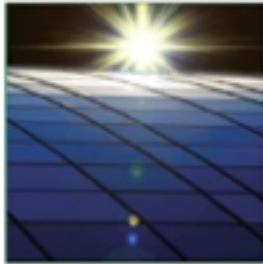


04

# Programación Web



AJAX y jQUERY



Apuntes realizados para el módulo de Desarrollo Aplicaciones en Entorno Cliente.

Profesor: Txema Serrano Sánchez

Curso 2020-2021.

Noviembre de 2020.



## ● Objetivos

---

- En esta unidad realizaremos una introducción a Ajax, y veremos como podemos mejorar la interacción que se produce entre los usuarios con las aplicaciones web mediante AJAX, evitando que se recargue constantemente la página, ya que este intercambio de información con el servidor se va a realizar en un segundo plano.
- También nos familiarizaremos con la nomenclatura de JSON, que nos será necesaria conocer para poder realizar correctamente nuestras aplicaciones en AJAX.
- Veremos tres aspectos más avanzados de JavaScript, que necesitaremos dominar para realizar correctamente nuestras aplicaciones Ajax. Igualmente veremos como tratar las excepciones y estudiaremos la reflexión.
- Se expondrá Veremos un ejemplo de una aplicación Ajax, montando un pequeño servidor en nuestro propio equipo.
- Descubriremos jQuery y aprenderemos a utilizar esta librería desde el principio. Veremos como conseguirla, como implementarla y añadirla a nuestras páginas webs.
- Nos iniciaremos en el uso de esta librería, usándola con JavaScript. Aprenderemos a seleccionar un elemento mediante su Id, por su tipo de elemento y descubriremos todos sus selectores.
- Veremos los métodos más importantes del objeto jQuery y los implementaremos cada uno de ellos en un ejemplo totalmente funcional.
- Y por último, conoceremos los eventos más importantes de jQuery y los trataremos también en ejemplos prácticos.

## ● Introducción

---

Ajax nos va a permitir mejorar la experiencia de usuario en el uso de aplicaciones web con mucha carga de peticiones al servidor, haciendo su uso mucho menos molesto.

Hasta este tema hemos visto que en algunas ocasiones, realizar según que funciones con JavaScript podía ser una acción costosa y complicada.

En esta unidad descubriremos a través de la librería de jQuery, que se pueden simplificar mucho las operaciones, ya que nos facilita muchos métodos útiles para realizar esto de una forma muy sencilla y entendible, con unos niveles básicos de programación y de diseño de páginas webs.

## 4.1. Introducción a AJAX

La primera vez que el término AJAX vio la luz, fue por el año 2005, en un artículo publicado por Jesse James Garrett. Hasta ese momento, no había un término normalizado para este tipo de aplicación web que estaba surgiendo.

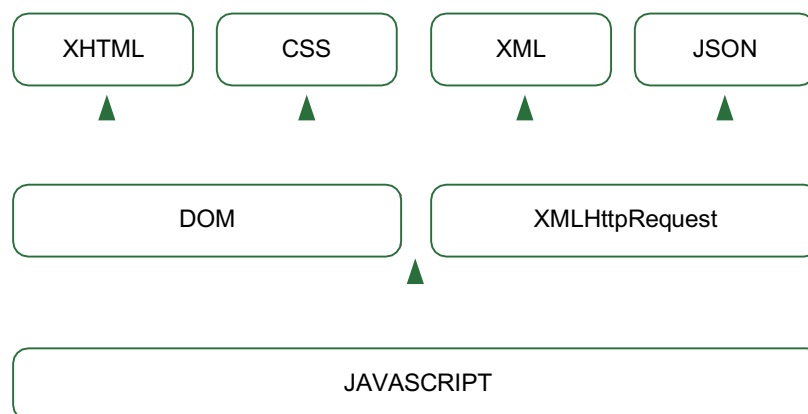


Definición

El término **AJAX** es un acrónimo de *Asynchronous JavaScript + XML*, traducido como "*JavaScript asíncrono + XML*". Su definición podría ser: "Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de formas nuevas y sorprendentes."

Las diferentes tecnologías que componen AJAX son:

- XHTML y CSS, con la que crearemos una presentación basada en estándares.
- DOM, para interactuar y manipular dinámicamente la presentación.
- XML, XSLT y JSON, para intercambiar y manipular la información.
- XMLHttpRequest, que sirve para el intercambio asíncrono de información.
- JavaScript, que es la que une todas las demás tecnologías.



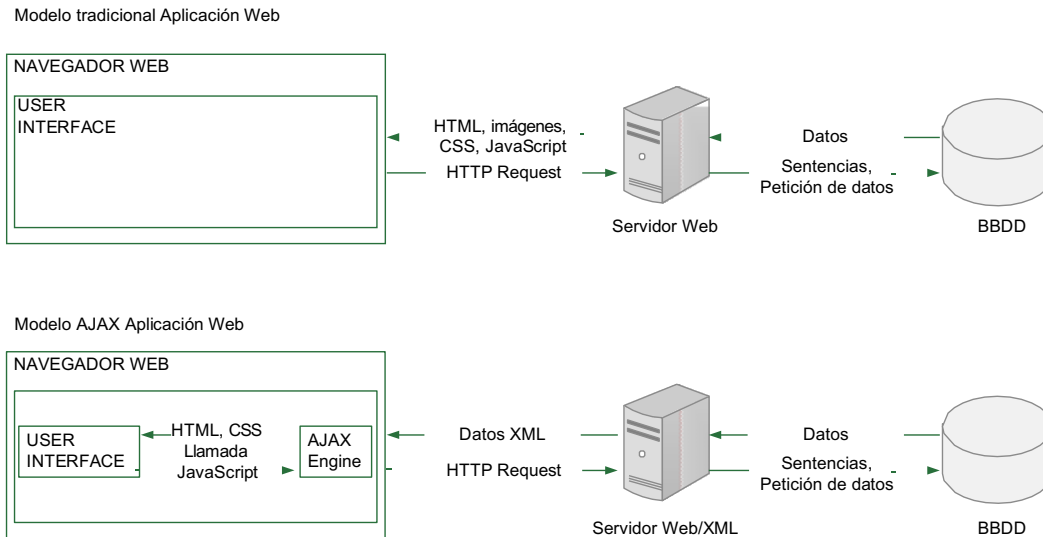
**Figura 4.1. Relación entre las tecnologías agrupadas de AJAX.**

Para poder desarrollar aplicaciones AJAX necesitamos un conocimiento de todas y cada una de las tecnologías anteriores.

En las aplicaciones web tradicionales, cualquier acción del usuario en la página desencadena una llamada al servidor. Cuando se procesa dicha petición, el servidor devuelve al navegador del usuario una nueva página HTML.

En el esquema siguiente, la imagen de arriba muestra el modelo tradicional de las aplicaciones web.

