>	Especifica una función que se ejecutará cuando la petición se complete con error
<code>ajaxSend()</code>	Especifica una función que se ejecutará antes de enviar la petición
<code>ajaxStart()</code>	Especifica una función que se ejecutará cuando empiece la primera petición
<code>ajaxStop()</code>	Especifica una función que se ejecutará cuando se completen todas las peticiones
<code>ajaxSuccess()</code>	Especifica una función que se ejecutará cuando la petición se complete con éxito
<code>serialize()</code>	Codifica un conjunto de elementos de formulario como una string para su envío
<code>serializeArray()</code>	Codifica un conjunto de elementos de formulario como un array de nombres/valores
<code>\$.param()</code>	Función de uso interno que crea una representación serializada de un objeto o array para (se puede usar como una string de consulta de URL en las peticiones AJAX)

Desde la versión 1.9 de jQuery, los manejadores de los eventos globales de Ajax (`ajaxComplete()`, `ajaxError()`, `ajaxSend()`, `ajaxStart()`, `ajaxStop()`, `ajaxSuccess()`, `serialize()` y `serializeArray()`) deben asociarse al documento. Por ejemplo:

```
$( document ).ajaxError(function( evento, petition, conf ) {  
    $( "#mens" ).append( "<li>Error solicitando la página " + conf.url + "</li>" );  
});
```

jQuery Mobile

Introducción

jQuery Mobile es un framework desarrollado por jQuery que combina HTML5 y jQuery para la creación de sitios web móviles. Nos permite generar aplicaciones cuya apariencia será siempre la misma independientemente del dispositivo desde el que acceda un usuario siempre que este usuario acceda desde un dispositivo que acepte HTML5.

Este framework nos provee de ciertas herramientas que nos hacen la tarea de crear una página mucho más sencilla. Con unas pocas asignaciones de atributos HTML podremos generar interfaces de altísima calidad y usabilidad.

Compatibilidad

Con jQuery Mobile se pueden crear aplicaciones para la mayoría de las plataformas móviles: iOS, Android, Blackberry, Kindle, etc.

Creación de Interfaces

jQuery Mobile nos permite crear interfaces optimizadas para dispositivos móviles muy rápidamente y con relativamente poco esfuerzo.

La forma en que trabaja, es reescribiendo el código original del documento y generando uno nuevo, más complejo, según las características y argumentos solicitados.

jQuery Mobile aprovecha al máximo los nuevos elementos de HTML5 y hace un uso intensivo de los atributos personalizados que se definen con el prefijo "data-"

Por ejemplo "data-role", uno de los atributos más usados en jQuery Mobile, permite definir el rol de funcionalidad y apariencia del elemento que lo contiene. Al definir el rol de un elemento, este atributo permite crear páginas, botones, menús y muchos elementos más.

Al insertar el atributo "data-role" en cualquier etiqueta HTML lo convertimos en un elemento de la interfaz. Sin necesidad agregar ningún código adicional jQuery agrega todos los elementos gráficos, clases y hasta animaciones necesarios para el funcionamiento de ese elemento en particular.

Páginas y cuadros de diálogo

En jQuery Mobile, una página consiste en un elemento *div* con un atributo *data-role="page"*. Dentro del contenedor de página, se puede usar cualquier etiqueta HTML, pero en las páginas típicas de jQuery Mobile, el hijo inmediato de una página son *divs* con los valores *header*, *content* y *footer* en el atributo *data-role*. El único imprescindible es el primero (*page*), el resto son opcionales.

Navegación AJAX y transiciones

jQuery Mobile también incluye un sistema de navegación AJAX como base a un conjunto de transiciones. Para ello se convierten de forma automática los enlaces estándar y los envíos de formularios en peticiones AJAX.

Siempre que se haga clic en un enlace o se envíe un formulario, el evento se intercepta automáticamente por el sistema de navegación de AJAX y se emite una petición AJAX sobre el *href* o el formulario en lugar de recargar toda la página. Mientras el *framework* espera por la respuesta AJAX se muestra una imagen de carga.

Cuando la página solicitada está cargada, jQuery Mobile busca en el documento un elemento con el atributo *data-role="page"* e inserta su código en el DOM de la página original. También se actualizará el título de la página con el de la página entrante.

Una vez que la página solicitada está en el DOM se muestra por medio de una transición. Por defecto, se aplica una transición de difuminado (*fade*). Para personalizar el efecto de transición se debe añadir el atributo *data-transition* al enlace.

Contenido y widgets

Dentro del contenedor del contenido, se puede añadir cualquier código HTML estándar (listas, párrafos, etc.) Se pueden añadir estilos personalizados para crear disposiciones (*layouts*) a medida, añadiendo una hoja de estilos adicional en el *head*, a continuación de la hoja de estilo de jQuery Mobile.

jQuery Mobile incluye una amplia variedad de widgets de interfaz de usuario: botones, collapsibles, popups, etc.

Instalación

Se puede usar una copia remota (CDN) o local de los ficheros necesarios (js y css). Una ventaja importante de usar un CND es que si el usuario ya lo descargó en cualquier navegación previa, queda guardado en su equipo no es necesaria una nueva descarga. Esta es la sintaxis para acceder al CDN:

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
        href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
  <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
  </script>
</head>
```

La etiqueta *meta* del código anterior indica que el ancho de la página debe ser igual al ancho de la pantalla (es decir, ocupar toda la pantalla) y que no debe haber *zoom* inicial (al cargar la página, *zoom = 1*)

Si se quiere trabajar con una copia local, el código a incluir en las páginas debería ser similar al siguiente:

```

<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="jquery.mobile-1.4.5.css">
  <script src="jquery.js"></script>
  <script src="jquery.mobile-1.4.5.js"></script>
</head>

```

El tamaño de los ficheros a descargar se puede reducir por medio de *jQuery Mobile Download Builder* incluyendo sólo las funcionalidades que vamos a usar.

Páginas

Página básica

En el *body*, se usa un *div* con el atributo *data-role* con valor *page* como contenedor de la página. De forma opcional, se añaden en su interior para establecer una cabecera (*data-role="header"*), un área de contenido (*data-role="main"*) y un pie de página (*data-role="footer"*). La clase *ui-content* añade relleno (*padding*) y margen (*margin*) al contenido (*class="ui-content"*)

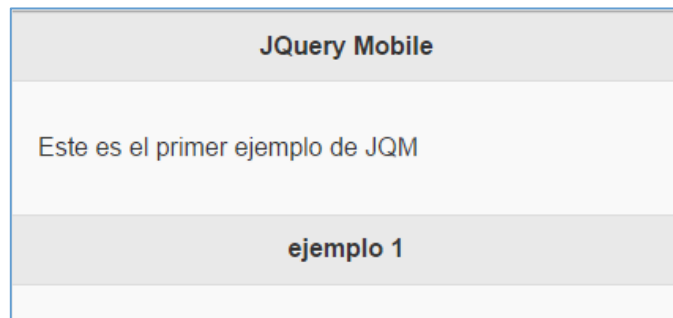
```

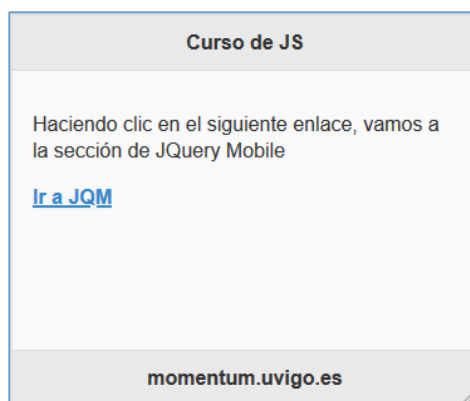
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <h1>jQuery Mobile</h1>
      </div>

      <div data-role="main" class="ui-content">
        <p>Este es el primer ejemplo de JQM</p>
      </div>

      <div data-role="footer">
        <h1>ejemplo 1</h1>
      </div>
    </div>
  </body>
</html>

```





```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
  </head>
  <body>
    <div data-role="page" id="pag1">
      <div data-role="header">
        <h1>Curso de Javascript</h1>
      </div>
      <div data-role="main" class="ui-content">
        <p>Haciendo clic en el siguiente enlace,
          vamos a la sección de JQuery Mobile</p>
        <a href="#pag2">Ir a JQM</a>
      </div>
      <div data-role="footer" data-position="fixed">
        <h1>2014</h1>
      </div>
    </div>

    <div data-role="page" id="pag2">
      <div data-role="header">
        <h1>JQuery Mobile</h1>
      </div>
      <div data-role="main" class="ui-content">
        <p>Haciendo clic en el siguiente enlace, vamos a
          la sección de Javascript</p>
        <a href="#pag1">Ir a Javascript</a>
      </div>
      <div data-role="footer" data-position="fixed">
        <h1>momentum.uvigo.es</h1>
      </div>
    </div>
  </body>
</html>
```

En un fichero HTML se pueden definir más de una página. Esto acelera las transiciones entre páginas. Para hacer referencia a páginas en ficheros HTML distintos, se usa la notación habitual de HTML:

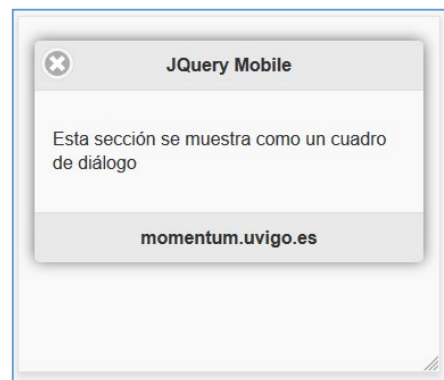
```
<a href="pagina1">Ir a Javascript</a>
<a href="http://momentum.uvigo.es/cursos/javascript.html">Ir a Javascript</a>
```

En el ejemplo anterior vemos cómo puede fijarse pie de página (`data-role="footer"`) a la parte inferior de la pantalla con el atributo `data-position="fixed"`.

Cuadros de diálogo

Para abrir una página como cuadro de diálogo, simplemente se añade el atributo `data-dialog="true"`:

```
<div data-role="page" data-dialog="true" id="pag2">
  <div data-role="header">
    <h1>jQuery Mobile</h1>
  </div>
  <div data-role="main" class="ui-content">
    <p>Esta sección se muestra como
      un cuadro de diálogo</p>
  </div>
  <div data-role="footer" data-position="fixed">
    <h1>momentum.uvigo.es</h1>
  </div>
</div>
```



Transiciones

Los efectos de transición se pueden aplicar a cualquier envío de enlace o formulario. Para establecer el tipo de transmisión se utiliza el atributo `data-transition`. Los valores posibles son *fade* (valor por defecto), *flip*, *flow*, *pop*, *slide*, *slidefade*, *slideup*, *slidedown*, *turn* o *none*.

Para especificar la dirección del efecto se usa el atributo `data-direction="reverse"`.

