Javascript / jQuery

Índice de contenido

Javascript básico

JavaScript es un **lenguaje de programación interpretado**. Se utiliza principalmente en la **programación web del lado del cliente (client-side)**, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas del lado del cliente.

Es un **lenguaje imperativo y orientado a objetos**. Posee un tipado débil y dinámico, lo cual significa que no es necesario establecer el tipo de una variable y éste puede cambiar durante la ejecución del programa.

La sintaxis básica la toma del lenguaje C, por tanto se parece a otros lenguajes que también basan su sintaxis en C, como C++, Java o PHP.

Las estructuras de selección (if, switch) y repetición (for, while y do-while) son prácticamente las mismas en todos los lenguajes anteriores.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model(DOM).

Situación

A continuación se muestran dos formas de insertar código Javascript en una página web.

La primera forma utiliza el código que se halla en el archivo jquery-1.6.4.js, que se halla en el mismo directorio que la página web.

La segunda forma permite que podamos escribirlo directamente en la página web.

```
<html>
<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />   <!-- iso-8859-1, cp1252 -->

<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">

        document.write ("<h1>Título de nivel 1</h1>");
        document.write ("<h2>Título de nivel 2</h2>");

</script>

</head>
<body>
…
</body>
</html>
```

Como todavía no estamos utilizando la biblioteca jQuery, deberías eliminar la línea

```
<script type="text/javascript" src="jquery-1.6.4.js"></script>
```

si quieres que la página cargue correctamente.

Comentarios

Al igual que C y otros lenguajes similares se admiten los comentarios de bloque (/* */), y los comentarios de línea (//).

```
<script type="text/javascript">
  /*
     Script que pide nombre al usuario y después lo muestra
     JAMJ - 2012
  */
  var nombre;

  // Usamos función prompt para introducir datos
  nombre = prompt ("Introduce tu nombre : ", "nombre y apellidos");

  // Usamos función alert para mostrar datos
  alert ("Tu nombre es "+ nombre);

</script>
```

Tipos de de datos

```
var dato;

dato = 123;             // ahora dato es un número
dato = "Hola que tal!"; // ahora dato es una cadena de texto o string
dato = new Array ();    // ahora dato es un array
dato = {};              // ahora dato es un diccionario
```

Si incrustamos el siguiente código javascript en un archivo HTML

```
var a = 123.01;
var b = "Hola que tal!";
var c = new Array ();
var d = {};
var f = function ()  {
    document.write ("Hola ");
    document.write ("a todos");
}

document.write (typeof a +"<br>"+ typeof b +"<br>"+ typeof c  +"<br>"+ typeof d  +"<br>"+ typeof f);
```

el resultado será:

number

string

object

object

function

Observa que tanto los arrays como los diccionarios se interpretan como objetos.

Para dar valores a un **array:**

```
var direccion = new Array ();

direccion[0] = "Avda. ";
direccion[1] = "Estudiantes ";
direccion[2] = 99;

alert (direccion[0] + direccion[1] + direccion[2]);
```

Para dar valores a un **diccionario:**

```
var nota = {};

nota["Rocio"] = "Sobresaliente";
nota["Eva"]   = "Notable";
nota["Jairo"] = "Suficiente";

alert (nota["Rocio"] + "\n" + nota["Eva"] + "\n" + nota["Jairo"]);
```

Una forma más sencilla de realizar lo anterior es:

```
var direccion = new Array ("Avda. ",  "Estudiantes ",  99);

alert (direccion[0] + direccion[1] + direccion[2]);


var nota = {Rocio: "Sobresaliente",  Eva: "Notable",  Jairo: "Suficiente"};

alert (nota["Rocio"] + "\n" + nota["Eva"] + "\n" + nota["Jairo"]);
```

Observa como utilizamos una **coma** para la separación entre los distintos elementos de un array o un diccionario.

Se pueden crear **objetos** haciendo uso de diccionarios. Por ejemplo:

```
var miObjeto = {
    miNombre : 'Jose'
    ,
    diHola : function() { alert ('Hola'); }
};

miObjeto.diHola();
alert (miObjeto.miNombre);
```

miObjeto tiene 2 elementos: el string **miNombre** (equivalente a una **propiedad** o atributo) y la función **diHola** (equivalente a un **método**). Observa que cada elemento consta de una clave y un valor. La separación entre elementos se realiza utilizando la coma (**,**)

Operadores

Concatenación

Los **operadores aritméticos** son los mismos que los existentes para la mayoría de lenguajes:

+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto

También existen los operadores

++	Incremento
--	Decremento

El operador + también se utiliza para la **concatenación de cadenas de texto**. Ej:

```
var a = 'hola';
var b = 'mundo';

alert (a + ' ' + b); // Se muestra 'hola mundo'
```

Cuando se concatena un número y una cadena el resultado es una cadena

```
var a = 1;
var b = '2';

alert (a + b);  // error: Se muestra 12
```

Podemos forzar a una cadena de caracteres actuar como un número mediante el constructor Number():

```
var a = 1;
var b = '2';

alert (a + Number(b));  // se muestra 3
```

o mediante el operador unario +:

```
alert (a + +b);         // se muestra 3
```

Al igual que en el lenguaje C, también se admiten los operadores que trabajan a nivel de bits.