La imagen de abajo muestra el nuevo modelo propuesto por AJAX:



**Figura 4.2. Comparación gráfica de modelos de aplicación web.**

Con la técnica tradicional, como se realizan continuamente peticiones al servidor, los usuarios deben de esperar a que se recarguen las páginas con los cambios solicitados, su uso se convierte en algo molesto.

Podemos mejorar esta interacción del usuario con la aplicación mediante AJAX, evitando que se recargue constantemente la página, ya que este intercambio de información con el servidor se va a realizar en un segundo plano.

Conseguiremos eliminar la recarga constante de páginas creando un elemento intermedio entre usuario y servidor. Esta nueva capa intermedia de AJAX va a mejorar la respuesta de la aplicación, por lo que el usuario nunca se encontrará con una ventana del navegador vacía, a la espera de la respuesta del servidor.

La diferencia más importante entre una tradicional y una aplicación web creada con AJAX, va a ser que en las segundas vamos a sustituir las peticiones HTTP al servidor, por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no van a requerir intervención del servidor, por lo que la respuesta será inmediata. Si dicha petición del usuario, sí requiere una respuesta del servidor, esta se realizará de forma asíncrona mediante AJAX. En este caso, el usuario no verá interrumpida la interacción, ya sea por recargas de página o por largas esperas por la respuesta del servidor.

Desde que apareció AJAX, existen cientos de aplicaciones web que:

- Pueden sustituir casi completamente a otras técnicas como Flash.
- Pueden llegar a sustituir a las aplicaciones de escritorio.



## 4.2. Notación JSON

En este punto vamos a familiarizarnos con la nomenclatura de JSON, que nos será necesaria conocer para poder realizar correctamente nuestras aplicaciones en AJAX.




---

**JSON** (*JavaScript Object Notation*) es el formato sencillo para el intercambio de información. Este formato nos permitirá representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto.

---

Es un mecanismo definido en los fundamentos básicos del lenguaje y es una de las características principales de JavaScript.

Es una alternativa al formato XML, ya que aparte de ser mucho más conciso, es más fácil de leer y escribir. Debemos decir que, sí que es verdad que XML es técnicamente superior porque es un lenguaje de marcado, y JSON es simplemente un formato para intercambiar datos.




---

La especificación completa de JSON se puede consultar en RFC 4627 y su tipo MIME oficial es application/json.

---

Como ya sabemos, la notación tradicional de un Array es la siguiente:

```
<script type="text/javascript">
  var modulos = new Array();
  modulos[0] = "Lector RSS";
  modulos[1] = "Gestor email";
  modulos[2] = "Agenda";
  modulos[3] = "Buscador";
  modulos[4] = "Enlaces";
</script>
```

En JSON, para crear un Array normal, indicaremos sus valores separados por comas y encerrados entre corchetes. El ejemplo anterior quedaría de la siguiente manera utilizando notación JSON:

```
<script type="text/javascript">
  var modulos = ["Lector RSS", "Gestor email", "Agenda", "Buscador",
  "Enlaces"];
</script>
```

Recordamos también la forma de definir los Arrays asociativos, que sigue siendo igual de tediosa que la de los Arrays normales:

```
<script type="text/javascript">
  var modulos = new Array();
  modulos.titulos = new Array();
  modulos.titulos['rss'] = "Lector RSS";
  modulos.titulos['email'] = "Gestor de email";
  modulos.titulos['agenda'] = "Agenda";
</script>
```

También podemos utilizar la notación de puntos para abreviar un poco en su definición:

```
<script type="text/javascript">
  var modulos = new Array();
  modulos.titulos = new Array();
  modulos.titulos.rss = "Lector RSS";
  modulos.titulos.email = "Gestor de email";
  modulos.titulos.agenda = "Agenda";
</script>
```

Con la notación JSON podemos definir los Arrays asociativos de una forma más concisa. El ejemplo anterior quedaría de la siguiente manera usando JSON:

```
<script type="text/javascript">
  var modulos = new Array();
  modulos.titulos = { rss: "Lector RSS", email: "Gestor de email",
agenda: "Agenda" };
</script>
```

Esta notación JSON para este tipo de Arrays se compone de tres partes:

- Los contenidos se encierran entre llaves ({ y })
- Los elementos se separan mediante una coma (,)
- La clave y el valor de cada elemento se separan mediante dos puntos (:)

Si la clave no tiene espacios en blanco, podemos prescindir de las comillas que encierran sus contenidos. Sin embargo, serán obligatorias cuando las claves contengan espacios en blanco:

```
<script type="text/javascript">
  var titulosModulos = { "Lector RSS": "rss", "Gestor de email":
"email", "Agenda": "agenda" };
</script>
```

Como ya sabéis, JavaScript ignora los espacios en blanco sobrantes, por lo tanto, será posible reordenar las claves y valores para que se muestren más claramente en el código fuente de la aplicación. Veamos esto en el ejemplo anterior:

```
<script type="text/javascript">
  var titulos = {
    rss: "Lector RSS",
    email: "Gestor de email",
```

```

        agenda: "Agenda"
    };
</script>

```

Combinando la notación de los Arrays simples y asociativos, podemos construir objetos muy complejos de una manera sencilla. Con la tradicional, un objeto complejo lo crearíamos de la siguiente manera:

```

<script type="text/javascript">
    var modulo = new Object();
    modulo.titulo = "Lector RSS";
    modulo.objetoInicial = new Object();
    modulo.objetoInicial.estado = 1;
    modulo.objetoInicial.publico = 0;
    modulo.objetoInicial.nombre = "Modulo_RSS";
    modulo.objetoInicial.datos = new Object();
</script>

```

Pero utilizando únicamente JSON, será posible rescribir el anterior ejemplo de una manera mucho más concisa:

```

    var modulo = {
        titulo: "Lector RSS",
        objetoInicial: { estado: 1, publico: 0, nombre: "Modulo RSS",
datos: {} }
    };

```

Los objetos se pueden definir en forma de pares clave/valor separados por comas y encerrados entre llaves. Para crear objetos vacíos, utilizaremos un par de llaves sin contenido en su interior {}.

Vamos a ver la notación JSON genérica con la que crearemos arrays y objetos:

- Arrays:

```
var array = [valor1, valor2, valor3, ..., valorN];
```

- Objetos:

```
var objeto = { clave1: valor1, clave2: valor2, clave3: valor3, ...,
claveN: valorN };
```

Esta notación la podemos combinar para crear Arrays de objetos, objetos con Arrays, objetos con objetos y arrays, etc.