```
<a href="#pag2" data-transition="slide" data-direction="reverse">Deslizando</a>
```

Botones

En jQuery Mobile, los botones se pueden crear de tres formas distintas: con los elementos `<input>` o `<button>` o asignándole al atributo `class` de un enlace el valor `"ui-btn"`:

```
<a href="#pag2" class="ui-btn">Botón de JQM </a>
```

Por defecto, los botones ocupan todo el ancho disponible. Los botones de tipo `<input type="button">` se muestran sombreados con esquinas redondeadas.



Existe la posibilidad de agrupar botones, dando valor a los atributos `data-role="controlgroup"` (agrupa y elimina márgenes) y `data-type="horizontal"` (o `"vertical"`) en un elemento `div`.

```

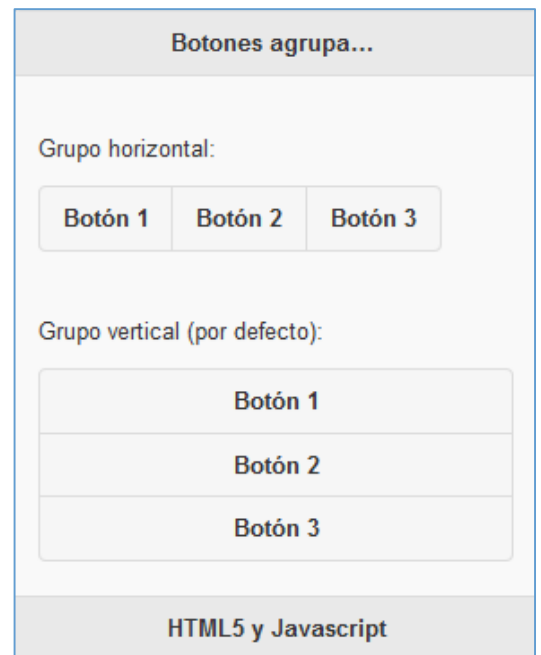
<body>
<div data-role="page" id="paginaPrincipal">
  <div data-role="header">
    <h1>Botones agrupados</h1>
  </div>

  <div data-role="main" class="ui-content">
    <div data-role="controlgroup"
      data-type="horizontal">
      <p>Grupo horizontal:</p>
      <a href="#pag1" class="ui-btn">Botón 1</a>
      <a href="#pag2" class="ui-btn">Botón 2</a>
      <a href="#pag3" class="ui-btn">Botón 3</a>
    </div><br>

    <div data-role="controlgroup"
      data-type="vertical">
      <p>Grupo vertical (por defecto):</p>
      <a href="#pag1" class="ui-btn">Botón 1</a>
      <a href="#pag2" class="ui-btn">Botón 2</a>
      <a href="#pag3" class="ui-btn">Botón 3</a>
    </div>
  </div>

  <div data-role="footer" data-position="fixed">
    <h1>Javascript</h1>
  </div>
</div>
</body>

```



Para incluir un botón de retroceso, basta con usar el atributo *data-rel="back"* (ignora el valor del atributo *href*):

```
<a href="#" class="ui-btn" data-rel="back">Atrás</a>
```

Para que un botón tenga comportamiento *in-line* (que ocupe el ancho de su contenido y pueda compartir su línea con otros elementos in-line), se usa la clase *ui-btn-inline*:

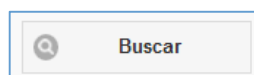
```
<a href="#pag2" class="ui-btn ui-btn-inline">Ir a la página 2</a>
```

Si se quiere reducir el tamaño de un botón de enlace (<a>), se le añadirá la clase *ui-mini*. Existe una larga lista de clases CSS con estilos predefinidos para aplicar a cualquier elemento (clases comunes), a botones (<a> y <button>, no <input>), a iconos (para botones, enlaces, items de listas), a temas (combinaciones de colores), a rejillas (*grid*), etc.

Iconos en botones de enlace

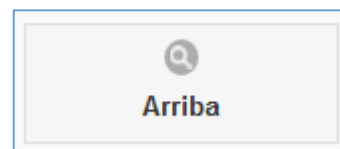
Para añadir un icono a un botón de enlace (<a>) se usa la clase *ui-icon*.

```
<a href="#pag2" class="ui-btn ui-icon-search ui-btn-icon-left">Buscar</a>
```



Existen clases para multitud de iconos: flechas (*ui-icon-arrow-r*, *ui-icon-arrow-l*), borrado (*ui-icon-delete*), inicio (*ui-icon-home*), audio (*ui-icon-audio*), etc.

Para posicionar el icono dentro del botón, se dispone de las clases: *ui-btn-icon-top* (arriba), *ui-btn-icon-bottom* (abajo), *ui-btn-icon-left* (izquierda) y *ui-btn-icon-right* (derecha).



Si se quiere omitir el texto del botón y mostrar sólo el icono, se utiliza la clase *ui-btn-icon-notext*.

Para otro tipo de botones, se debe usar el atributo *data-icon*.

Popup

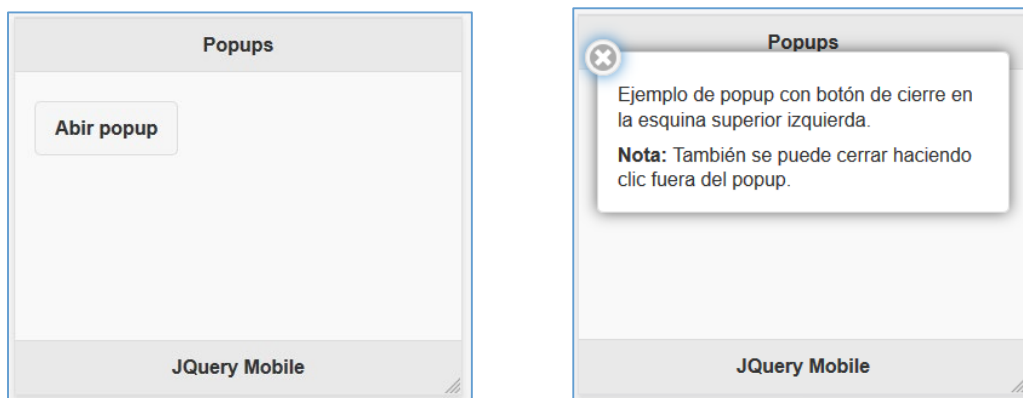
Para crear un popup, se necesitan un enlace que lo abra y un div que contendrá la información del pop-up. El enlace debe contener el atributo *data-rel="popup"* y el div tendrá *data-role="popup"*, así como un identificador *id* que será referenciado desde el enlace:

```
<a href="#pop1" data-rel="popup" class="ui-btn ui-btn-inline ui-corner-all">
  Abre el popup</a>

<div data-role="popup" id="pop1">
  <p>Esto es un popup</p>
</div>
```

Para añadir más margen y relleno en el popup, se utiliza la clase *ui-content* en el *div*.

Para cerrar un popup se puede hacer clic fuera de su área o usar el botón de escape. Para evitar lo primero, se puede añadir al div el atributo *data-dismissable="false"*. Para mostrar una equis en la esquina del popup que permita cerrarlo, se puede añadir un botón con el atributo *data-rel="back"* y posicionarlo por medio de las clases CSS.



A diferencia de los *popups*, los diálogos tienen las siguientes características:

- ocupan toda la página anterior, oscureciéndola
- se pueden encadenar
- aplican transiciones.

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
  <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
  </script>
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Popups</h1>
    </div>

    <div data-role="main" class="ui-content">
      <a href="#pop1" data-rel="popup"
        class="ui-btn ui-btn-inline ui-corner-all">Abir popup</a>

      <div data-role="popup" id="pop1" class="ui-content">
        <a href="#" data-rel="back" class="ui-btn ui-corner-all ui-shadow
          ui-btn ui-icon-delete ui-btn-icon-notext ui-btn-left">
          Cerrar <!-- no se muestra -->
        </a>
        <p>Ejemplo de popup con botón de cierre en la esquina
          superior izquierda.</p>
        <p><b>Nota:</b> También se puede cerrar haciendo clic fuera
          del popup.</p>
      </div>
    </div>

    <div data-role="footer" data-position="fixed">
      <h1>JQuery Mobile</h1>
    </div>
  </div>

</body>
</html>

```

Los popups se muestran sobre el elemento cuyo clic lo origina. Si se quiere controlar su ubicación, se puede usar el atributo *data-position-to*, que permite centrarlo con respecto a la ventana, posicionarlo sobre un elemento concreto (indicando su id) o sobre el elemento sobre el que se ha hecho clic (posicionamiento por defecto):

```

<a href="#a" data-rel="popup" class="ui-btn" data-position-to="window">Window</a>
<a href="#b" data-rel="popup" class="ui-btn" data-position-to="#elem">id="elem"</a>
<a href="#c" data-rel="popup" class="ui-btn" data-position-to="origin">Origen</a>

```

Se pueden aplicar transiciones con el atributo *data-transition* en el enlace:

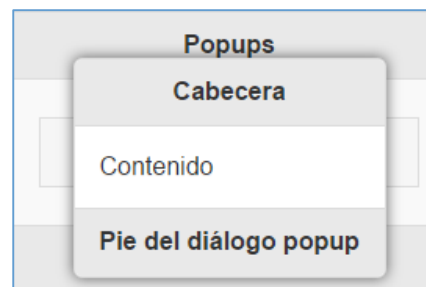
```

<a href="#pop1" data-rel="popup" class="ui-btn" data-transition="fade">Difumina</a>

```

Los popup pueden tener cabecera, contenido y pie, como cualquier otra página:

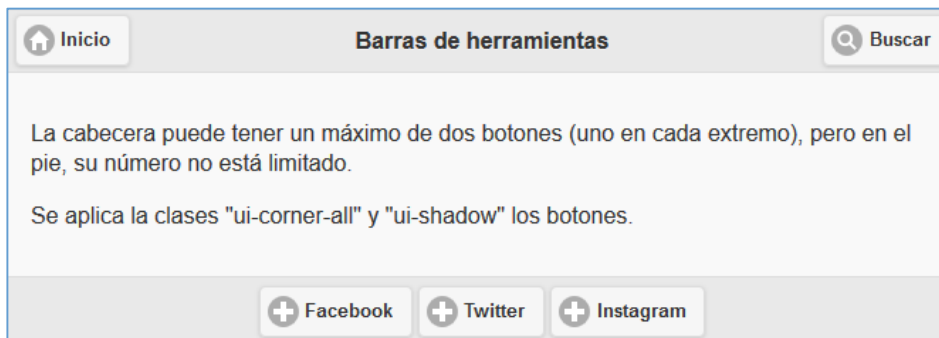
```
<a href="#popdiag" data-rel="popup" class="ui-btn">Abrir diálogo popup</a>
<div data-role="popup" id="popdiag">
  <div data-role="header"><h1>Cabecera</h1></div>
  <div data-role="main" class="ui-content">
    <p>Contenido</p>
  </div>
  <div data-role="footer">
    <h1>Pie del diálogo popup</h1>
  </div>
</div>
```



Barras de herramientas

Una cabecera (*header*) puede tener dos botones (uno en cada extremo). El pie (*footer*) puede tener un número ilimitado de botones. Sus botones no se centran por defecto, para lo cual hay que usar CSS (*style="text-align:center;"*)

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="https://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="https://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
  </head>
  <body>
    <div data-role="page">
      <div data-role="header">
        <a href="#" class="ui-btn ui-corner-all ui-shadow
          ui-icon-home ui-btn-icon-left ">Inicio</a>
        <h1>Barras de herramientas</h1>
        <a href="#" class="ui-btn ui-corner-all ui-shadow
          ui-icon-search ui-btn-icon-left ">Buscar</a>
      </div>
      <div data-role="main" class="ui-content">
        <p>La cabecera puede tener un máximo de dos botones (uno en cada
          extremo), pero en el pie, su número no está limitado.</p>
        <p>Se aplica la clases "ui-corner-all" y "ui-shadow" los botones.</p>
      </div>
      <div data-role="footer" data-position="fixed" style="text-align:center;">
        <a href="#" class="ui-btn ui-corner-all ui-shadow ui-icon-plus
          ui-btn-icon-left">Facebook</a>
        <a href="#" class="ui-btn ui-corner-all ui-shadow ui-icon-plus
          ui-btn-icon-left">Twitter</a>
        <a href="#" class="ui-btn ui-corner-all ui-shadow ui-icon-plus
          ui-btn-icon-left">Instagram</a>
      </div>
    </div>
  </body>
</html>
```



Los botones del *footer* se pueden agrupar por medio de un *div* y el atributo *data-role="controlgroup"*.

```
<div data-role="footer" data-position="fixed" style="text-align:center;">
  <div data-role="controlgroup" data-type="horizontal">
    <a href="#" class="ui-btn ui-icon-plus ui-btn-icon-left">Facebook</a>
    <a href="#" class="ui-btn ui-icon-plus ui-btn-icon-left">Twitter</a>
    <a href="#" class="ui-btn ui-icon-plus ui-btn-icon-left">Instagram</a>
  </div>
</div>
```

Para que las cabeceras (*header*) y pies (*footer*) queden fijas independientemente del desplazamiento (*scroll*), debe usarse el atributo *data-position="fixed"* (por defecto, este atributo toma el valor *"inline"*). Si, además, se quiere que ocupen todo el ancho de la pantalla, se usa el atributo *data-fullscreen="true"*.

```
<div data-role="header" data-position="fixed" data-fullscreen="true"></div>
<div data-role="footer" data-position="fixed" data-fullscreen="true"></div>
```

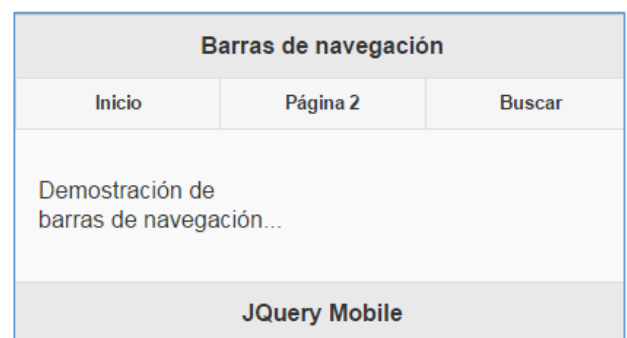
En este caso, cuando se toca la pantalla, se ocultan las cabeceras y pies con posiciones fijas y ancho de pantalla completa.

Barras de navegación

Una barra de navegación es grupo de enlaces alineados horizontalmente dentro de una cabecera o pie. Se suelen implementar como listas no ordenadas (**) dentro de un elemento *div* con el atributo *data-role="navbar"*.

Dentro de una barra de navegación los enlaces se convierten automáticamente en botones sin necesidad de añadir ningún atributo.

Estos botones ocuparán todo el ancho, a partes iguales. Si se incluyen más de cinco, se dividirán en más de una línea.



```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
  </head>
  <body>
    <div data-role="page" id="p1">
      <div data-role="header">
        <h1>Barras de navegación</h1>
        <div data-role="navbar">
          <ul>
            <li><a href="#a">Inicio </a></li>
            <li><a href="#b">Página 2 </a></li>
            <li><a href="#c">Buscar </a></li>
          </ul>
        </div>
      </div>
      <div data-role="main" class="ui-content">
        <p>Demostración de<br>barras de navegación...</p>
      </div>
      <div data-role="footer" data-position="fixed">
        <h1>jQuery Mobile</h1>
      </div>
    </div>
  </body>
</html>

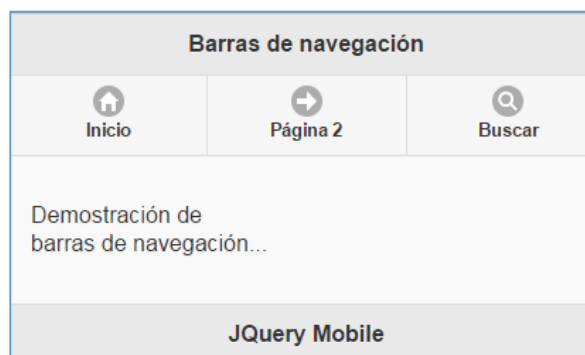
```

Para asignarle un icono y establecer su posición se utilizan los atributos *data-icon* ("home", "arrow-r", "search", etc.) y *data-iconpos* ("top", "right", "left" o "bottom") respectivamente:

```

<div data-role="navbar" data-iconpos="top">
  <ul>
    <li><a href="#a" data-icon="home">
      Inicio </a></li>
    <li><a href="#b" data-icon="arrow-r">
      Página 2 </a></li>
    <li><a href="#c" data-icon="search">
      Buscar </a></li>
  </ul>
</div>

```



Paneles

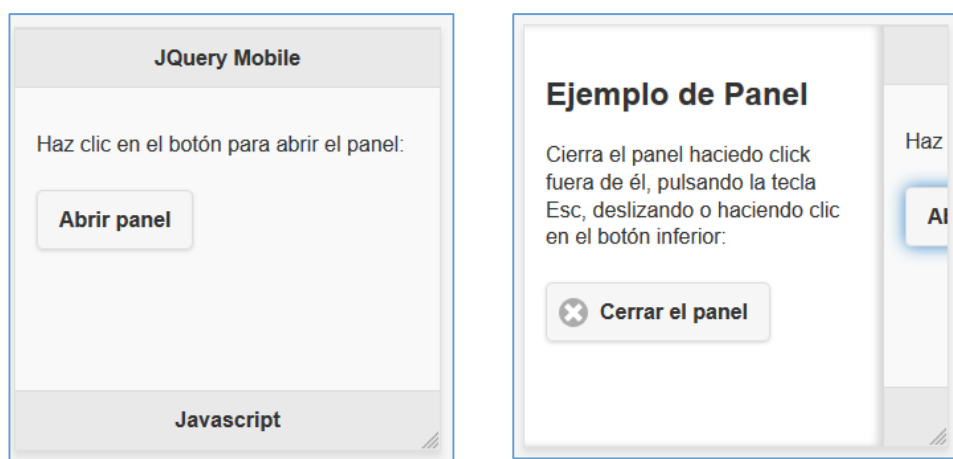
Un panel es un contenedor que se desplaza a la derecha o izquierda con contenido adicional. Para ello tan sólo hay que añadir a un *div* un *id* y el atributo *data-role="panel"*. Para acceder al panel, tan solo es necesario un enlace que apunte a su *id*.

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
  </head>
  <body>
    <div data-role="page" id="pag1">
      <div data-role="panel" id="panel1">
        <h2>Ejemplo de Panel</h2>
        <p>Cierra el panel haciendo click fuera de él, pulsando la tecla Esc,
          deslizando o haciendo clic en el botón inferior:</p>
        <a href="#pag1" data-rel="close" class="ui-btn ui-btn-inline ui-shadow
          ui-corner-all ui-btn-a ui-icon-delete ui-btn-icon-left">
          Cerrar el panel
        </a>
      </div>

      <div data-role="header">
        <h1>jQuery Mobile</h1>
      </div>
      <div data-role="main" class="ui-content">
        <p>Haz clic en el botón para abrir el panel:</p>
        <a href="#panel1"
          class="ui-btn ui-btn-inline ui-corner-all ui-shadow">
          Abrir panel
        </a>
      </div>
      <div data-role="footer" data-position="fixed">
        <h1>Javascript</h1>
      </div>
    </div>
  </body>
</html>

```



Es importante escribir el panel (*div*) antes o después de las tres secciones *header*, *footer* y *content* de la página que lo contiene.

Para cerrar un panel, es suficiente con hacer clic fuera de él o presionar la tecla *Esc*. Para gestionar estas dos posibilidades, se asignan los valores *true* o *false* a los atributos *data-dismissible* y *data-swipe-close*, respectivamente.

Para cerrarlo por medio de un botón, éste debe tener el argumento *data-rel="close"* y el atributo *href* indicando la página a la que debe regresar tras cerrar el panel.

Para controlar la animación por la que se muestra el panel y se oculta el contenido, se utiliza el atributo *data-display*.