>> Desplazar a la derecha un número de bits

<< Desplazar a la izquierda un número de bits

& Operación AND

| Operación OR

^ Operación XOR

~ Operación NOT

Ejemplo:

```
var a = 2;    // ...010
var b = 3;    // ...011
var c = 4;    // ...100

document.write (c >> 2);  //  ...001  = 1
document.write ("<br>");
document.write (a << 1);  //  ...100  = 4
document.write ("<br>");
```

```
document.write (a & b);   //  ...010  = 2
document.write ("<br>");
document.write (a | b);   //  ...011  = 3
document.write ("<br>");
document.write (a ^ b);   //  ...001  = 1
document.write ("<br>");
document.write (~a );     //  ...111101  = -3 (número en complemento a 2)
document.write ("<br>");
```

Los **operadores lógicos** son && (and) y || (or) y ! (not). Ej:

```
var foo = 1;
var bar = 0;
var baz = 2;


foo && bar;   // devuelve 0, el cual es falso (false)
foo && baz;   // devuelve 2, el cual es verdadero (true)
baz && foo;   // devuelve 1, el cual es verdadero (true)

foo || bar;   // devuelve 1, el cual es verdadero (true)
bar || foo;   // devuelve 1, el cual es verdadero (true)

!bar;         // devuelve verdadero (true)
!baz;         // devuelve falso (false)
```

Resumiendo, estos operadores se comportan de la siguiente forma:

&& Devuelve true si los dos operandos son verdaderos, en el resto de casos devuelve false.

|| Devuelve false si los dos operandos son falsos, en el resto de casos devuelve true.

! Devuelve lo opuesto al valor del operando.

Los **operadores de comparación** son

== Igual

!= Distinto

=== Igual y con el mismo tipo

!== Distinto o con distinto tipo

> Mayor

>= Mayor o igual

< Menor

<= Menor o igual

Ejemplo:

```
var foo = 1;
var bar = 0;
var baz = '1';
var bim = 2;

foo == bar;   // devuelve falso (false)
foo != bar;   // devuelve verdadero (true)
foo == baz;   // devuelve verdadero (true); tenga cuidado

foo === baz;             // devuelve falso (false)
foo !== baz;             // devuelve verdadero (true)
foo === parseInt(baz);   // devuelve verdadero (true)

foo > bim;    // devuelve falso (false)
bim > baz;    // devuelve verdadero (true)
foo <= baz;   // devuelve verdadero (true)
```

Elementos Verdaderos y Falsos

Para controlar el flujo adecuadamente, es importante entender qué tipos de valores son “verdaderos” y cuales “falsos”. A veces, algunos valores pueden parecer una cosa pero al final terminan siendo otra.

Valores que devuelven verdadero (true)

```
'0';
'a veces';  // cualquier cadena
[];         // un arreglo vacío
{};         // un objeto vacío
1;          // cualquier número distinto a cero
```

Valores que devuelven falso (false)

```
0;
'';         // una cadena vacía
NaN;        // la variable JavaScript "not-a-number" (No es un número)
null;       // un valor nulo
undefined;  // tenga cuidado -- indefinido (undefined) puede ser redefinido
```

Estructuras de decisión

```
var a = true;
var b = false;

if (b) {
      // este código nunca se ejecutará
      alert ('hola');
}

if (b) {
      // este código no se ejecutará
} else {
      if (a) {
          // este código se ejecutará
      } else {
          // este código se ejecutará si a y b son falsos (false)
      }
}
```

```
var a = "uno";

switch (a) {

      case 'uno':
          alert('el valor es uno');
          break;

      case 'dos':
          alert('el valor es dos');
          break;
```

```
      default:
          alert('cuando el valor no es uno ni dos');
          break;
}
```

Estructuras de iteración

Un bucle utilizando `for` se compone de cuatro estados y posee la siguiente estructura:

```
for ([expresiónInicial]; [condición]; [incrementoDeLaExpresión])
   [cuerpo]
```

El estado *expresiónInicial* es ejecutado una sola vez, antes que el bucle comience. éste otorga la oportunidad de preparar o declarar variables.
El estado *condición* es ejecutado antes de cada repetición, y retorna un valor que decide si el bucle debe continuar ejecutándose o no. Si el estado condicional evalúa un valor falso el bucle se detiene.
El estado *incrementoDeLaExpresión* es ejecutado al final de cada repetición y otorga la oportunidad de cambiar el estado de importantes variables. Por lo general, este estado implica la incrementación o decrementación de un contador.
El *cuerpo* es el código a ejecutar en cada repetición del bucle.

```
for (var i=0, limite=10; i<limite; i++) {
      // Este bloque de código será ejecutado 10 veces
      document.write ('<br>' + i);
}
```

Un bucle utilizando `while` es similar a una declaración condicional `if`, excepto que el cuerpo va a continuar ejecutándose hasta que la condición a evaluar sea falsa.

```
while ([condición]) [cuerpo]
```

```
var i=0;
var limite=10;

while (i<limite) {
      // Este bloque de código será ejecutado 10 veces
      document.write ('<br>' + i);
      i++;
}
```

El bucle `do-while` es exactamente igual que el bucle utilizando `while` excepto que el cuerpo es ejecutado al menos una vez antes que la condición sea evaluada.
`do [cuerpo] while ([condición])`

```
var i=0;
var limite=10;

do {
    // Este bloque de código será ejecutado 10 veces
    document.write ('<br>' + i);
    i++;
} while (i<limite);
```

Funciones

Las funciones en Javascript son muy flexibles, pueden ser utilizadas de múltiples maneras.