Con lo comentado hasta ahora deducimos que la forma habitual para definir los objetos en JavaScript se basa en el siguiente modelo creado con la notación JSON:

```

<script type="text/javascript">
    var objeto = {
        "propiedad1": valor_simple_1,
        "propiedad2": valor_simple_2,
        "propiedad3": [array1_valor1, array1_valor2],
        "propiedad4": { "propiedad anidada": valor },
        "metodo1": nombre_funcion_externa,
        "metodo2": function() { ... },
        "metodo3": function() { ... },
    };

```

```
        "metodo4": function() { ... }  
    };  
</script>
```

Aunque también podemos usar de forma simultánea la notación tradicional de JavaScript y la notación JSON:

```
<script type="text/javascript">  
    var libro = new Object();  
    libro.numeroPaginas = 150;  
    libro.autores = [{ id: 50 }, { id: 67}];  
</script>
```

Y usando la notación tradicional el ejemplo anterior quedaría de la siguiente manera:

```
<script type="text/javascript">  
    var libro = { numeroPaginas: 150 };  
    libro.autores = new Array();  
    libro.autores[0] = new Object();  
    libro.autores[0].id = 50;  
    libro.autores[1] = new Object();  
    libro.autores[1].id = 67;  
</script>
```

Y utilizando exclusivamente la notación JSON, quedaría de la siguiente manera:

```
<script type="text/javascript">  
    var libro = { numeroPaginas: 150, autores: [{ id: 50 }, { id: 67}] };  
</script>
```

## 4.3. Clases en JavaScript

En este punto, vamos a ver conceptos más avanzados de JavaScript, que no se vieron en la unidad 1 de este mismo temario.

Si recordáis de la unidad 1, lo que se ha visto hasta ahora respecto a los Objetos, es que son una colección de propiedades y métodos definidos para cada objeto individual.

En cambio, si tenéis conocimientos de programación, sabréis que en la programación orientada a objetos, el concepto fundamental es el de **clase**.

Los objetos son creados a partir de la definición de las clases, y esta es la forma habitual de trabajar en muchos lenguajes de programación como Java o C++. Sin embargo, JavaScript no permite crearlas de forma similar a la de estos lenguajes. Como ejemplo, la palabra reservada “class”, de momento está reservada para futuras versiones de JavaScript.

La forma de utilizar estas clases en JavaScript será haciéndolo con unos elementos parecidos llamados **pseudoclases**. Usaremos los conceptos de funciones constructoras y el prototype de los objetos para poder realizar la simulación de estas.

En los siguientes puntos vamos a estudiar cada uno de estos conceptos de una forma más extensa

### 4.3.1. Funciones constructoras

En Programación Orientada a Objetos (**POO** a partir de este momento) existe el concepto de constructor.



---

Un **constructor** es un método de las clases, el cual es llamado automáticamente cuando se crea un objeto de esa clase.

---

En cambio, este concepto de constructor no existe en JavaScript. Cuando definamos una clase, no podremos incluir dichos constructores, pero podremos emular su funcionamiento usando funciones.

Lo mejor es que esto lo veamos con un pequeño ejemplo de código. En el siguiente ejemplo, vamos a crear un objeto genérico y un objeto de tipo Array:

```
var elObjeto = new Object();  
var elArray = new Array(5);
```

Si nos fijamos hemos utilizado la palabra reservada `new` y el nombre del tipo de objeto que se quiere crear. Realmente va a ser el nombre de una función que se ejecuta para crear el nuevo objeto. Además, como recordaréis de la unidad 1, como estamos trabajando con funciones, podemos incluir parámetros en la creación del objeto.

Como hemos dicho antes, vamos a utilizar funciones para simular los constructores de objetos, y por ello las denominaremos "funciones constructoras".

En el siguiente ejemplo vamos a crear una función que llamaremos `Factura`, que usaremos para crear los objetos `Factura`:

```
function Factura(idFactura, idCliente) {  
    this.idFactura = idFactura;  
    this.idCliente = idCliente;  
}
```

Normalmente cuando creamos un objeto, le pasamos al constructor de la clase una serie de valores con las que vamos a inicializar algunas de sus propiedades. En JavaScript también se utiliza, pero realizándolo de una forma diferente. En este caso, inicializamos las propiedades de cada objeto usando la palabra reservada `this` dentro de la función constructora, en la cual definiremos todos los parámetros que necesitemos para construir los nuevos objetos y posteriormente usarlos para la inicialización de las propiedades. En el ejemplo de código anterior, hemos inicializado el objeto `Factura` mediante el identificador de factura (`idFactura`) y el identificador de cliente (`idCliente`).

Una vez definida la función constructora, ya podemos crear un objeto de tipo `Factura` y realizar la simulación del funcionamiento de un constructor:

```
var laFactura = new Factura(3, 7);
```

En el ejemplo anterior, definimos el objeto `laFactura` es de tipo `Factura`, con sus propiedades y métodos, pudiendo acceder a ellos usando la notación de puntos habitual:

```
alert("cliente = " + laFactura.idCliente + ", factura = " +  
laFactura.idFactura);
```



Recuerde

---

Lo normal es que este tipo de funciones no devuelvan ningún valor y únicamente sirvan para definir las propiedades y los métodos del nuevo objeto.

---

### 4.3.2. Prototype

Dentro de las funciones constructoras, además de establecer las propiedades del objeto, también se pueden definir sus métodos.

Siguiendo con el ejemplo de la Factura, vamos a insertar sus métodos a la hora de crearlo:

```
function Factura(idFactura, idCliente) {  
    //Declaramos las propiedades  
    this.idFactura = idFactura;  
    this.idCliente = idCliente;  
  
    //Declaramos los metodos  
    this.muestraCliente = function () {  
        alert(this.idCliente);  
    }  
  
    this.muestraId = function () {  
        alert(this.idFactura);  
    }  
}
```

Con el código anterior, hemos definido una pseudoclase mediante la función constructora, y ya podemos crear objetos de ese tipo. Veamos como crear dos objetos diferentes y como se emplean sus métodos:

```
//creamos el objeto laFactura  
var laFactura = new Factura(3, 7);  
//mediante los metodos, accedemos a sus propiedades  
laFactura.muestraCliente();  
  
var otraFactura = new Factura(5, 4);  
otraFactura.muestraId();
```

Esta técnica usada hasta aquí, la de incluir los métodos de los objetos como funciones dentro de la propia función constructora funciona correctamente, pero tiene un inconveniente.

Si nos fijamos en el ejemplo anterior, cada vez que instanciamos un objeto, tenemos que definir tantas funciones como métodos incluye el objeto. En nuestro caso, hemos creado por cada objeto creado, las funciones muestraCliente() y muestraId(). Esto produce una ralentización en el rendimiento y un excesivo consumo de recursos, por lo que supondrá un inconveniente en grandes aplicaciones realizadas con JavaScript.

Por suerte, JavaScript tiene una propiedad, que además de no estar presente en otros lenguajes de programación, nos soluciona este problema. Es la propiedad **prototype** y es una de las características más poderosas de JavaScript.

Debemos saber, que todos los objetos de JavaScript van a incluir una referencia interna a otro objeto llamado prototype o "prototipo". Por lo tanto, cualquiera de las

propiedades o métodos que tenga este objeto prototipo, estará incluida automáticamente en el objeto original.