- valor "overlay": el panel se muestra sobre el contenido.
- valor "push": el panel aparece empujando la página hacia fuera.
- valor "reveal": la página va desplazándose y va mostrando el panel (fijo) por debajo. Es el valor por defecto.

```
<div data-role="panel" id="panelOverlay" data-display="overlay">
<div data-role="panel" id="panelReveal" data-display="reveal">
<div data-role="panel" id="panelPush" data-display="push">
```

El atributo *data-position* especifica dónde se mostrará el panel: *right* o *left* (valor por defecto):

```
<div data-role="panel" id="panelABC" data-position="right">
```

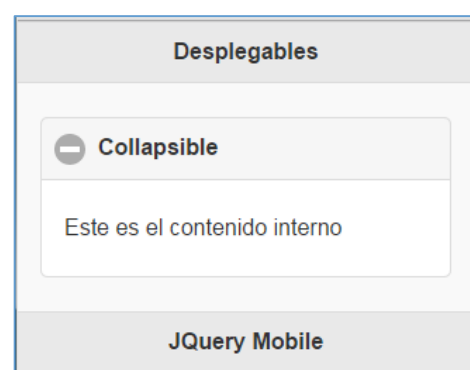
Cuando se muestra el panel, si el usuario hace un *scroll* de la página hay dos posibilidades: que la posición del panel no cambie, o que se desplace con ella. Para establecer el comportamiento, se dispone del atributo *data-position-fixed*, que puede tomar el valor *true* (panel inmóvil) o *false*.

```
<div data-role="panel" id="panelABC" data-position-fixed="true">
```

Desplegables

Estos elementos permiten ocultar o mostrar contenido. Para ello, se crea un contenedor con el atributo *data-role="collapsible"*. Dentro del contenedor debe tener un encabezado h (h1 a h6) o un elemento *<legend>*, seguido de cualquier elemento HTML que será el que se oculte o se muestre.

```
<body>
  <div data-role="page" id="pag1">
    <div data-role="header">
      <h1>Desplegables</h1>
    </div>
    <div data-role="main" class="ui-content">
      <div data-role="collapsible">
        <h1>Collapsible</h1>
        <p>Este es el contenido interno</p>
      </div>
    </div>
    <div data-role="footer" data-position="fixed">
      <h1>jQuery Mobile</h1>
    </div>
  </div>
</body>
```



Para que el desplegable se muestre inicialmente desplegado, se asigna el valor *false* al atributo *data-collapsed*: `<div data-role="collapsible" data-collapsed="false">`

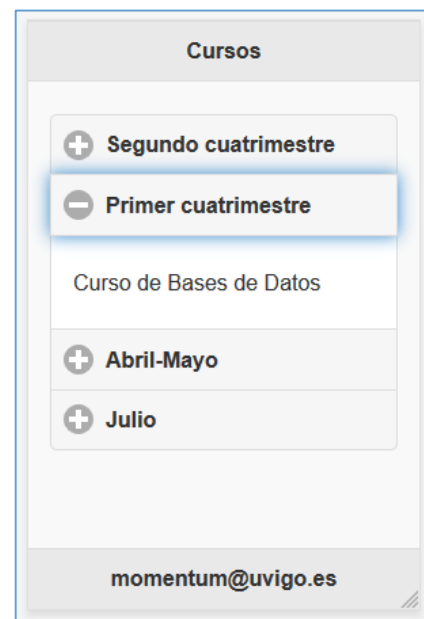
Los desplegables pueden anidarse:

```
<div data-role="collapsible">
  <h1>Desplegable 1</h1>
  <p>Contenido 1</p>
  <div data-role="collapsible">
    <h1>Deplegable 2</h1>
    <p>Contenido 2 </p>
  </div>
</div>
```



Los desplegables se pueden agrupar en conjuntos (llamados acordeones), de forma que cuando se despliega un conjunto se cierran los demás. Para crear un acordeón, se agrupan los desplegables en un contenedor con el atributo *data-role="collapsibleset"*:

```
<div data-role="page" id="pag1">
  <div data-role="header"> <h1>Cursos</h1></div>
  <div data-role="main" class="ui-content">
    <div data-role="collapsibleset">
      <div data-role="collapsible">
        <h3>Segundo cuatrimestre</h3>
        <p>Javascript</p>
      </div>
      <div data-role="collapsible">
        <h3>Primer cuatrimestre</h3>
        <p>Curso de Bases de Datos</p>
      </div>
      <div data-role="collapsible">
        <h3>Abril-Mayo</h3>
        <p>Curso de Programación Java</p>
      </div>
      <div data-role="collapsible">
        <h3>Julio</h3>
        <p>Curso de Programación Web con PHP</p>
      </div>
    </div>
  </div>
  <div data-role="footer" data-position="fixed"><h1>momentum@uvigo.es</h1></div>
</div>
```



Con el atributo *data-mini="true"* se consigue que el encabezado o leyenda que se muestra en el desplegable tenga un tamaño de fuente menor.

Es posible mostrar iconos distintos cuando el *collapsible* está desplegado o plegado, usando los atributos *data-collapsed-icon* y *data-expanded-icon*. En el ejemplo se utiliza una flecha hacia abajo y otra hacia arriba.

```
<div data-role="collapsible" data-collapsed-icon="arrow-d"
      data-expanded-icon="arrow-u" >
```

También es posible especificar la posición del icono por medio de los valores "right" y "left" en el atributo *data-iconpos*.

Las tablas en en JQuery Mobile son responsivas y se definen con el atributo *data-role="table"*. Hay dos tipos distintos: tablas *reflow* y tablas *column toggle*.

Las primeras muestran la información por filas, pero en caso de no ser posible, organiza la información en columnas. Para ello, tan solo es necesario asignarle la clase *"ui-responsive"*.

```
<body>
<div data-role="page" id="pag1">
  <div data-role="header"><h1>Tabla Reflow</h1></div>
  <div data-role="main" class="ui-content">
    <p>Cuando el ancho de la página es demasiado pequeño, la información pasa de
      mostrarse por filas a mostrarse en columnas</p>
    <table data-role="table" class="ui-responsive">
      <thead>
        <tr>
          <th>ID</th>
          <th>Empresa</th>
          <th>Contacto</th>
          <th>Dirección</th>
          <th>Ciudad</th>
          <th>Código postal</th>
          <th>País</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>1</td>
          <td>Alfreds Futterkiste</td>
          <td>Maria Anders</td>
          <td>Obere Str. 57</td>
          <td>Berlín</td>
          <td>12209</td>
          <td>Alemania</td>
        </tr>
        <tr>
          <td>2</td>
          <td>Antonio Moreno Taquería</td>
          <td>Antonio Moreno</td>
          <td>Mataderos 2312</td>
          <td>México D.F.</td>
          <td>05023</td>
          <td>México</td>
        </tr>
        <tr>
          ...
        </tr>
      </tbody>
    </table>
  </div>
  <div data-role="footer" data-position="fixed"><h1>JQuery Mobile</h1></div>
</div>
</body>
```

Cuando el ancho de la página es demasiado pequeño, la información pasa de mostrarse por filas a mostrarse en columnas	
ID	1
Empresa	Alfreds Futterkiste
Contacto	Maria Anders
Dirección	Obere Str. 57
Ciudad	Berlín
Código postal	12209
País	Alemania
ID	2
Empresa	Antonio Moreno Taquería
Contacto	Antonio Moreno
Dirección	Mataderos 2312
Ciudad	México D.F.
Código postal	05023
País	México
ID	3
Empresa	Around the Horn
Contacto	Thomas Hardy

Cuando el ancho de la página es demasiado pequeño, la información pasa de mostrarse por filas a mostrarse en columnas						
ID	Empresa	Contacto	Dirección	Ciudad	Código postal	País
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlín	12209	Alemania
2	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	México
3	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	Reino Unido
4	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Suecia

JQuery Mobile

La tabla *column toggle* ocultará columnas cuando no haya espacio para mostrarlas todas. Para crearla, además del atributo *data-role="table"*, se le añade el atributo *data-mode="columntoggle"* y la clase *class="ui-responsive"*:

```
<table data-role="table" data-mode="columntoggle" class="ui-responsive">
```

Las columnas se ocultan por defecto empezando por la derecha. Se puede establecer la prioridad para la ocultación de las columnas, en la cabecera de la tabla:

```
<th data-priority="6">CustomerID</th>
```

Los valores de *data-priority* pueden ir de 1 (máxima prioridad) a 6. Las columnas sin prioridad no se ocultarán.

```
<thead>
  <tr>
    <th data-priority="6">ID</th>
    <th>Empresa</th>
    <th data-priority="1">Contacto</th>
    <th data-priority="2">Dirección</th>
    <th data-priority="3">Ciudad</th>
    <th data-priority="4">Código postal</th>
    <th data-priority="5">País</th>
  </tr>
</thead>
```

Tabla Toggle		
Cuando el ancho de la página es demasiado pequeño, la información pasa de mostrarse por filas a mostrarse en columnas		
Columns...		
Empresa	Contacto	Dirección
Alfreds Futterkiste	Maria Anders	Obere Str. 57
Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312
Around the Horn	Thomas Hardy	120 Hanover Sq.
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8
jQuery Mobile		

Tabla Toggle		
Cuando el ancho de la página es demasiado pequeño, la información pasa de mostrarse por filas a mostrarse en columnas		
<div> <input type="checkbox"/> ID <input checked="" type="checkbox"/> Contacto <input checked="" type="checkbox"/> Dirección <input type="checkbox"/> Ciudad <input type="checkbox"/> Código postal <input type="checkbox"/> País </div>		
Empresa	Contacto	Dirección
Alfreds Futterkiste	Maria Anders	
Antonio Moreno Taquería	Antonio Moreno	120 Hanover Sq.
Around the Horn	Thomas Hardy	
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8
jQuery Mobile		

El texto del botón de columnas ocultas por defecto pone "Columns...". Para modificarlo se usa el atributo *data-column-btn-text* de la tabla: `data-column-btn-text="+columnas"`

Rejillas (grids)

Para posicionar pequeños elementos (como por ejemplo botones) en forma de columnas, se usan los *grids*. Todas las columnas de un grid son del mismo ancho, sin borde, fondo, margen o relleno. Existen cinco formatos de rejillas, que se construyen añadiendo a un div una de las cinco clases disponibles: *ui-grid-solo* (una columna), *ui-grid-a* (dos columnas), *ui-grid-b* (tres columnas), *ui-grid-c* (cuatro columnas) y *ui-grid-d* (cinco columnas).