La forma más simple, cuando la declaramos para que realice una actividad y después la invocamos.

```
// Declaración de la función
var saludar = function(persona, saludo) {
      var texto = saludo + ', ' + persona;
      alert (texto);
```

```
};

// Invocación de la función
saludar ('Ana', 'Hola');  // muestra 'Hola, Ana'
```

Una función también puede devolver un valor. Entonces utilizar la función es igual a utilizar dicho valor.

```
var saludar = function(persona, saludo) {
      var texto = saludo + ', ' + persona;
      return texto;
};

alert (saludar ('Ana','Hola'));   // la función devuelve 'Hola, Ana', lo cual se muestra en el alert
```

Todavía más, una función puede devolver otra función

```
var saludar = function(persona, saludo) {
      var texto = saludo + ', ' + persona;
      return function() { alert (texto); };
};


var saludos = saludar('Ana', 'Hola');
saludos ();  // se muestra 'Hola, Ana'
```

jQuery

jQuery es una una biblioteca (library) desarrollada en Javascript que permite básicamente 3 cosas:

- Efectos visuales.
- Modificación de los elementos de una página web (inserción, modificación, etc)
- Carga asíncrona de información (AJAX).

En este breve documento nos centraremos sobre todo en la primera funcionalidad.

Es relativamente sencillo hacer uso de esta biblioteca y sus funcionalidades y potencia son más que aceptables. No obstante se está convirtiendo rápidamente en una de las bibliotecas Javascript más utilizadas en numerosos sitios web.

jQuery viene en un archivo .js. Existen dos variantes, la normal (jquery-1.6.4.js) y la minimizada (jquery-1.6.4.**min**.js).

La minimizada ocupa menos, por lo tanto es adecuado su uso si nos preocupa el tiempo de descarga. Esta variante no posee la mayoría de los comentarios, elimina los saltos de página, tabulaciones y espacios en blanco y utiliza identificadores cortos, todo ello para reducir su peso.

La variante normal ocupa algo más, pero por otro lado es más legible para el programador. Nosotros haremos uso de esta última.

Posteriormente cuando tengamos desarrollada y depurada la página podremos enlazar a la versión minimizada.

Podemos bajarnos la biblioteca jQuery desde su sitio oficial www.jquery.com. Una vez hecho copiaremos el archivo jquery-1.6.4.js o similar, dependiendo de la versión, en el mismo directorio donde tengamos la página web e indicamos en su cabecera que vamos a hacer uso de él.

```
<script type="text/javascript" src="jquery-1.6.4.js"></script>
```

NOTA: es posible colocar el archivo en otro directorio, normalmente se crea un directorio js donde se colocan todo el código javascript.

Las páginas que desarrollaremos tendrán una estructura similar a la siguiente:

```
<html>
<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />   <!-- iso-8859-1, cp1252 -->

<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">

$(document).ready(function() {

            alert ("Hola");


});

</script>

</head>
<body>
…
</body>
</html>
```

En la primera etiqueta script que aparece indicamos que vamos a utilizar la librería jQuery.

En la segunda etiqueta script colocaremos nuestro código javascript.

Lo más normal es que todo nuestro código vaya entre las líneas

```
$(document).ready(function() {

      // Todo nuestro código

});
```

Así incluiremos todas las operaciones que vamos a realizar cuando el documento este listo (ready). Una forma abreviada de escribir lo anterior es

```
$(function() {

      // Todo nuestro código

});
```

NOTA: el $ equivale a jQuery, es decir, lo anterior es equivalente a escribir:

```
jQuery(function() {

      // Todo nuestro código

});
```

NOTA: JavaScript distingue entre minúsculas y mayúsculas. Si en lugar de **jQuery** escribimos JQuery, el código interior no se ejecutará.

Selectores

Para indicar sobre que elemento de una página web vamos a operar jQuery proporciona unos selectores muy potentes. Son tan numerosos que aquí sólo trataremos los más utilizados.

Para seleccionar o consultar utilizamos la función **jQuery()** o más corrientemente **$().**

Dentro de los paréntesis colocamos entre comillas los identificadores de los elementos que deseamos tratar. **Se admiten los selectores CSS**. Por ejemplo:

$(“p”)	Selecciona todos los elementos p
$(“#autor)	Selecciona el único elemento con id=”autor”
$(“.gris)	Selecciona todos los elementos con class=”gris”

NOTA: En todo el documento sólo debe haber un elemento con id=”elquesea”. Los identificadores se diseñaron para identificar elementos únicos en el documento.

Ejemplo:

```
<html>
<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />   <!-- iso-8859-1, cp1252 -->

<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">
$(document).ready(function() {
    $("p").css({border: "solid 1px black", height: "100px"});

    $(".gris").css ({backgroundColor: "gray"});

    $("#autor").css({textAlign: "center"});
});

</script>

</head>
<body>
<p>Un ejemplo muy sencillo</p>

<p class="gris">Línea 1</p>
<p class="gris">Línea 2</p>

<p id="autor">JAMJ – 2012</p>
</body>
</html>
```

En la anterior página

- Seleccionamos todos los párrafos (p) y le ponemos borde y altura.
- Seleccionamos todos los elementos con clase “gris” (no tienen por que ser párrafos) y le ponemos un fondo gris.
- Seleccionamos el elemento con id “autor” (no tiene por que ser párrafo) y ponemos el texto alineado en el centro.

NOTA: La función **css( )** nos permite cambiar el estilo a los elementos seleccionados o leer el valor de una propiedad.