Realmente, es como si automáticamente heredara las propiedades y métodos de otro objeto llamado prototype. Por lo tanto, si se modifica el objeto prototipo o le añadimos más funcionalidades nuevas, los objetos que estén fabricados con ese molde, también tendrán esas características nuevas.

Lo normal es que los métodos no varíen de un objeto a otro del mismo tipo, por lo que evitaremos el problema de rendimiento añadiendo dichos métodos al prototipo a partir del cual se crean los objetos.

Siguiendo el ejemplo anterior, podemos rescribir utilizando el objeto prototype:

```
function Factura(idFactura, idCliente) {  
    this.idFactura = idFactura;  
    this.idCliente = idCliente;  
}  
  
Factura.prototype.muestraCliente = function () {  
    alert(this.idCliente);  
}  
  
Factura.prototype.muestraId = function () {  
    alert(this.idFactura);  
}
```

Para que podamos incluir los métodos en el prototipo de un objeto, vamos a utilizar la propiedad prototype del objeto. Vemos como se han añadido los dos métodos en su prototipo. Así, todos los objetos que creamos con esta función constructora incluirán por defecto estos dos métodos. De esta manera, no tenemos que crear dos nuevas funciones por cada objeto, sino que hemos definido dos funciones para todos los objetos creados.



---

Tened en cuenta que en el prototype de un objeto sólo debemos añadir los elementos comunes para todos los objetos. Lo normal será añadir los métodos y las constantes (son las propiedades cuyo valor no varía durante la ejecución de la aplicación). Pero las propiedades del objeto deben permanecer en la función constructora, de esta manera cada objeto diferente podrá tener un valor distinto en esas propiedades.

---



Debemos tener cuidado con la propiedad prototype, y es que podemos rescribir un método o una propiedad sin darnos cuenta. Veámoslo más claro con un ejemplo:

```
Factura.prototype.iva = 16;
var laFactura = new Factura(3, 7); // laFactura.iva = 16

Factura.prototype.iva = 7;
var otraFactura = new Factura(5, 4);
// Ahora, los objetos laFactura.iva y otraFactura.iva = 7
```

En el ejemplo anterior, al objeto de tipo Factura le hemos definido una propiedad llamada iva, a la que le damos el valor de 16. El primer objeto creado, laFactura, dispone de una propiedad llamada iva cuyo valor es 16. En la siguiente línea de código, modificamos el prototipo del objeto y establecemos un nuevo valor en la propiedad iva.

En la siguiente instrucción, creamos otro objeto de tipo Factura, llamado otraFactura. En estos momentos, la propiedad iva de este objeto creado vale 7. Pero también el valor de la propiedad del objeto laFactura ha cambiado y ahora vale 7 y no 16. No es una cosa normal, modificar el prototipo en tiempo de ejecución, pero debemos tener en cuenta que es posible modificarlo de forma accidental.

### Ejemplo práctico

---

Siguiendo con el ejemplo del objeto Factura, vamos a crear también dos pseudoclases, que serán Cliente y Elemento.

Cliente va a guardar los datos del cliente. Veamos como declararla:

```
function Cliente(nombre, direccion, telefono, nif) {
    this.nombre = nombre;
    this.direccion = direccion;
    this.telefono = telefono;
    this.nif = nif;
}
```

La pseudoclase Elementos es un Array simple que va a contener las pseudoclases de todos los elementos que forman la factura. Veamos su código para entenderlo mejor:

```
// Definición de la clase Elemento
function Elemento(descripcion, cantidad, precio) {
    this.descripcion = descripcion;
    this.cantidad = cantidad;
    this.precio = precio;
}
```

La pseudoclase Factura vendrá definida por la siguiente declaración:

```
// Definición de la clase Factura
function Factura(cliente, elementos) {
  this.cliente = cliente;
  this.elementos = elementos;
  this.informacion = {
    baseImponible: 0,
    iva: 16,
    total: 0,
    formaPago: "contado"
  };
};
```

Veamos ahora como usar el prototype, añadiendo datos a la clase Factura, que se suponen que no cambiarán durante la ejecución del script

```
// La información de la empresa que emite la factura se
// añade al prototype porque se supone que no cambia
Factura.prototype.empresa = {
  nombre: "Nombre de la empresa",
  direccion: "Direccion de la empresa",
  telefono: "900900900",
  nif: "XXXXXXXX"
};
```

Y en este código, vemos como añadimos también al prototype de Factura, el método calculaTotal()

```
// Métodos añadidos al prototype de la Factura
Factura.prototype.calculaTotal = function () {
  for (var i = 0; i < this.elementos.length; i++) {
    this.informacion.baseImponible += this.elementos[i].cantidad *
this.elementos[i].precio;
  }
  this.informacion.total = this.informacion.baseImponible *
this.informacion.iva;
}
```

Este método calculará la base imponible, realizando el producto de la cantidad de elementos por el precio de los mismos. Posteriormente realiza el producto de la base imponible por el iva, almacenándolo en la propiedad total de la propiedad informacion de la clase Elementos.

```
Factura.prototype.muestraTotal = function () {
  this.calculaTotal();
  alert("TOTAL = " + this.informacion.total + " euros");
}
```

Mediante la definición del método `muestraTotal()`, añadida al prototype de `Factura` mostrará por pantalla mediante una alerta, el total calculado.

Por último, realizamos la creación del objeto con los parámetros necesarios:

```
// Creación de una factura
// creamos el cliente
var elCliente = new Cliente("Cliente 1", "", "", "");
// creamos los elementos
var losElementos = [new Elemento("elemento1", "1", "5"),
                    new Elemento("elemento2", "2", "12"),
                    new Elemento("elemento3", "3", "42")
];
// creamos el objeto laFactura pasandole como paramentros las
pseudoclases
var laFactura = new Factura(elCliente, losElementos);
// llamamos al metodo muestraTotal()
laFactura.muestraTotal();
```




---

Puedes descargar el código completo desde la plataforma,  
el archivo `JavaScript_Unidad4_EjemploPractico1.doc`

---

Aparte de lo visto hasta ahora, la propiedad `prototype` también nos permite modificar los objetos predefinidos en JavaScript, con lo que redefiniremos el comportamiento de algunos métodos de dichos objetos o añadiremos nuevas propiedades o métodos totalmente nuevos.

Veámoslo en un ejemplo con la clase `Array`. Esta clase no tiene un método que nos indique la posición de un elemento determinado, sería la función `indexOf()` en Java. Veamos como hacerlo:

```
Array.prototype.indexOf = function (objeto) {
    var resultado = -1;
    for (var i = 0; i < this.length; i++) {
        if (this[i] == objeto) {
            resultado = i;
            break;
        }
    }
    return resultado;
}
```

Con el código anterior hemos dado una nueva funcionalidad. Hemos creado un método llamado `indexOf()`, por la que todos los arrays disponen ahora de dicho método, que nos va a devolver el índice de la posición de un elemento determinado que le pasaremos como parámetro al método dentro del array o -1 si el elemento no se encuentra en el array.