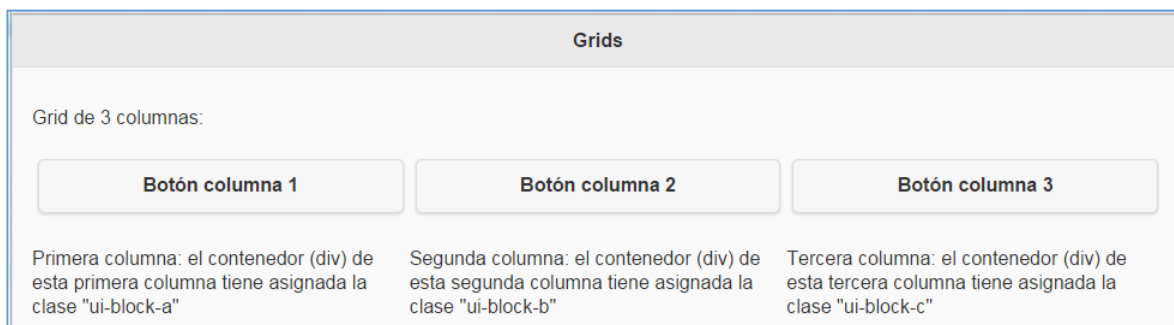
Cada columna será un *div* interno al grid, con la clase *ui-block-a*, *ui-block-b*, *ui-block-c*, *ui-block-d*, *ui-block-e*, en función de la posición de la columna (de 1 a 5).

```
<body>
  <div data-role="page" id="pag1">
    <div data-role="header"> <h1>Grids</h1> </div>
    <div data-role="main" class="ui-content">
      <p>Grid de 3 columnas:</p>
      <div class="ui-grid-b">
        <div class="ui-block-a">
          <a href="#pag2" class="ui-btn ui-corner-all ui-shadow">
            Botón columna 1
          </a><br>
          <span>Primera columna: el contenedor (div) de esta primera
            columna tiene asignada la clase "ui-block-a"
          </span>
        </div>
        <div class="ui-block-b">
          <a href="#pag3" class="ui-btn ui-corner-all ui-shadow">
            Botón columna 2
          </a><br>
          <span>Segunda columna: el contenedor (div) de esta segunda
            columna tiene asignada la clase "ui-block-b"
          </span>
        </div>
      </div>
    </div>
  </div>
```

```

        <div class="ui-block-c">
            <a href="#pag4" class="ui-btn ui-corner-all ui-shadow">
                Botón columna 3
            </a><br>
            <span>Tercera columna: el contenedor (div) de esta tercera
                columna tiene asignada la clase "ui-block-c"
            </span>
        </div>
    </div>
</div>
</div>
</body>

```



Para personalizar los estilos, se puede crear reglas CSS para las clases *ui-block-a*, *ui-block-b*, etc.

```

.ui-block-a, .ui-block-b, .ui-block-c
{
    background-color: lightgray;
    border: 1px solid black;
    height: 100px;
    font-weight: bold;
    text-align: center;
    padding: 30px;
}

```

Dado que los elementos de la clase *ui-block-x* representan la columna en la que se posiciona dicho elemento, podemos conseguir múltiples filas en cada columna repitiendo dicha clase:

```

<div class="ui-grid-b">
    <div class="ui-block-a" style="border:1px solid black;">
        <span>Texto A1</span>
    </div>
    <div class="ui-block-b" style="border:1px solid black;">
        <span>Texto B1</span>
    </div>
    <div class="ui-block-c" style="border:1px solid black;">
        <span>Texto C</span>
    </div>
    <div class="ui-block-a" style="border:1px solid black;">
        <span>Texto A2</span>
    </div>
    <div class="ui-block-b" style="border:1px solid black;">
        <span>Texto B2</span>
    </div>
    <div class="ui-block-a" style="border:1px solid black;">
        <span>Texto A3</span>
    </div>
</div>

```

Grids de múltiples filas		
Disposición en tres columnas con múltiples filas		
Texto A1	Texto B1	Texto C
Texto A2	Texto B2	
Texto A3		

Se puede conseguir un comportamiento responsivo de modo que cuando no se pueda mostrar la información en el número de columnas especificadas, se reordena como una única columna y tantas filas como columnas se indicó con la clase *ui-grid-x*:

```
<div class="ui-grid-b ui-responsive">
```

Grids

Grid de 3 columnas:

Botón columna 1

Primera columna: el contenedor (div) de esta primera columna tiene asignada la clase "ui-block-a"

Botón columna 2

Segunda columna: el contenedor (div) de esta segunda columna tiene asignada la clase "ui-block-b"

Botón columna 3

Tercera columna: el contenedor (div) de esta tercera columna tiene asignada la clase "ui-block-c"

Grids

Grid de 3 columnas:

Botón columna 1

Botón columna 2

Botón columna 3

Primera columna: el contenedor (div) de esta primera columna tiene asignada la clase "ui-block-a"

Segunda columna: el contenedor (div) de esta segunda columna tiene asignada la clase "ui-block-b"

Tercera columna: el contenedor (div) de esta tercera columna tiene asignada la clase "ui-block-c"

Listas

En JQuery Mobile, las listas ordenadas o sin ordenar, se crean con las etiquetas `` y `` respectivamente, añadiéndoles el atributo *data-role="listview"*.

```
<body>
  <div data-role="page" id="pag1">
    <div data-role="main" class="ui-content">
      <h2>Listas con divisores</h2>
      <ul data-role="listview" data-inset="true">
        <li data-role="list-divider">Europa</li>
        <li><a href="#nor">Noruega</a></li>
        <li><a href="#ale">Alemania</a></li>
        <li data-role="list-divider">Asia</li>
        <li><a href="#ind">India</a></li>
        <li><a href="#tai">Tailandia</a></li>
        <li data-role="list-divider">Africa</li>
        <li><a href="#mar">Marruecos</a></li>
        <li><a href="#tun">Túnez</a></li>
      </ul>
    </div>
  </div>
</body>
```

Listas con divisores

Europa

Noruega

Alemania

Asia

India

Tailandia

Africa

Marruecos

Túnez

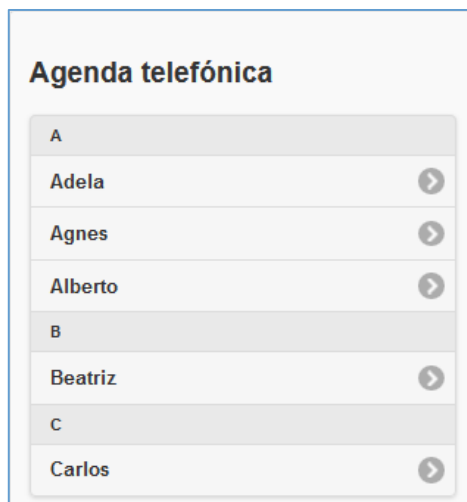
En el ejemplo, se usa el atributo `data-inset="true"` para redondear las esquinas y añadir algo de margen.

Los elementos `` con el atributo `data-role="list-divider"` actúan como divisores y por eso se muestran con un estilo distinto. Los elementos de las listas (sean divisores o no) pueden ser combinaciones de enlaces, texto, paneles, imágenes etc.

Divisores automáticos

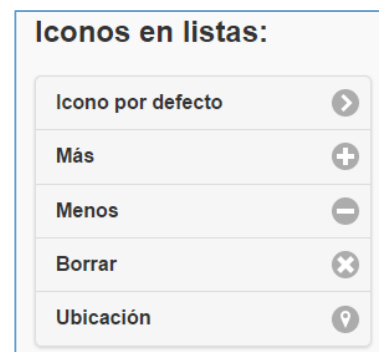
Si se quieren divisores automáticos para ordenación alfabética, tan sólo hay que añadir a la lista (`ol` / `ul`) el atributo `data-autodividers="true"`:

```
<div data-role="page" id="pag1">
  <div data-role="main" class="ui-content">
    <h2>Agenda telefónica</h2>
    <ul data-role="listview"
        data-autodividers="true" data-
inset="true">
      <li><a href="#">Adela</a></li>
      <li><a href="#">Agnes</a></li>
      <li><a href="#">Alberto</a></li>
      <li><a href="#">Beatriz</a></li>
      <li><a href="#">Carlos</a></li>
      ...
      <li><a href="#">Zacarías</a></li>
    </ul>
  </div>
</div>
```



Iconos estándar

```
<ul data-role="listview" data-inset="true">
  <li><a href="#">Icono por defecto</a></li>
  <li data-icon="plus"><a href="#">Más</a></li>
  <li data-icon="minus"><a href="#">Menos</a></li>
  <li data-icon="delete"><a href="#">Borrar</a></li>
  <li data-icon="location">
    <a href="#">Ubicación</a>
  </li>
</ul>
```



El valor `data-icon="false"` se utiliza para que no se muestre icono alguno.

Iconos 16x16

```
<ul data-role="listview" data-inset="true">
  <li>
    <a href="#">
      
      EE.UU.
    </a>
  </li>
  ...
  <li>
    <a href="#">Francia</a>
  </li>
</ul>
```



Miniaturas

Si se usan imágenes superiores a 16x16 píxeles (iconos), estas se escalarán a 80x80 píxeles.

```
<div data-role="page" id="pag1">
  <div data-role="main" class="ui-content">
    <h2>Lista con miniaturas</h2>
    <ul data-role="listview" data-inset="true">
      <li> <a href="#chrome">
        <h2>Google Chrome</h2>
        <p>Desarrollado por Google y publicado en 2008</p>
      </a>
    </li>
    <li> <a href="#firefox/">
        <h2>Mozilla Firefox</h2>
        <p>Mozilla Firefox es un navegador web libre y de código
          abierto publicado en 2004.</p>
      </a>
    </li>
    </ul>
  </div>
</div>
```



Botones divididos

Si un elemento contiene dos enlaces, automáticamente dibujará una línea de división a la derecha de modo que si se hace clic en el icono de la derecha se sigue el segundo enlace. Si se hace clic a la izquierda de la división se seguirá el primer enlace:

```
<li>
  <a href="#chrome">
    
    <h2>Google Chrome</h2>
    <p>Desarrollado por Google publicado en 2008</p>
  </a>
  <a href="http://www.google.com/intl/es/chrome/">Sitio oficial</a>
</li>

<li>
  <a href="#firefox">
    
    <h2>Mozilla Firefox</h2>
    <p>Desarrollado por la fundación
      Mozilla. Publicado en 2004</p>
  </a>
  <a href="https://www.mozilla.org/es-ES/firefox/">Sitio oficial</a>
</li>
```