Otro ejemplo:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />   <!-- iso-8859-1, cp1252 -->

<script type="text/javascript">


$(document).ready(function() {

    $(".gris").html ("Linea en color gris");

    alert($("#autor").html());
});


</script>

</head>
<body>
<p>Un ejemplo muy sencillo</p>

<p class="gris">Línea 1</p>
<p class="gris">Línea 2</p>

<p id="autor">JAMJ – 2012</p>
</body>
</html>
```

Seleccionamos los elementos de la clase gris y escribimos dentro de ellos “Línea en color gris”.

Seleccionamos el elemento con id “autor” y lo mostramos en un alert.

NOTA: La función **html( )** permite escribir dentro de una etiqueta o mostrar su contenido. Existe además una función **attr( )** que permite leer y modificar atributos de las etiquetas.

Filtros

Existen muchos filtros, los más comunes son:

:first	Primer elemento
:last	Último elemento
:even	Elementos pares
:odd	Elementos impares
:eq(n)	Elemento igual a n
:lt(n)	Elementos menores que n
:gt(n)	Elementos mayores que n

NOTA: **Los elementos se numeran empezando en 0.**

Algunos ejemplos:

```
<html>

<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
    .gris { background-color: gray }
    .tan { background-color: tan }
</style>
<script type="text/javascript" src="jquery-1.6.4.js"></script>
<script type="text/javascript">

$(document).ready(function() {

    $("p:gt(0)").addClass("tan");

});

</script>
```

```
</head>

<body>

<p>Un ejemplo muy sencillo</p>
<p>Línea 1</p>
<p>Línea 2</p>
<p>Línea 3</p>
<p>Línea 4</p>

</body>
</html>
```

```
    $("p:gt(0)").addClass("tan");
```

Un ejemplo muy sencillo

Línea 1

Línea 2

Línea 3

Línea 4

```
$("p:gt(0)").removeClass("tan");
$("p:even").addClass("gris");
```

Un ejemplo muy sencillo

Línea 1

Línea 2

Línea 3

Línea 4

```
$("p:even").removeClass("gris");
$("p:eq(3)").addClass("tan");
```

Un ejemplo muy sencillo

Línea 1

Línea 2

Línea 3

Línea 4

Para formularios:

:input	Selecciona todos los elementos de un formulario incluyendo <input>, <select>, <textarea>, y <button>.
:text	
:password	
:radio	
:checkbox	
:submit	
:reset	
:file	
:image	
:hidden	

:enabled	Habilitados
:disabled	Deshabilitados
:checked	Marcados
:selected	Seleccionados

Efectos

- **hide**: Oculta un elemento.
- **show:** Muestra un elemento oculto.
- **slideDown**: Desenrolla un elemento oculto.
- **slideUp**: Enrolla un elemento.
- **fadeIn**: Hacer aparecer un elemento oculto.
- **fadeOut**: Desvanece un elemento.
- **animate**: Anima un elemento.

jQuery nos proporciona varias funciones para realizar efectos visuales en nuestra páginas. Supongamos que disponemos de la siguiente página web.

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style type="text/css">
    #mensaje { background-color: tan; font-size: 14px; border: solid 1px black; width: 100px; height: 100px;}
</style>
<script type="text/javascript" src="jquery-1.6.4.js"></script>
<script type="text/javascript">

$(document).ready(function() {
     // Nuestro código jQuery
});

</script>
</head>

<body>
<div id="mensaje">
Mensaje para hacer prueba de efectos visuales en jQuery
<div>
</body>
</html>
```

Para ocultar el mensaje escribimos:

```
$("#mensaje").hide("normal");
```

Para volver a mostrarlo:

```
$("#mensaje").show("normal");
```

Tanto la función hide() como show() reciben como parámetro la velocidad del efecto. Esta puede ser “slow”, “normal” o “fast”. También ponemos escribir una cifra sin comillas en milisegundos. Por ejemplo para el que efecto tarde 5 segundos escribimos 5000 en lugar de “normal”.

Con la función **toggle()**, nos permite ocultar el elemento si esta visible, o hacerlo visible si esta oculto. Por ejemplo, si ejecutamos:

```
for (var i=0; i<4; i++)  {
      $("#mensaje").toggle(3000);
}
```

ocultará, mostrará, ocultará y volverá a mostrar el mensaje, cada vez con una duración de 3 segundos.

Con las funciones **sliceUp()** y **sliceDown()**, podemos realizar un efecto “persiana”, donde el elemento se recoge hacia arriba hasta desaparecer y luego se desenrolla hacia abajo hasta volver a aparecer.

```
$("#mensaje").slideUp(3000);
$("#mensaje").slideDown(3000);
```

NOTA: Para que sliceDown() funcione el elemento debe de estar enrollado, es decir estar oculto.

También existe una función **slideToggle()**, similar a la función toggle().

Con las funciones **fadeOut()** y **fadeIn()**, podemos realizar un efecto "desvanecimiento", donde el elemento se desvanece hasta desaparecer y luego aparece de nuevo.

```
$("#mensaje").fadeOut(3000);
$("#mensaje").fadeIn(3000);
```

NOTA: Para que fadeIn() funcione el elemento debe de estar desvanecido, es decir estar oculto.

No existe una función fadeToggle(), pero si una función **fadeTo()**, nos permite hacer un desvanecimiento parcial. Por ejemplo:

```
$("#mensaje").fadeTo(3000, .5);
```

realizará un desvanecimiento al 50% (0.5), es decir semitransparente.