Lo que hace dicho método es recorrer el array actual (con `this.length` obtenemos su número de elementos) y comparamos cada elemento con el elemento que se quiere encontrar. Si lo encontramos, se devuelve su posición dentro del array. Si no existe, se devuelve -1.

A continuación, realiza el ejercicio voluntario 2 de la unidad, que corresponde con el archivo de la plataforma **JavaScript\_Unidad4\_EV2.doc**

Con el ejemplo anterior hemos visto que podemos modificar cualquier clase nativa de JavaScript mediante la propiedad `prototype`, incluso la clase `Object`. Veamos otro ejemplo, en el que vamos a modificar la clase `String` para añadir un método que convierta una cadena en un array:

```
<script type="text/javascript">
  String.prototype.toArray = function () {
    return this.split('');
  }
</script>
```

Si recordáis, la función `split()` divide una cadena de texto según el separador indicado. Si no le indicamos ningún separador, la dividirá carácter a carácter. De este modo, la función anterior nos va a devolver un array en el que cada elemento será cada una de las letras de la cadena del texto original.

Con la función `trim()` ocurre lo mismo que con la anterior. Es muy común en otros lenguajes de programación y, en cambio, JavaScript no dispone de ella. Esta función elimina el espacio en blanco que pueda existir al principio y al final de una cadena de texto. Implementémosla mediante la propiedad `prototype` en el objeto `String`:

```
<script type="text/javascript">
  String.prototype.trim = function () {
    return this.replace(/^\s*|\s*$/g, '');
  }

  var cadena = "  prueba  de  cadena  ";
  cadena.trim();
  // Ahora cadena = "prueba  de  cadena"
</script>
```

Usando las expresiones regulares con las que detectamos todos los espacios en blanco que existen tanto al principio como al final de la cadena, unidos a la función `trim()` que hemos añadido al prototipo de la clase `String`, podemos eliminar los espacios sustituyéndolos por una cadena vacía.

De la misma manera, si creamos dos funciones que eliminen los espacios de la derecha e izquierda, `rtrim()` y `ltrim()`, combinándolas realizaremos la misma función que en el código anterior.

```
<script type="text/javascript">
  //funcion que elimina los espacios de la derecha
```

```
String.prototype.rtrim = function () {
    return this.replace(/\s*$/g, '');
}
//funcion que elimina los espacios de la izquierda
String.prototype.ltrim = function () {
    return this.replace(/^\s*/g, '');
}
//funcion que combinando las anteriores elimina ambos lados
String.prototype.trim = function () {
    return this.ltrim().rtrim();
}

var cadena = "   prueba   de   cadena   ";
cadena.trim();
// Ahora cadena = "prueba   de   cadena"
</script>
```

Veamos otra función, con la que eliminaremos todas las etiquetas HTML que pueda contener:

```
<script type="text/javascript">
    String.prototype.stripTags = function () {
        return this.replace(/<\|/?[>]+>/gi, '');
    }

    var cadena = '<html><head><meta content="text/html; charset=UTF-8"
http-equiv="content-type">';
    cadena += '</head><body><p>Parrafo de prueba</p></body></html>';
    //vemos la cadena de texto antes del metodo
    alert(cadena);
    //presentamos la cadena despues del metodo
    alert(cadena.stripTags());
    // Ahora cadena = "Parrafo de prueba"
</script>
```

En este caso también hemos usado las expresiones regulares complejas para eliminar cualquier etiqueta HTML, por lo que se buscan patrones como <...> y </...>.

### 4.3.3. Herencia y ámbito (scope)

Si conoces algún lenguaje de programación orientado a objetos, conocerás los términos de:

- Herencia entre clases.
- El concepto de ámbito (scope) de sus métodos y propiedades, si son public, private, protected...

En JavaScript no tenemos de forma nativa estos conceptos. Si necesitamos hacer uso de estos, debemos simular el funcionamiento mediante clases, funciones y métodos desarrollados a medida.

Se puede realizar una simulación para usar de las propiedades privadas, renombrando su nombre con un guión bajo, distinguiéndola de esta manera de sus propiedades publicas.

También podemos simular la herencia, usando Prototype, y añadiendo un método a la clase Object, llamado extend(), con el que podemos copiar las propiedades de una clase origen en otra que sea la clase destino:

```
<script type="text/javascript">
  Object.extend = function (ClaseDestino, ClaseOrigen) {
    for (var nombrePropiedad in ClaseOrigen) {
      ClaseDestino[nombrePropiedad] = ClaseOrigen[nombrePropiedad];
    }
    return ClaseDestino;
  }
</script>
```

## 4.4. Otros conceptos avanzados

En este apartado vamos a ver otros conceptos más avanzados de JavaScript, que nos serán necesarios conocer para realizar los siguientes puntos de esta unidad.

### 4.4.1. Excepciones

Para realizar el tratamiento de excepciones en JavaScript tenemos un mecanismo muy parecido al de otros lenguajes de programación. Para hacerlo, se definen las palabras reservadas **try**, **catch** y **finally**.

Con **try** encerraremos el bloque de código al cual queremos controlar las excepciones. Lo normal es que al bloque **try** le siga el otro bloque de código definido por **catch**.

Si en el bloque de código encerrado anteriormente se produce una excepción, se ejecutarán las instrucciones que están contenidas dentro del bloque **catch** asociado. Después de este bloque, se puede definir, que no es obligatorio, un bloque con la palabra reservada **finally**.

Debemos saber que el código contenido en el bloque **finally** se ejecutará independientemente de la excepción ocurrida en el bloque **try**.

Además el bloque **try** debe ir seguido obligatoriamente de:

- Bloque **catch**.
- Bloque **finally**.

También puede ir seguido de los dos bloques.

Veamos un ejemplo de excepción y uso de los bloques **try** y **catch**:

```
<script type="text/javascript">
  try {
    var resultado = 5 / a;
  } catch (excepcion) {
    alert(excepcion);
  }
</script>
```

En el bloque **catch** indicaremos el nombre del parámetro que se creará automáticamente si se produce una excepción. Como podéis observar, este sólo está definido dentro del bloque **catch** y lo usaremos para obtener más información sobre la excepción producida.

En este caso, vemos que al dividir el número 5 por la variable “a”, la cual no está definida, se va a producir una excepción que mostrará el siguiente mensaje dentro del bloque catch:

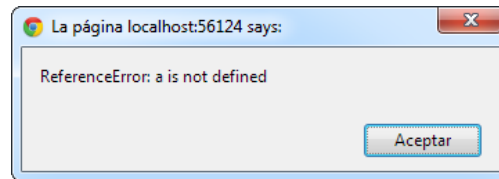


Figura 4.3. Excepción provocada por una variable no definida.

El bloque **finally** no es tan sencillo como el bloque **catch**. Si se ejecuta cualquier parte de código dentro de un bloque **try**, siempre se ejecutará el bloque **finally**, sin importar el resultado de la ejecución del bloque **try**. Si están definidos los bloques **catch** y **finally**, en primer lugar se ejecutará el bloque **catch** y a continuación el bloque **finally**.