Contadores de burbuja

Permiten mostrar un número al lado del icono de la derecha, como por ejemplo para indicar el número de mensajes de correo electrónico en cada carpeta.

```

<div data-role="page" id="pag1">
  <div data-role="main" class="ui-content">
    <h2>Correo electrónico</h2>
    <ul data-role="listview" data-inset="true">
      <li>
        <a href="#in">
          Bandeja de entrada
          <span class="ui-li-count">25</span>
        </a>
      </li>
      <li>
        <a href="#out">Mensajes enviados<span class="ui-li-count">432</span></a>
      </li>
      <li><a href="#trash">Papelera<span class="ui-li-count">7</span></a></li>
    </ul>
  </div>
</div>

```



Filtros

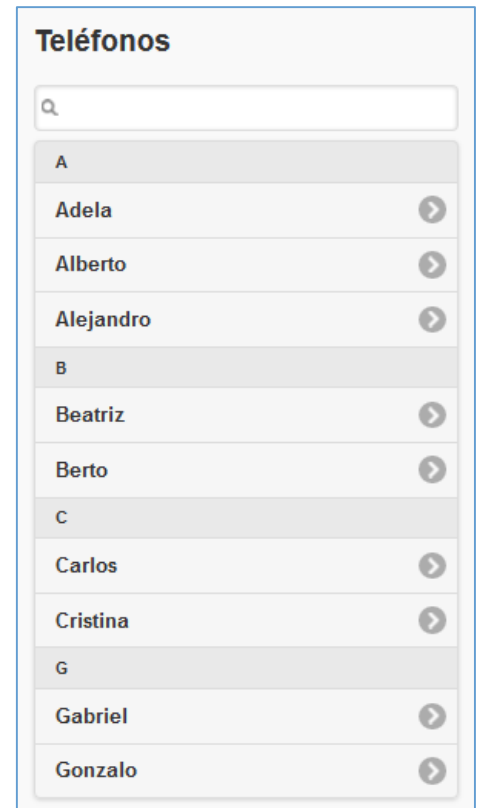
Se pueden filtrar los elementos de la lista. Para ello hay que seguir tres pasos:

1. Añadir a la lista (elemento `` o ``) el atributo `data-filter="true"`.
2. Crear un elemento `<input>` con un identificador (atributo `id`) y con el atributo `data-type="search"`. Por motivos de formato, se recomienda introducirlo en un elemento `form` con la clase `"ui-filterable"`.
3. Añadir el atributo `data-input` a la lista, con el valor del identificador del elemento `<input>` anterior.

```

<body>
  <div data-role="page" id="pag1">
    <div data-role="main" class="ui-content">
      <h2>Teléfonos</h2>
      <form class="ui-filterable">
        <input id="filtrol" data-type="search">
      </form>
      <ul data-role="listview" data-filter="true" data-input="#filtrol"
        data-autodividers="true" data-inset="true">
        <li><a href="#adela">Adela</a></li>
        <li><a href="#alberto">Alberto</a></li>
        <li><a href="#alex">Alejandro</a></li>
        <li><a href="#beatriz">Beatriz</a></li>
        <li><a href="#berto">Berto</a></li>
        <li><a href="#carlos">Carlos</a></li>
        <li><a href="#cris">Cristina</a></li>
        <li><a href="#gaby">Gabriel</a></li>
        <li><a href="#gonzalo">Gonzalo</a></li>
      </ul>
    </div>
  </div>
</body>

```



Si se quiere que el filtro no use el texto del `` para la búsqueda, sino un texto distinto, se debe asignar dicho texto nuevo al atributo `data-filtertext`. En este caso, el texto inicial será obviado en el filtrado:

```
<li data-filtertext="Chiki"><a href="#adela">Adela</a></li>
```

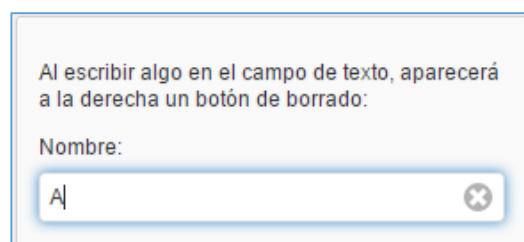
En este ejemplo, no se tendrá en cuenta el nombre mostrado (Adela), sino solo el texto indicado en el atributo `data-filtertext` ("Chiki").

Formularios

JQuery Mobile utiliza CSS para dar estilo de forma automática a los elementos de los formularios: campos de texto y de búsqueda, botones de radio, casillas de verificación, menús de selección, deslizadores e interruptores.

Para usar las características de los formularios de JQuery Mobile, los elementos `<form>` deben tener asignado un método de conexión ("get" o "post") y un atributo `action`. Todos los componentes de un formulario deben tener un identificador único (*id*) a lo largo de todas las páginas (ya que puede haber varias páginas en el mismo fichero). Además, cada elemento del formulario también debe tener un elemento *label* asignándole el *id* del elemento correspondiente a su atributo *for*.

```
<body>
<div data-role="page">
  <div data-role="main" class="ui-content">
    <form method="post" action="demoform.php">
      <p>Al escribir algo en el campo de texto,
        aparecerá a la derecha un botón de
        borrado:</p>
      <label for="nombre">Nombre:</label>
      <input type="text" name="nombre"
        id="nombre" data-clear-btn="true">
    </form>
  </div>
</div>
</body>
```



El atributo `data-clear-btn`, hace que al escribir algo en el cuadro de texto, aparezca un botón de borrado a la derecha (icono con el aspa). Se puede usar con cualquier tipo de elemento *input* pero no con el *textarea*.

Para ocultar la etiqueta (por ejemplo por que se usa un *placeholder* en el campo de entrada) se asigna la clase `class="ui-hidden-accessible"`.

Botones

JQuery Mobile da estilo automáticamente a los botones estándar de HTML (*button*, *submit* y *reset*) para que sean cómodos tanto en aplicaciones de escritorio como en dispositivos móviles.

Además, con los botones `<input>` se pueden usar los siguientes atributos: *data-corners* (esquinas redondeadas), *data-icon* (icono del botón), *data-iconpos* (posición del icono), *data-inline* (si el botón es inline o no), *data-mini* (botón pequeño) y *data-shadow* (botón con sombras).

```

<div data-role="page">
  <div data-role="main" class="ui-content">
    <form method="post" action="demoform.php">
      <label for="nombre">Nombre:</label>
      <input type="text" name="nombre" id="nombre">
      <input type="submit" value="Enviar" data-icon="check"
        data-iconpos="right" data-inline="true">
    </form>
  </div>
</div>

```

Contenedores

Se pueden usar elementos *div* y *fieldset* (se verán más adelante) con la clase "ui-field-contain" para formatear los formularios adecuadamente. Por ejemplo, cuando la página es demasiado estrecha, las etiquetas no se alinean con los campos, sino que se sitúan en la línea anterior.

```

<body>
<div data-role="page">
  <div data-role="main" class="ui-content">
    <form method="post" action="demoform.php">
      <div class="ui-field-contain">
        <label for="nombre">Nombre:</label>
        <input type="text" name="nombre" id="nombre">
        <label for="apellidos">Apellidos:</label>
        <input type="text" name="apellidos" id="apellidos">
      </div>
      <input type="submit" data-inline="true" value="Enviar">
    </form>
  </div>
</div>
</body>

```

En caso de no querer que jQuery Mobile de estilo automáticamente a los elementos, se debe usar el atributo *data-role="none"*:

```

<label for="nombre">Nombre:</label>
<input type="text" name="nombre" id="nombre" data-role="none">

```

Elementos de entrada

Estos elementos (incluyendo los de HTML5, ya son formateados automáticamente para su uso en dispositivos móviles:

Los *textarea* crecerán a medida que hagan falta nuevas líneas. Los botones de radio se agrupan con el elemento *fieldset* y puede haber un elemento *legend* para él. En el *fieldset* se puede utilizar el atributo *data-type="horizontal"* que los botones de radio se dispongan en horizontal. El atributo *checked* marca de inicio un botón de radio o una casilla de verificación.

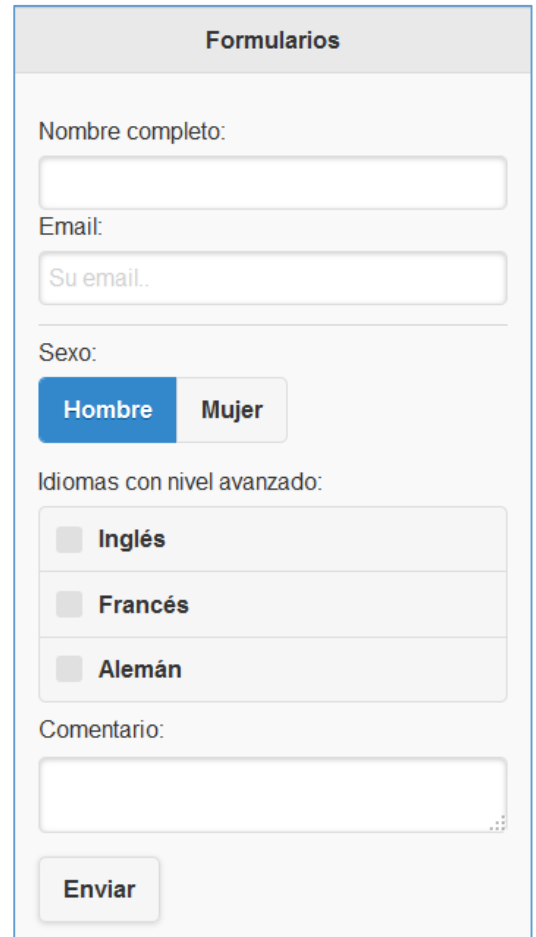
```
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>Formularios</h1>
    </div>
    <div data-role="main" class="ui-content">
      <form method="post" action="demoform.php">

        <div class="ui-field-contain">
          <label for="nombre">
            Nombre completo:</label>
          <input type="text" name="nombre"
            id="nombre">
          <label for="email">Email:</label>
          <input type="email" name="email"
            id="email"
            placeholder="Su email..">
        </div>

        <fieldset data-role="controlgroup"
          data-type="horizontal">
          <legend>Sexo:</legend>
          <label for="hombre">Hombre</label>
          <input type="radio" name="sexo"
            id="hombre" value="male"
            checked>
          <label for="mujer">Mujer</label>
          <input type="radio" name="sexo"
            id="mujer" value="mujer">
        </fieldset>

        <fieldset data-role="controlgroup">
          <legend>Idiomas con nivel avanzado:</legend>
          <label for="ingles">Inglés</label>
          <input type="checkbox" name="idiomas" id="ingles" value="ingles">
          <label for="frances">Francés</label>
          <input type="checkbox" name="idiomas" id="frances" value="frances">
          <label for="aleman">Alemán</label>
          <input type="checkbox" name="idiomas" id="aleman" value="aleman">
        </fieldset>

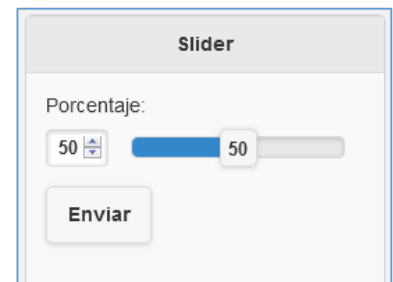
        <label for="info">Comentario:</label>
        <textarea name="comentario" id="info"></textarea>
        <input type="submit" data-inline="true" value="Enviar">
      </form>
    </div>
  </div>
</body>
```