Por fin llegamos a la función estrella: la función **animate()**. Esta función te permitirá crear efectos espectaculares con muy poco trabajo. El prototipo de está función es:

animate(params, duration, callback)

Como ves, tiene 3 parámetros, aunque sólo el primero es obligatorio. Los params deben ser **propiedades CSS numéricas**. Por ejemplo:

```
$("#mensaje").animate({height: "50px", width: "200px" }, 3000);
```

cambia la altura a 50 píxeles y la anchura a 200 píxeles en 3 segundos. Observa que los parámetros los escribimos entre llaves y separados por comas.

Las propiedades CSS que se pueden animar son las propiedades con valores numéricos y más concretamente las siguientes:

height: El alto de un elemento.

width: El ancho de un elemento.

borderWidth (border-width): El ancho de los 4 bordes. También se puede hacer individualmente con borderBottomWidth (border-bottom-width), borderLeftWidth (border-left-width), borderRightWidth (border-right-width), y borderTopWidth (border-top-width).

margin: El margen de un elemento. También se puede hacer individualmente con marginBottom (margin-bottom), marginLeft (margin-left), marginRight (margin-right), y marginTop (margin-top).

padding: Relleno alrededor de un elemento.También se puede hacer individualmente con paddingBottom (padding-bottom), paddingLeft (padding-left), paddingRight (padding-right) y paddingTop (padding-top).

fontSize (font-size): El tamaño de fuente del texto.

wordSpacing (word-spacing): Incremento o decremento de espacio entre palabras

opacity: Opacidad de un elemento.

top: Margen superior. Útil cuando un elemento tiene position: absolute en su CSS.

left: Margen izquierdo. Útil cuando un elemento tiene position: absolute en su CSS.

right: Margen derecho. Útil cuando un elemento tiene position: absolute en su CSS.

bottom: Margen inferior. Útil cuando un elemento tiene position: absolute en su CSS.

Veamos algunos ejemplos:

- Cambiamos tamaño de texto y opacidad a la vez.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){

              $("p").animate({fontSize: "100px", opacity: "0.5"},3000);

        });

       </script>
    </head>
    <body>
        <p>Párrafo en movimiento. </p>
    </body>
</html>
```

- Cambiamos ancho y alto a la vez.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; height: 25px; width: 300px; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
        $(document).ready( function (){

            $("p").animate({width: "500px", height: "200px"},3000);


        });
        </script>
    </head>
    <body>
        <p>Párrafo de muestra. </p>
    </body>
</html>
```

- Movemos párrafo hacia abajo y hacia la derecha a la vez.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; height: 25px; width: 300px; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
        $(document).ready( function (){

              $("p").animate({bottom: "0px", right: "0px"},3000);

        });
        </script>
```

```
    </head>
    <body>
        <p>Párrafo en movimiento. </p>
    </body>
</html>
```

Podemos **encadenar efectos**, es decir crear una cola de efectos, si los aplicamos uno detrás de otro. Prueba los siguientes ejemplos en tu navegador para ver el resultado.

- Cambiamos el ancho y después el alto.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; height: 25px; width: 300px; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
        $(document).ready( function (){

            $("p").animate({width: "500px"},3000)
                  .animate({height: "200px"},3000);


        });
        </script>
    </head>
    <body>
        <p>Párrafo de muestra. </p>
    </body>
</html>
```

- Movemos un párrafo por la pantalla.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; height: 25px; width: 300px; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
        $(document).ready( function (){

              $("p").animate({bottom: "0px"},3000)
                    .animate({right: "0px"},3000)
                    .animate({top: "0px"},3000)
                    .animate({left: "0px"},3000);

        });
        </script>
    </head>
    <body>
        <p>Párrafo en movimiento. </p>
    </body>
</html>
```

No sólo podemos encadenar los efectos de animate, también podemos hacerlo con los otros efectos vistos en este apartado (hide, show, fadeIn, fadeOut, slideUp, slideDown). En la siguiente página hacemos cada 3 segundos lo siguiente: primero cambiamos el tamaño de texto y opacidad a la vez, después hacemos un desvanecimiento, después vuelve a aparecer, después se enrolla hacia arriba y por último se desenrolla hacia abajo.

- Varios efectos además de animate.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){
              var ms = 3000;
              $("p").animate({fontSize: "100px", opacity: "0.5"}, ms)
                    .fadeOut(ms)
                    .fadeIn(ms)
                    .slideUp(ms)
                    .slideDown(ms);
        });

       </script>
    </head>
    <body>
        <p>Párrafo en movimiento. </p>
    </body>
</html>
```

Eventos

Javascript (y jQuery) tiene un buen soporte para la **programación dirigida por eventos**.

Hasta ahora las operaciones se ejecutaban una vez que el documento estaba listo (ready). Es decir el código se ejecutaba cuando el evento ready sucedía sobre el document.

En lugar de hacer esto, es más común dejar indicado que eventos pueden suceder sobre qué elementos y asociar las instrucciones deseadas.