Con JavaScript también podemos lanzar excepciones manualmente usando la palabra reservada **throw**:

```
<script type="text/javascript">
  try {
    if (typeof a == "undefined" || isNaN(a)) {
      throw new Error('La variable "a" no es un número');
    }
    var resultado = 5 / a;
  } catch (excepcion) {
    alert(excepcion);
  } finally {
    alert("Se ejecuta");
  }
</script>
```

Vemos que en este caso, se muestran los dos siguientes mensajes de forma consecutiva:

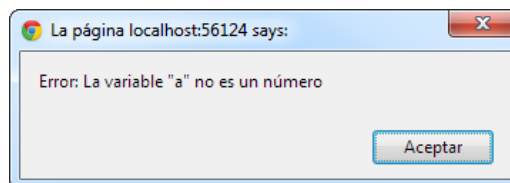


Figura 4.4. Mensaje forzado manual.

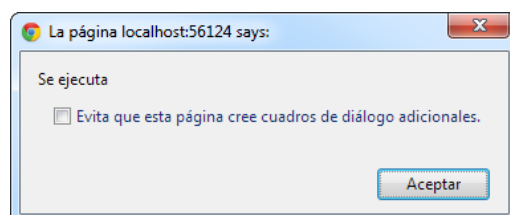


Figura 4.5. Excepción provocada por la ejecución del código.



## 4.4.2. Reflexión

Con JavaScript se pueden definir mecanismos que permitirán simular la reflexión sobre los objetos.



---

La **reflexión** es el proceso en tiempo de ejecución, mediante el cual un programa obtiene información sobre sí mismo y por tanto, es capaz de automodificarse en tiempo de ejecución.

---

Este concepto lo usaremos para descubrir propiedades y métodos de objetos externos. Veamos un sencillo ejemplo, en el que vamos a averiguar, si existe una determinada propiedad en un objeto.

```
if (e1Objeto.laPropiedad) {  
    // el objeto posee la propiedad buscada  
}
```

Si no existe esa propiedad en el objeto, la respuesta será undefined, que será equivalente a un valor false, con lo que no se ejecutará el bloque if.

Pero tenemos un problema con el código anterior, ya que si la propiedad buscada tuviese un valor de false, null o el número 0, el código anterior no se ejecutará correctamente. Para que sea del todo correcto, deberemos usar el siguiente:

```
if (typeof (e1Objeto.laPropiedad) != 'undefined') {  
    // el objeto posee la propiedad buscada  
}
```

Usando el operador `typeof`, nos devolverá el tipo del objeto o variable que se le pasa como parámetro. Este operador devuelve los siguientes valores:

- Undefined.
- Number.
- Object.
- Boolean.
- String.
- Function.

También podemos usar el operador `instanceof`, con el que comprobaremos si un objeto es una instancia de otro objeto:

```
if (elObjeto instanceof Factura) {
    alert("Se trata de un objeto de tipo Factura");
}
```

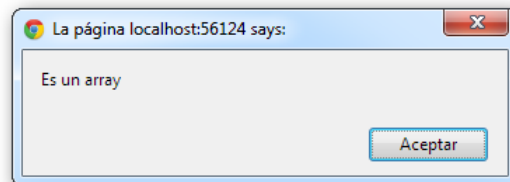


Este operador también se puede emplear con objetos nativos de JavaScript.

Veamos como implementarlo:

```
<script type="text/javascript">
    var elObjeto = [];
    if (elObjeto instanceof Array) {
        alert("Es un array");
    }
    else if (elObjeto instanceof Object) {
        alert("Es un objeto");
    }
</script>
```

Si ejecutamos el código anterior, se mostrará por pantalla:

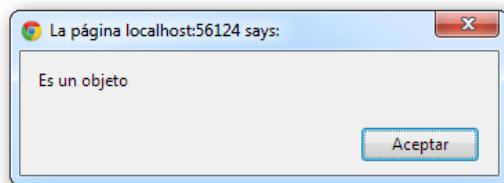


**Figura 4.6. Salida de la ejecución del código.**

Y esto ocurre, ya que se cumple la primera comprobación por ser la variable `elObjeto` un array. Pero si cambiamos el orden de las comprobaciones:

```
<script type="text/javascript">
    var elObjeto = [];
    if (elObjeto instanceof Object) {
        alert("Es un objeto");
    }
    else if (elObjeto instanceof Array) {
        alert("Es un array");
    }
</script>
```

Si ejecutamos este ejemplo, la salida por pantalla será la siguiente:



**Figura 4.7.** Salida de la ejecución del código.

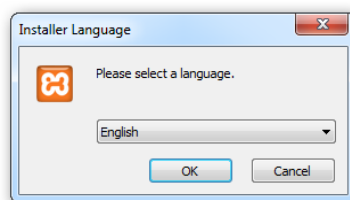
El motivo es el siguiente: recordad que JavaScript no soporta el concepto de herencia en los objetos definidos a medida. En cambio, los objetos nativos `Function` y `Array` sí que heredan del objeto `Object`. Por lo tanto, `(elobjeto instanceof Object)` devuelve `true`, ya que `Array` es una clase que hereda de `Object`.

## 4.5. Instalando Servidor

Para poder realizar las pruebas necesarias, necesitamos tener instalado un servidor en nuestro equipo. Para ello vamos a usar uno gratuito, XAMPP, que lo podéis descargar de la siguiente dirección.

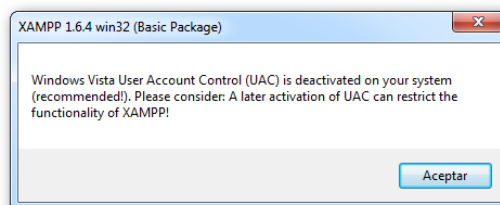
- <http://www.apachefriends.org/en/xampp-windows.html>

Para comenzar la instalación, realizamos doble click en el archivo, descargamos y procedemos con la instalación. Lo primero seleccionamos el idioma en el que se instalará dicho servidor:



**Figura 4.8. Ventana de instalación. Selección del idioma.**

Si en vuestro equipo tenéis desactivado el Control de Cuentas de Usuario, os podrá salir la siguiente advertencia:



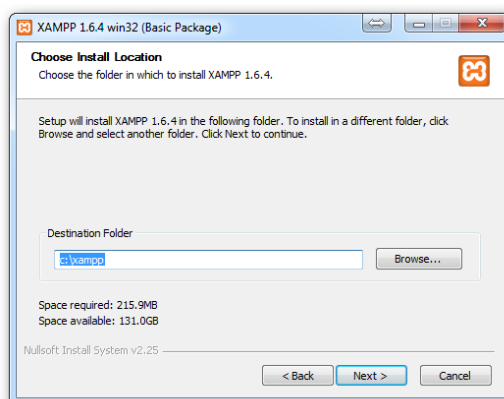
**Figura 4.9. Ventana de instalacion: ventana de advertencia.**