Slider

Este tipo de control (*type="range"*) permite introducir un valor numérico deslizando un cursor:

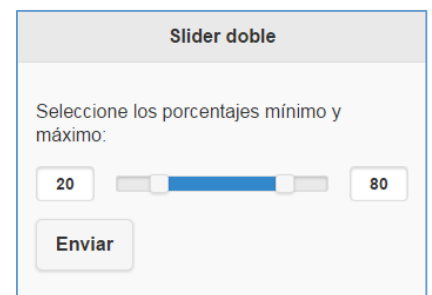
```
<div data-role="page">
  <div data-role="header"><h1>Slider</h1></div>
  <div data-role="main" class="ui-content">
    <form method="post" action="demoform.php">
      <label for="porcentaje">Porcentaje:</label>
      <input type="range" name="porcentaje"
        id="points" min="0" max="100" value="50"
        step="5" data-highlight="true"
        data-show-value="true">
      <input type="submit" data-inline="true" value="Enviar">
    </form>
  </div>
</div>
```



En el ejemplo anterior se usa los atributos *data-highlight* para destacar la parte inicial de la barra hasta el cursor. Con el atributo *data-show-value* se muestra el valor seleccionado sobre el cursor.

El siguiente ejemplo muestra un slider con dos cursores:

```
<div data-role="page">
  <div data-role="header"><h1>Slider doble</h1></div>
  <div data-role="main" class="ui-content">
    <p>Seleccione los porcentajes mínimo
      y máximo:</p>
    <form method="post" action="demoform.php">
      <div data-role="rangeslider" data-mini="true">
        <input type="range" name="minimo" id="minimo"
          value="20" min="0" max="100">
        <input type="range" name="maximo" id="maximo"
          value="80" min="0" max="100">
      </div>
      <input type="submit" data-inline="true" value="Enviar">
    </form>
  </div>
</div>
```



Interruptores (toggle switch)

Son botones con dos estados: activado (*on*) o desactivado (*off*). Se consiguen aplicando el atributo *data-role="flipswitch"* a una casilla de verificación (*input type="checkbox"*). Se puede controlar el texto que muestra (por defecto On / Off), por medio de los atributos *data-off-text* y *data-on-text*.

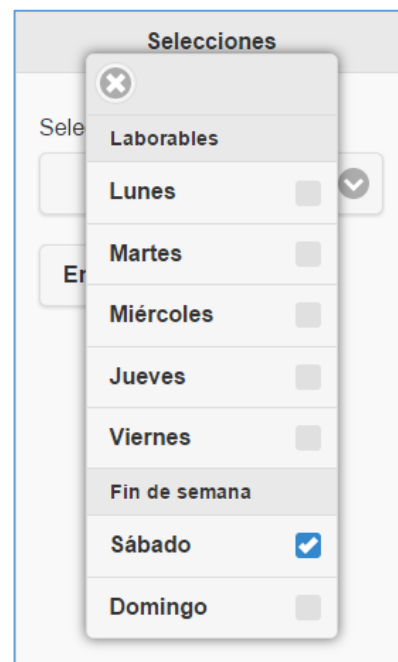
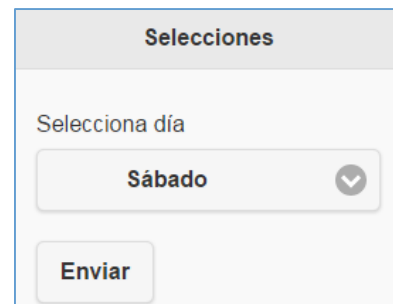
```
<div data-role="page">
  <div data-role="main" class="ui-content">
    <form method="post" action="demoform.php">
      <label for="switch">Interrupor:</label>
      <input type="checkbox" data-role="flipswitch" name="switch" id="switch"
        data-on-text="Sí" data-off-text="No"><br>
      <input type="submit" data-inline="true" value="Enviar">
    </form>
  </div>
</div>
```



Listas desplegables de selección

El elemento *select* crea una lista de opciones desplegable. Se pueden agrupar opciones con el elemento *optgroup*.

```
<div data-role="main" class="ui-content">
  <form method="post" action="demoform.php">
    <fieldset class="ui-field-contain">
      <label for="dia">Selecciona día</label>
      <select name="dia" id="dia"
        data-native-menu="false" multiple>
        <optgroup label="Laborables">
          <option value="lu">Lunes</option>
          <option value="ma">Martes</option>
          <option value="mi">Miércoles</option>
          <option value="ju">Jueves</option>
          <option value="vi">Viernes</option>
        </optgroup>
        <optgroup label="Fin de semana">
          <option selected value="sa">Sábado</option>
          <option value="do">Domingo</option>
        </optgroup>
      </select>
    </fieldset>
    <input type="submit" data-inline="true"
      value="Enviar">
  </form>
</div>
```



Para que el menú se muestre igual en todos los dispositivos, se debe usar el atributo `data-native-menu="false"`

El atributo *multiple* en el elemento *select* permite seleccionar más de una opción. Aquellas opciones que contengan el atributo *selected* aparecerán seleccionadas de inicio.

Se pueden hacer múltiples tareas como agrupar múltiples listas de selección en horizontal o vertical (`<fieldset data-role="controlgroup" data-type="horizontal">`), reducir el tamaño de las listas con el atributo *data-mini* (`<select name="dia" id="dia" data-mini="true">`), cambiar el icono y su posición con los atributos *data-icon* y *data-iconpos* respectivamente (`<select name="dia" id="dia" data-icon="plus" data-iconpos="left">`)

Formularios en desplegables y *pop-ups*

La inclusión de formularios en *pop-ups* o en desplegables (*collapsibles*) permite ocultar y mostrar de forma sencilla cualquier formulario.

Formularios en desplegados (*collapsibles*)

```
<div data-role="main" class="ui-content">
  <form method="post" action="demoform.php">
    <fieldset data-role="collapsible">
      <legend>Formulario</legend>
      <label for="nombre">Nombre completo:</label>
      <input type="text" name="texto" id="nombre">
      <p>Selecione color:</p>
      <div data-role="controlgroup">
        <label for="rojo">Rojo</label>
        <input type="checkbox" name="favcolor"
          id="rojo" value="rojo">
        <label for="verde">Verde</label>
        <input type="checkbox" name="favcolor"
          id="verde" value="verde">
        <label for="azul">Azul</label>
        <input type="checkbox" name="favcolor"
          id="azul" value="azul">
      </div>
      <input type="submit" data-inline="true"
        value="Enviar">
    </fieldset>
  </form>
</div>
```

Formularios en *pop-ups*

```
<div data-role="page">
  <div data-role="header"><h1>Formularios<br>pop-ups</h1></div>
  <div data-role="main" class="ui-content"><!-- ui-content: más margen y relleno-->
    <a href="#popup1" data-rel="popup"
      class="ui-btn ui-btn-inline ui-corner-all ui-icon-check ui-btn-icon-left">
      Abrir menú </a>
    <div data-role="popup" id="popup1"
      class="ui-content" style="min-width:250px;">
      <form method="post" action="demoform.php">
        <fieldset class="ui-field-contain">
          <label for="name">Nombre completo:</label>
          <input type="text" name="text" id="name">
          <p>Selecione color:</p>
          <div data-role="controlgroup">
            <label for="rojo">Rojo</label>
            <input type="checkbox" name="favcolor"
              id="rojo" value="rojo">
            <label for="verde">Verde</label>
            <input type="checkbox" name="favcolor"
              id="verde" value="verde">
            <label for="azul">Azul</label>
            <input type="checkbox" name="favcolor"
              id="azul" value="azul">
          </div>
          <input type="submit" data-inline="true" value="Enviar">
        </fieldset>
      </form>
    </div>
  </div>
</div>
```

jQuery Mobile proporciona dos combinaciones de colores predefinidas (temas). Se puede asignar un tema a la aplicación y también aplicar el tema individualmente a cada control. Los temas suelen ser de texto oscuro sobre fondo blanco (tema *a*) o texto claro sobre fondo oscuro (tema *b*). Por defecto, se aplica el tema *a* para la mayoría de los elementos. Los elementos heredan los temas de su padre.

Aplicar temas

Los temas se aplican por medio del atributo *data-theme*, excepto en los botones definidos por medio de la clase *ui-btn*. En este caso, se usan las clases *ui-btn-a* y *ui-btn-b*. En el caso del contenido de un collapsible, el atributo a usar es *data-content-theme*. De modo similar, en el caso de botones divididos, se utiliza el atributo *data-split-theme*.

páginas

```
<div data-role="page" data-theme="b">
```

buscar

```
<a href="#" class="ui-btn ui-btn-b ui-icon-search ui-btn-icon-notext">iconos</a>
```

header

```
<div data-role="header" data-theme="b"></div>
```

listas con botones divididos

```
<ul data-role="listview" data-split-theme="b">
```

botones

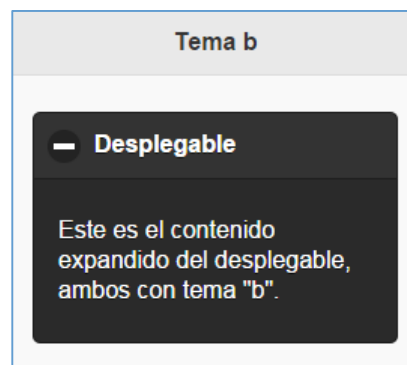
```
<a href="#" class="ui-btn ui-btn-b">Botón negro</a>
```

collapsible

```
<div data-role="main" class="ui-content">
  <div data-role="collapsible" data-theme="b"
    data-content-theme="b">
    <h1>Desplegable</h1>
    <p>Este es el contenido expandido del
      desplegable, ambos con tema "b".</p>
  </div>
</div>
```

pop-ups

```
<div data-role="popup" id="popup1" data-theme="b">
```



Definir temas nuevos

Es posible definir mediante CSS nuevos temas y modificar los ya existentes. Para ello, es suficiente con copiar un bloque de estilos, renombrar las clases con una letra entre c y z y modificar los colores y las fuentes como se desee.