Por ejemplo en la siguiente página cuando pulsamos sobre el párrafo éste se enrolla hacia arriba.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            p { border: solid 1px black; position: absolute;}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){

           $('p').click (function (){
               $(this).slideUp (3000);
           });

        });

       </script>
    </head>
    <body>
        <p>Párrafo de muestra. </p>
    </body>
</html>
```

Los eventos que pueden suceder son los siguientes:

Formularios

- **focus**: El elemento ha obtenido el foco.
- **blur**: El elemento ha perdido el foco.
- **change**: El elemento ha cambiado.
- **select**: Se ha seleccionado texto de un input o un textarea.

Teclado

- keydown: Una tecla es presionada.
- keyup: Una tecla es liberada.
- **keypress: Una tecla es presionada y liberada.**

Ratón

- mousedown: El botón es presionado.
- mouseup: El botón es liberado.
- **click**: El botón es presionado y liberado.
- **dblclick**: El botón es presionado y liberado dos veces.
- **mousemove**: El ratón es movido.
- **mouseover**: El ratón entra en un elemento.
- mouseout: El ratón sale de un elemento.

Otros

- **ready:** El elemento ha terminado de cargarse (Equivale al evento load de javascript)
- **hover**: Engloba los eventos mouseover y mouseout
- **toggle**: Ejecuta distinto código en cada click.

Veamos algunos ejemplos.

- FORMULARIO: Utilizamos eventos focus y blur para comprobar cuando el usuario entra en el input y cuando sale.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){

            $('#nombre').focus (function (){
               $(this).css({backgroundColor: "wheat"});
            });

            $('#nombre').blur (function (){
               $(this).css({backgroundColor: "white"});
            });

        });

       </script>
    </head>
    <body>
        <form><input id="nombre" type="text" /></form>
    </body>
</html>
```

Tanto cuando tratamos eventos de teclado como de ratón podemos obtener cierta información del evento. Si llamamos al evento con el nombre **event**, entonces:

- event.charCode nos da el código de caracteres imprimibles
- event.keyCode nos da el código para teclas especiales: Esc, Fn, AvPag, …
- event.clientX nos da la posición x del cursor del ratón dentro de la página
- event.clientY nos da la posición y del cursor del ratón dentro de la página

Existe otro tipo de información acerca del evento a la que podemos acceder, pero nosotros sólo vamos a utilizar la anterior.

- TECLADO: Comprobamos que carácter pulsa el usuario y mostramos su valor ASCII decimal.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){

            var t = $('#texto');
            $('#nombre').keypress (function (event){
                    t.html (event.charCode);
            });

        });

       </script>
    </head>
    <body>
        <form><input id="nombre" type="text" /></form>
        <div id="texto"></div>
    </body>
</html>
```

- RATÓN: Comprobamos la posición del ratón cuando se mueve por una división.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            #area { border: solid 1px black; width: 200px; height: 200px}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">

        $(document).ready( function (){

            var t = $('#texto');
            $('#area').mousemove (function (event){
                    t.html (event.clientX + ' ' + event.clientY);
            });

        });

        </script>
    </head>
    <body>
        <div id="area"></div>
        <div id="texto"></div>
    </body>
</html>
```

- RATÓN: Mostramos un mensaje flotante cuando pasamos por encima de una división.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            #area { border: solid 1px black; width: 200px; height: 200px}
            #mensaje {border: solid 1px gray; background-color: wheat; display: none; position: absolute}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
```

```
        $(document).ready( function (){

            var t = $('#texto');
            $('#area').mouseover (function (){
                  $('#mensaje').show();
            });
            $('#area').mouseout (function (){
                  $('#mensaje').hide();
            });
            $('#area').mousemove (function (event){
                   $('#mensaje').css({left: event.clientX+5, top: event.clientY+5});
            });

        });

        </script>
    </head>
    <body>
        <div id="area"></div>
        <div id="mensaje">Esto es un mensaje de prueba</div>
    </body>
</html>
```

El mensaje está oculto en un principio. Cuando el ratón entra (mouseover) en la división entonces el mensaje se hace visible. Cuando sale (mouseout) se vuelve a ocultar.

El evento **hover** hace esto mismo pero de una forma más sencilla.

- RATÓN: El mismo ejemplo anterior con hover.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
            #area { border: solid 1px black; width: 200px; height: 200px}
            #mensaje {border: solid 1px gray; background-color: wheat; display: none; position: absolute}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
```

```
        $(document).ready( function (){

            var t = $('#texto');
            $('#area').hover (
               function (){  $('#mensaje').show(); } ,
               function (){  $('#mensaje').hide(); }
            );

            $('#area').mousemove (function (event){
                 $('#mensaje').css({left: event.clientX+5, top: event.clientY+5});
            });

        });

       </script>
    </head>
    <body>
        <div id="area"></div>
        <div id="mensaje">Esto es un mensaje de prueba</div>
        <div id="texto"></div>
    </body>
</html>
```

En este caso el evento hover recibe dos parámetros en forma de función separados por una coma. La primera función para cuando el ratón entra en la división y la segunda para cuando sale.

Al evento **toggle**, se le pueden pasar como parámetros varias funciones y cada vez que se haga clic con el ratón se ejecutará una de ellas según el orden especificado. Cuando se llegué a la última función, volverá a ejecutarse la primera. Y así sucesivamente.

- RATÓN: Cambiamos una imagen cada vez que pulsamos sobre ella (cada vez una imagen distinta).

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title></title>
        <style type="text/css">
            * {margin:0; padding: 0}
        </style>
        <script type="text/javascript" src="jquery-1.6.4.js"></script>
        <script type="text/javascript">
```

```
        $(document).ready( function (){

            $('#imagen').toggle (
               function (){  $('#imagen').attr({src: "foto1.png"}); } ,
               function (){  $('#imagen').attr({src: "foto2.png"}); } ,
               function (){  $('#imagen').attr({src: "foto3.png"}); }
            );

        });

       </script>
    </head>
    <body>
        <img id="imagen" src="foto3.png" />
    </body>
</html>
```

Comunicación asíncrona con un servidor

Hace unos años el único tipo de comunicación de datos entre un servidor y un cliente http era la comunicación síncrona. Esto significa que el cliente recopilaba los datos, normalmente a través de un formulario, estos eran enviados mediante el método get o post al servidor, se procesaban en el servidor y, cuando el servidor tenía el resultado, se enviaba la respuesta al cliente, que en dicho instante mostraba un mensaje en una nueva página.

Este procedimiento a menudo provocaba una experiencia del usuario no del todo satisfactoria, puesto que las esperas se hacían evidentes.

Una forma de mejorar esto es utilizar comunicación asíncrona con el servidor. En este caso los datos pueden enviarse en cualquier momento al servidor, no es necesario realizar un submit. Además se hace de forma transparente al usuario, es decir, el usuario no tiene por que ser consciente de que esto está ocurriendo. Además la respuesta del servidor puede asimismo mostrarse dentro de la misma página si tener que refrescar o cambiar de página.

Esta técnica se conoce actualmente como AJAX y es ampliamente utilizada en la Web 2.0.

AJAX son las siglas de

- Asynchronous
- Javascript
- And
- XML

Principalmente se utiliza Javascript en el lado del cliente para gestionar toda la transmisión de datos entre el usuario y el servidor, y XML para el formato de los datos intercambiados entre cliente y servidor.

Sin embargo la comunicación asíncrona con el servidor no tiene porque hacerse en formato XML, de hecho pueden utilizarse otros formatos como HTML, XHTML, JSON, texto plano, …

Existen numerosas funciones en jQuery para realizar comunicación asíncrona con el servidor:

$.get()

$.post()

$.ajax()

y otras.

No obstante, nosotros utilizaremos la función **load()**, que es muy sencilla de utilizar y válida para nuestros fines.

Recibiendo datos del servidor

Es posible solicitar datos al servidor. Estos pueden estar en distintos formatos: texto plano, HTML, XML, …

- Ejemplo: solicitando un archivo mensaje.txt al servidor.

```
<html>
<head>
<title>Cargando un archivo de texto</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">
  $(document).ready(function(){
    $('pre').load('mensaje.txt');
  });
</script>
</head>

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

NOTA: El archivo **mensaje.txt** debe existir en el servidor. Por ejemplo, el contenido de dicho archivo será:

```
Este es el contenido de un archivo de texto plano.

Una línea.
Otra línea.
```

- Ejemplo: solicitando un archivo dias.html al servidor.

```
<html>
<head>
<title>Cargando un archivo html</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">
  $(document).ready(function(){
     $('div').load('dias.html');
  });
</script>
</head>

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

NOTA: El archivo **dias.html** debe existir en el servidor. Por ejemplo, el contenido de dicho archivo será:

```
<html>
<body>
<div id="lunes">El Lunes es el primer día de la semana.</div>
<div id="martes">El Martes es el segundo día de la semana.</div>
<div id="miercoles">El Miércoles es el tercer díade la semana.</div>
<div id="jueves">El Jueves es el cuarto día de la semana.</div>
<div id="viernes">El Viernes es el quinto día de la semana.</div>
<div id="sabado">El Sábado es el primer día del fin de semana.</div>
<div id="domingo">El Domingo es el segundo día del fin de semana.</div>
</body>
</html>
```

Si en lugar de cargar la página completa queremos cargar sólo una parte, digamos el contenido del domingo, escribimos

```
     $('div').load('dias.html #domingo');
```

en lugar de

```
     $('div').load('dias.html');
```

Enviando y recibiendo datos del servidor

En los ejemplos anteriores hemos cargado contenido estático de archivos txt o html. Es posible cargar contenido producido dinámicamente en el servidor, por ejemplo un script php. Este script puede realizar ciertas operaciones como consultar una base de datos o realizar cálculos y enviar los resultados al cliente.

Normalmente el cliente envía previamente cierta información sobre la que desea que el servidor realice las operaciones.

La función load() puede tener hasta 3 parámetros.

load (url, data, response())

url: dirección del archivo o script solicitado.

data: datos enviados al servidor en forma de diccionario

response(): función que se ejecutará cuando se reciba la respuesta del servidor.

- Ejemplo: solicitando información a dias.php en el servidor.

```
<html>
<head>
<title>Solicitando información a un archivo PHP</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>

<script type="text/javascript">
  $(document).ready(function(){
    $('p').click ( function (){
        $('div').load('dias.php',  {'dia': $(this).text()});
    });

  });
</script>
</head>

<body>
<p>lunes</p>
<p>martes</p>
<p>miercoles</p>
<p>jueves</p>
<p>viernes</p>
<p>sabado</p>
<p>domingo</p>
```

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

NOTA: El archivo **dias.php** debe existir en el servidor. Por ejemplo, el contenido de dicho archivo será:

```
<?php

if ($_REQUEST['dia']=='lunes')
    echo "El Lunes es el primer día de la semana.";
else if ($_REQUEST['dia']=='martes')
    echo "El Martes es el segundo día de la semana.";
else if ($_REQUEST['dia']=='miercoles')
    echo "El Miércoles es el tercer día de la semana.";
else if ($_REQUEST['dia']=='jueves')
    echo "El Jueves es el cuarto día de la semana.";
else if ($_REQUEST['dia']=='viernes')
    echo "El Viernes es el quinto día de la semana.";
else if ($_REQUEST['dia']=='sabado')
    echo "El Sábado es el primer día del fin de semana.";
else if ($_REQUEST['dia']=='domingo')
    echo "El Domingo es el segundo día del fin de semana.";
else
    echo "No sé en que día estoy.";

?>
```

- Ejemplo: solicitando información a dias.php en el servidor a través de una selección en el cliente.