La herramienta *ThemeRoller* (<http://themeroller.jquerymobile.com/>) facilita la tarea de editar y modificar temas.

Resumen de iconos

Como resumen sobre el uso de iconos, con enlaces `<a>` y botones de tipo `<button>`, se usan las clases *ui-icon-** (*ui-icon-refresh*, por ejemplo) para seleccionar el icono a utilizar.

Con los elementos `<input type="button">` y enlaces en las barras de navegación o en listas, se debe usar el atributo *data-icon* (*data-icon="refresh"*, por ejemplo)

Eventos

Además de los eventos estándar de jQuery, jQuery Mobile proporciona varios eventos diseñados para dispositivos móviles: eventos de toque (*touch events*), de desplazamiento de página (*scroll events*), de orientación (*orientation events*) y de página (*page events*).

Para evitar que se ejecute código de jQuery Mobile antes de que la página esté completamente cargada, se usa el evento *pagecreate*:

```
$(document).on("pagecreate", "#pag1", function(){...});
```

En versiones anteriores a la 1.4.0 de jQuery Mobile se usaba el evento *pageinit* para inicializar la página, pero actualmente está desaconsejado y debe utilizarse *pagecreate* en su lugar.

Si se quieren cambiar la configuración por defecto de jQuery Mobile, debe hacerse usando el evento *mobileinit*:

```
$( document ).on( "mobileinit", function() {  
    $.mobile.popup.prototype.options.overlayTheme = "b";  
});
```

Eventos touch

Toques: *tap*, *taphold* (toque largo, configurable), *swipe* (deslizamiento sobre un elemento; configurable), *swipeleft*, *swiperight*.

Ratón virtual: *vmouseover*, *vmouseout*, *vmousedown*, *vmousemove*, *vmouseup*, *vclick* (problemático en navegadores basados en webkit, por lo que se recomienda el evento *click* en su lugar) y *vmousecancel*.

Tap

En el siguiente ejemplo, se asocia una función anónima al evento *tap* que se produzca sobre los elementos `<p>`. Dicha función oculta el párrafo mediante la llamada del método *hide* de jQuery.

La asociación de la función al evento se realiza tras haberse completado la carga de la página, lo cual se detecta mediante el evento *pagecreate*.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
        href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
  <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
  <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
  </script>
  <script>
    $(document).on("pagecreate", "#pag1", function() {
      $("p").on("tap", function() {
        $(this).hide();
      });
    });
  </script>
</head>
<body>
  <div data-role="page" id="pag1">
    <div data-role="header"><h1>Evento Tap</h1></div>
    <div data-role="main" class="ui-content">
      <p>Toca aquí y desaparecerá.</p>
      <p>Toque corto y borra</p>
      <p>Un toque y desaparece</p>
    </div>
  </div>
</body>
</html>
```

Aunque se puede crear una función con nombre en lugar de anónima para asociar al evento *pagecreate*, no es posible evitar que la función asociada al otro evento (*tap* en este caso) sea anónima:

```
function inicializa() {
  $("p").on("tap", function() {           // no se puede evitar la función anónima
    $(this).hide();
  });
}
$(document).on("pagecreate", "#pag1", inicializa); // evitamos la función anónima
```

Taphold

```
$( "p" ).on( "taphold", function() {
  $(this).hide();
});
```

Swipe

```
$("#p").on("swipe",function(){
    $("#span").text("Deslizamiento detectado");
});
```

Swipeleft

```
$("#p").on("swipeleft",function(){
    alert("deslizamiento a la izquierda");
});
```

Swiperight

```
$("#p").on("swiperight",function(){
    alert("deslizamiento a la derecha");
});
```

Eventos scroll

Se dispone de dos eventos: `scrollstart` y `scrollstop`. iOS congela la manipulación del DOM cuando se hace scroll, por lo que las operaciones sobre aquel se encolan hasta que termine la operación de scroll. Por ejemplo:

```
$(document).on("scrollstop",function(){
    alert("Fin del scroll");
});
```

Evento orientation

El evento `orientationchange` se dispara cuando hay un cambio de orientación del dispositivo. El evento recibido por la función de respuesta incluye un campo *orientation* con el valor adecuado: "portrait" o "landscape". Si no se soporta de modo nativo este evento o se ha deshabilitado (estableciendo `$.mobile.orientationChangeEnabled="false"`), se invoca el evento *resize*.

```
$(document).on("pagecreate",function(){
    $(window).on("orientationchange",function(evento){
        alert("Nueva orientación: " + evento.orientation);
    });
});
```

Eventos de página

Hay cuatro categorías de eventos de página en jQuery Mobile: de inicialización, de carga y descarga, de transición y de cambio.

Eventos de inicialización

Dado que en jQuery Mobile se utiliza una estructura multipágina, el evento `$(document).ready` de jQuery se dispararía al tras cargar el fichero HTML, pero no una vez para cada página de jQuery Mobile. Por ello se sustituye por los siguientes eventos.

El evento *pagebeforecreate* se produce cuando en la página HTML ya se ha creado en el DOM y antes de que jQuery Mobile haya empezado a aplicar sus modificaciones. Es el evento a utilizar si se quiere modificar el DOM.

El evento *pagecreate* se dispara cuando la página se ha creado, jQuery Mobile ha inicializado sus elementos.

Finalmente, el evento *pageinit* aparece tras todas las inicializaciones, aunque su uso está desaconsejado y se recomienda usar *pagecreate* en su lugar.

Eventos de transición

Las transiciones de página (animaciones en el cambio de página) producen los siguientes eventos:

- *pagebeforeshow*: se dispara sobre la página destino antes de que empiece la animación.
- *pageshow*: se produce sobre la página destino tras completarse la animación.
- *pagebeforehide*: se dispara sobre la página origen antes de que empiece la animación.
- *pagehide*: se produce sobre la página origen tras completarse la animación.

Ejemplos:

```
$(document).on("pagebeforeshow", "#pag2", function(){ // Al entrar en pag2
    alert("se va a mostrar la pag2");
});
$(document).on("pageshow", "#pag2", function(){ // Al entrar en pag2
    alert("se está mostrando pag2");
});
$(document).on("pagebeforehide", "#pag2", function(){ // Al salir de pag2
    alert("se va a ocultar la pag2");
});
$(document).on("pagehide", "#pag2", function(){ // Al salir de pag2
    alert("la pag2 se ha ocultado");
});
```

Eventos de carga

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
      href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css">
    <script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
    <script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js">
    </script>
    <script>
      $(document).on("pagecontainerload", function(evento, info) {
        alert("página solicitada:\nURL: " + info.url);
      });
      $(document).on("pagecontainerloadfailed", function(evento, info) {
        alert("La página solicitada no existe");
      });
    </script>
  </head>
```

```

<body>
  <div data-role="page">
    <div data-role="header"><h1>Eventos de carga</h1></div>
    <div data-role="main" class="ui-content">
      <a href="pagexterna.html">Página externa</a><br><br>
      <a href="pagnoexiste.html">La página externa no existe</a>
    </div>
  </div>
</body>
</html>

```

Siempre que se carga una página externa en el DOM, se generan dos eventos: *pagecontainerbeforeload* (*pagebeforeload* está desaconsejado y en las últimas versiones ya no se dispara) y, al terminar, *pagecontainerload* o *pagecontainerloadfailed*, según la carga haya tenido éxito o no. Por defecto, estos eventos sólo se disparan cuando la página a cargar está en el mismo servidor (`$.mobile.allowCrossDomainPages` toma el valor *false* por defecto).

En los eventos de carga, el segundo atributo de la función de respuesta contiene información extra.

Evento *updatelayout*

Se dispara por los componentes de jQuery Mobile que muestran y ocultan contenido de forma dinámica, ya que ese comportamiento produce un cambio en el tamaño de la página (como por ejemplo *collapsible* o *listview*). El evento se genera sobre el componente que cambia de tamaño o posición y se transmite de forma ascendente hasta el elemento *document*.

```

$("#collapsible1").on("updatelayout", function() {
  alert("Cambio de tamaño");
});

```

Cuando se desarrolle una aplicación dinámica que introduzca, oculte o elimine contenido de la página, o lo manipule de cualquier modo que afecte a las dimensiones de la página, se puede disparar de forma explícita este evento, para asegurarse de que el resto de los componentes puedan actuar de acuerdo a los cambios. Por ejemplo:

```

$( "#abc" ).hide().trigger( "updatelayout" );

```

Inicializaciones

Existe una serie de opciones que definen el funcionamiento de jQuery Mobile. Para cambiar esta configuración por defecto, se dispone de un conjunto de variable. A continuación se presentan algunas con sus valores por defecto:

Transición en el cambio entre páginas: `$.mobile.defaultPageTransition = "fade";`

Transición entre diálogos: `$.mobile.defaultDialogTransition = "pop";`

Mensaje mientras se carga una página: `$.mobile.loadingMessage = "Cargando...";`

Mensaje si no se carga la página : `$.mobile.pageLoadErrorMessage = "Error de carga";`

Permiso para cargar páginas de servidores externos (por motivos de seguridad se inicializa a *false*):
`$.mobile.allowCrossDomainPages = false;`

Para inicializar estas variables se utiliza el evento *mobileinit*, que se dispara cuando jQuery Mobile ha terminado de cargarse, pero aún no ha empezado a ejecutarse.

```
<script>
    $( document ).on( "mobileinit", function() {
        $.mobile.loadingMessage = "Cargando...";
        $.mobile.pageLoadErrorMessage = "Error de carga";
    });
</script>
```