```
<html>
<head>
<title>Solicitando información a un archivo PHP</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $('select').change ( function (){
        $('div').hide();
        $('div').load('dias.php',  {'dia': f.dia.value}, $('div').fadeIn());
    });
```

```
});
</script>
</head>
<body>
    <form name="f">
    <select name='dia'>
        <option>lunes</option>
        <option>martes</option>
        <option>miercoles</option>
        <option>jueves</option>
        <option>viernes</option>
        <option>sabado</option>
        <option>domingo</option>
    </select>
    </form>

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

NOTA: El archivo **dias.php** debe existir en el servidor. Su contenido no varía respecto al anterior ejemplo.

En este último ejemplo hemos utilizado los 3 parámetros de la función **load (url, data, response())**

url: **'dias.php'**

data: **{'dia': f.dia.value}**

response(): **$('div').fadeIn()**

Si los datos que debemos enviar al servidor son numerosos, porque el formulario posea muchos campos, es posible enviarlos todos juntos haciendo uso de la función **serialize().**

En lugar de

```
<html>
<head>
<title>Solicitando información a un archivo PHP</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $(':input').change ( function (){
        $('div').hide();
        $('div').load('dias.php',  {'dia': f.dia.value, 'mes': f.mes.value, 'anno': f.anno.value}, $('div').fadeIn());
    });

});
</script>
</head>
<body>
    <form name="f">
    Día:<select name='dia'>
        <option>lunes</option>
        <option>martes</option>
        <option>miercoles</option>
        <option>jueves</option>
        <option>viernes</option>
        <option>sabado</option>
        <option>domingo</option>
    </select><br/>
    Mes: <input name="mes" /><br/>
    Año: <input name="anno" /><br />
    </form>

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

podemos hacer

```
<html>
<head>
<title>Solicitando información a un archivo PHP</title>
<script type="text/javascript" src="jquery-1.6.4.js"></script>
<script type="text/javascript">
$(document).ready(function(){
    $(':input').change ( function (){
        $('div').hide();
        $('div').load('dias.php', $('form').serialize(), $('div').fadeIn());
    });

});
</script>
</head>
<body>
    <form name="f">
    Día:<select name='dia'>
        <option>lunes</option>
        <option>martes</option>
        <option>miercoles</option>
        <option>jueves</option>
        <option>viernes</option>
        <option>sabado</option>
        <option>domingo</option>
    </select><br/>
    Mes: <input name="mes" /><br/>
    Año: <input name="anno" /><br />
    </form>

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

En este caso los datos

```
 {'dia': f.dia.value, 'mes': f.mes.value, 'anno': f.anno.value}
```

pueden expresarse como

```
$('form').serialize()
```

NOTA: El archivo **dias.php** debe existir en el servidor. En este caso tiene el siguiente contenido:

```
<?php
if ($_REQUEST['dia']=='lunes')
    echo "El Lunes es el primer día de la semana.";
else if ($_REQUEST['dia']=='martes')
    echo "El Martes es el segundo día de la semana.";
else if ($_REQUEST['dia']=='miercoles')
    echo "El Miércoles es el tercer día de la semana.";
else if ($_REQUEST['dia']=='jueves')
    echo "El Jueves es el cuarto día de la semana.";
else if ($_REQUEST['dia']=='viernes')
    echo "El Viernes es el quinto día de la semana.";
else if ($_REQUEST['dia']=='sabado')
    echo "El Sábado es el primer día del fin de semana.";
else if ($_REQUEST['dia']=='domingo')
    echo "El Domingo es el segundo día del fin de semana.";
else
    echo "No sé en que día estoy.";


echo "<br/>Has introducido el mes $_REQUEST[mes]";
echo "<br/>Y has introducido el año $_REQUEST[anno]";

?>
```

La función $.ajax()

Existe una función con más posibilidades que load(). Su forma es:

`$.ajax(parámetros)`

Esta función lanza una petición AJAX con los parámetros indicados. Devuelve el objeto XMLHttpRequest creado.

Los parámetros están formados por un diccionario con distintos elementos. Dependiendo de la situación y el nivel de detalle de control de la conexión podemos utilizar más o menos de ellos.

```
$.ajax({
async:          ¿Petición asíncrona? (por defecto : true )
url:            URL a solicitar
data:           Datos a enviar al servidor {nombre:valor, nombre:valor}
complete:       Función llamada al completar la petición
success:        Función llamada en caso de éxito. Parámetros: datos, mensaje de estado
error:          Función llamada en caso de error. Parámetros: XMLHttpRequest, mensaje de Error, Excepción
beforeSend:     Función a invocar antes de la llamada. Recibe como parámetro el objeto XMLHttpRequest
timeout:        Tiempo de espera máximo en ms para la petición
username:       Nombre de usuario (para autenticación)
password:       Clave de usuario (para autenticación)
cache:          ¿Permitir el uso de la cache? (por defecto: true, salvo para dataType="script")
dataType:       Tipo de datos: "xml", "html", "script", "json", "text", "jsonp"
contentType:    Tipo de datos enviados. Por defecto "application/x-www-form-urlencoded"
})
```

Por ejemplo, en lugar de:

```
$(':input').change ( function (){
        $('div').hide();
        $('div').load('dias.php',   $('form').serialize(),  $('div').fadeIn());
    });
});
```

podíamos haber escrito:

```
$(':input').change ( function (){
        $.ajax ({
                url:     'dias.php',
                data:    $('form').serialize(),
                success: function (msg) {  $('div').hide().html(msg).fadeIn(); }
        });
});
```