Continuamos con la instalación, y pulsamos en "Next", en la siguiente ventana que nos aparezca:



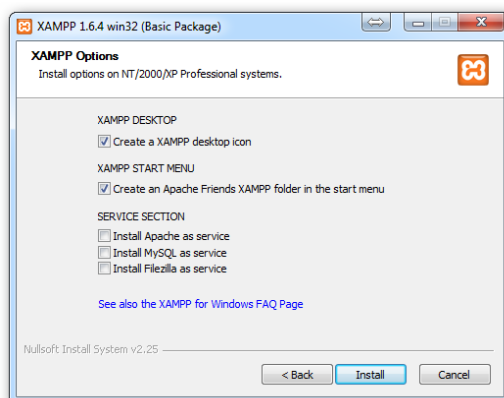
**Figura 4.10. Ventana de instalación: seguimos con la instalación.**

Dejamos que lo instale en el directorio por defecto que nos da.



**Figura 4.11. Ventana de Instalación: dejamos seleccionado el directorio por defecto.**

Y en la siguiente ventana dejamos seleccionado las siguientes opciones:



**Figura 4.12. Ventana de Instalación: dejamos seleccionadas las siguientes opciones.**

Dejamos que finalice la instalación, y cuando acabe nos pedirá permiso para iniciar el "Panel de Control" del servidor.

## 4.6. Primera aplicación AJAX

En este punto vamos a desarrollar una aplicación sencilla de ejemplo, que nos servirá para entender mejor el funcionamiento de este tipo de programas, y ver el papel que desempeñan las distintas tecnologías mencionadas en el punto 1 de esta unidad.

Esta aplicación va a consistir en una adaptación del clásico "Hola Mundo". Lo que haremos en nuestro ejemplo será realizar la petición al servidor, para que se descargue un archivo y muestre el contenido del mismo, sin que sea necesario recargar de nuevo la página.

Para ello necesitamos crear la siguiente página XHTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title></title>
  <script type="text/javascript">
    function descargaArchivo() {
      // Obtener la instancia del objeto XMLHttpRequest
      if (window.XMLHttpRequest) {
        petition_http = new XMLHttpRequest();
      }
      else if (window.ActiveXObject) {
        petition_http = new ActiveXObject("Microsoft.XMLHTTP");
      }

      // Preparar la funcion de respuesta
      petition_http.onreadystatechange = muestraContenido;

      // Realizar peticion HTTP
      petition_http.open('GET', 'http://localhost:56124/HolaMundo.txt',
true);
      petition_http.send(null);

      function muestraContenido() {
        if (petition_http.readyState == 4) {
          if (petition_http.status == 200) {
            alert(petition_http.responseText);
          }
        }
      }
    }

    window.onload = descargaArchivo;
  </script>
</head>
<body>
</body>
</html>
```

En este ejemplo, cuando cargamos la página ejecutamos el método JavaScript que va a mostrar el contenido de un archivo llamado `HolaMundo.txt` alojado en el servidor. La clave del código anterior es que la petición HTTP y la descarga de los contenidos del archivo se realizan sin necesidad de recargar la página. Veamos parte a parte el código anterior.

En la anterior aplicación tenemos cuatro grandes bloques:

- Parte en la que instanciamos el objeto `XMLHttpRequest`.
- Parte en la que preparamos la función de respuesta.
- Bloque donde realizamos la petición al servidor.
- Bloque donde ejecutamos la función de respuesta.

Siempre, en primer lugar, debemos instanciar el objeto `XMLHttpRequest`, que será el objeto clave para realizar las comunicaciones con el servidor en segundo plano, sin que sea necesario recargar las páginas.

Tenemos que tener claro, que la implementación del objeto `XMLHttpRequest` depende de cada navegador, por lo que para evitar problemas de compatibilidad de los navegadores, emplearemos una discriminación sencilla en función del navegador en el que se está ejecutando el código:

```
if (window.XMLHttpRequest) {  
    petition_http = new XMLHttpRequest();  
}  
else if (window.ActiveXObject) {  
    petition_http = new ActiveXObject("Microsoft.XMLHTTP");  
}
```

El objeto `XMLHttpRequest` lo implementan de forma nativa:

- Firefox.
- Safari.
- Opera.
- Internet Explorer 7 y 8.

Los navegadores obsoletos como Internet Explorer 6 y anteriores implementan el objeto `XMLHttpRequest` como un objeto de tipo `ActiveX`.

Después preparamos la función que va a procesar la respuesta del servidor. Con la propiedad `onreadystatechange` del objeto anteriormente instanciado, indicaremos esta función directamente incluyendo su código mediante una función anónima o indicando una referencia a una función independiente. Nosotros lo vamos a realizar indicando directamente el nombre de la función:

```
petition_http.onreadystatechange = muestraContenido;
```

Con el código anterior hemos indicado a nuestra aplicación, que si se recibe la respuesta del servidor, se ejecutará la función `muestraContenido()`.



La referencia a la función se debe indicar mediante su nombre sin paréntesis, ya que si pusiésemos dicho paréntesis, estaríamos ejecutando la función y almacenando el valor devuelto en la propiedad `onreadystatechange`.

Después realizamos la petición HTTP al servidor:

```
peticion_http.open('GET', 'http://localhost/HolaMundo.txt', true);
peticion_http.send(null);
```

Con este código hemos realizado la petición al servidor mediante una petición de tipo GET simple, con la que no enviamos ningún parámetro al servidor. Dicha petición se crea mediante el método `open()`, en el que debemos incluir lo siguiente:

- El tipo de petición (GET).
- La URL solicitada.
- Un tercer parámetro que vale `true`.

Una vez creada la petición se envía al servidor mediante el método `send()`. Hemos incluido un parámetro que le daremos un valor `null`. Más adelante veremos en detalle los diferentes métodos y propiedades que permiten hacer las peticiones al servidor.

Y ya por último, una vez recibida la respuesta del servidor, se ejecutará de forma automática la función `muestraContenido()`:

```
function muestraContenido() {
    if (peticion_http.readyState == 4) {
        if (peticion_http.status == 200) {
            alert(peticion_http.responseText);
        }
    }
}
```

Esta función, primeramente comprobará que se ha recibido la respuesta del servidor, dependiendo del valor de la propiedad `readyState`. Si es así, comprobamos que sea válida y correcta, comprobando si el status o código de estado de HTTP, es igual a 200.

Una vez realizadas las comprobaciones, mostraremos por pantalla el contenido de la respuesta del servidor (en este caso, el contenido del archivo solicitado) mediante la propiedad `responseText`.



Una vez que ya se ha visto un primer contacto con una aplicación AJAX, vamos a mejorar este código desarrollado hasta ahora. Lo primero es definir unas variables que usaremos en la función que procesa la respuesta del servidor:

```
var READY_STATE_UNINITIALIZED = 0;
var READY_STATE_LOADING = 1;
var READY_STATE_LOADED = 2;
var READY_STATE_INTERACTIVE = 3;
var READY_STATE_COMPLETE = 4;
```

La respuesta del servidor sólo puede ser uno de los cinco estados definidos por las variables anteriores. Así, podemos usar el nombre del estado en vez de su valor numérico, cosa que nos facilitará la lectura y el mantenimiento de las aplicaciones.

Otra de las mejoras que podemos implementar va a ser el almacenar la instancia del objeto XMLHttpRequest en una variable global, de tal manera que todas las funciones que hacen uso de ese objeto va a tener acceso directo al mismo:

```
var petition_http;
```

Y, además, vamos a crear una función genérica de carga de contenidos mediante AJAX:

```
function cargaContenido(url, metodo, funcion) {
    petition_http = inicializa_xhr();

    if (petition_http) {
        petition_http.onreadystatechange = funcion;
        petition_http.open(metodo, url, true);
        petition_http.send(null);
    }
}
```

Esta función va a admitir tres parámetros:

- La URL del contenido que se va a cargar
- El método utilizado para realizar la petición HTTP
- Una referencia a la función que procesa la respuesta del servidor.

Esta función va a inicializar el objeto XMLHttpRequest, con la función inicializa\_xhr(), después, establecerá la función que procesa la respuesta del servidor, utilizando el objeto petition\_http. Por último, se realizará la petición al servidor, usando la URL y el método HTTP, pasados como parámetros a la función.

Anteriormente hemos hablado de la función inicializa\_xhr(). Esta función se va a emplear para encapsular la creación del objeto XMLHttpRequest:

```
function inicializa_xhr() {
    if (window.XMLHttpRequest) {
        return new XMLHttpRequest();
    }
}
```

```

    }
    else if (window.ActiveXObject) {
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}

```

La función `muestraContenido()`, que ya teníamos definida, la vamos a refactorizar para emplear las variables globales definidas:

```

function muestraContenido() {
    if (peticion_http.readyState == READY_STATE_COMPLETE) {
        if (peticion_http.status == 200) {
            alert(peticion_http.responseText);
        }
    }
}

```

Y ya por último, creamos la función `descargaArchivo()` que va a ser la que realice una llamada a la función `cargaContenido()` con los parámetros adecuados:

```

function descargaArchivo() {
    cargaContenido("http://localhost/holamundo.txt", "GET",
muestraContenido);
}

```

A continuación se muestra el código completo de la refactorización de la primera aplicación, y lo puedes descargar desde la plataforma, descargando el archivo `EjemploAjax_2.htm`:

```

<script type="text/javascript">
    var READY_STATE_UNINITIALIZED = 0;
    var READY_STATE_LOADING = 1;
    var READY_STATE_LOADED = 2;
    var READY_STATE_INTERACTIVE = 3;
    var READY_STATE_COMPLETE = 4;

    var peticion_http;

    function cargaContenido(url, metodo, funcion) {
        peticion_http = inicializa_xhr();

        if (peticion_http) {
            peticion_http.onreadystatechange = funcion;
            peticion_http.open(metodo, url, true);
            peticion_http.send(null);
        }
    }

    function inicializa_xhr() {
        if (window.XMLHttpRequest) {
            return new XMLHttpRequest();
        }
        else if (window.ActiveXObject) {

```

```
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function muestraContenido() {
    if (peticion_http.readyState == READY_STATE_COMPLETE) {
        if (peticion_http.status == 200) {
            alert(peticion_http.responseText);
        }
    }
}

function descargaArchivo() {
    cargaContenido("http://localhost/holamundo.txt", "GET",
muestraContenido);
}

window.onload = descargaArchivo;
</script>
```

## 4.7. Otros métodos y propiedades del objeto XMLHttpRequest

En este punto vamos a ver otros métodos y propiedades que tiene el objeto XMLHttpRequest, con la definición de cada uno de ellos:

Las propiedades definidas para el objeto XMLHttpRequest son:

Propiedad	Descripción
readyState	Valor numérico (entero) que almacena el estado de la petición.
responseText	Contenido de la respuesta del servidor en forma de cadena de texto.
responseXML	Contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM.
status	Código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para "No encontrado", 500 para un error de servidor, etc.).

**Figura 4.13. Propiedades del Objeto XMLHttpRequest.**

Los valores definidos para la propiedad readyState son los siguientes:

Valor	Descripción
0	No inicializado (objeto creado, pero no se ha invocado el método open).
1	Cargando (objeto creado, pero no se ha invocado el método send).
2	Cargado (se ha invocado el método send, pero el servidor aún no ha respondido).
3	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad responseText).

**Figura 4.14. Valores definidos de la propiedad readyState.**

Veamos también los métodos disponibles para el objeto XMLHttpRequest:

Valor	Descripción
abort()	Detiene la petición actual.
getAllResponseHeaders()	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor.
getResponseHeader("cabecera")	Devuelve una cadena de texto con el contenido de la cabecera solicitada.
onreadystatechange	Responsable de manejar los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP. Normalmente es una referencia a una función JavaScript.
open("metodo", "url")	Establece los parámetros de la petición que se realiza al servidor. Los parámetros necesarios son el método HTTP empleado y la URL destino (puede indicarse de forma absoluta o relativa).
send(contenido)	Realiza la petición HTTP al servidor.
setRequestHeader("cabecera", "valor")	Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método open() antes que setRequestHeader().

**Figura 4.15. Métodos disponibles del objeto XMLHttpRequest.**

En la tabla anterior vemos que el método open() requiere dos parámetros (método HTTP y URL) pero que también puede aceptar de forma opcional otros tres parámetros. La definición formal del método open() sería de la siguiente manera:

```
open(string metodo, string URL [,boolean asincrono, string usuario, string password]);
```

Recordad que por defecto, las peticiones realizadas son asíncronas. Por lo tanto, si indicamos un valor false al tercer parámetro, la petición la vamos a realizar de forma síncrona, y por lo tanto, detendremos la ejecución de la aplicación hasta que se reciba completamente la respuesta del servidor.



Recuerde

La filosofía de AJAX son contrarias a las peticiones **síncronas**. Una petición síncrona congelará el navegador y no permitirá al usuario realizar ninguna acción hasta que no se haya recibido la respuesta completa del servidor.

Tendremos una sensación de que el navegador se ha colgado, por lo que recomendamos el uso de peticiones asíncronas siempre que se pueda.

Los parámetros `string usuario` y `string password`, son opcionales y nos van a permitir indicar un nombre de usuario y una contraseña válidos para acceder al recurso solicitado, únicamente si son necesarios usarlos.

Por otra parte, también debemos comentar que el método `send()` va a requerir de un parámetro que indicará que tipo de información se va a enviar al servidor junto con la petición HTTP. Los valores que tendrá dicho parámetro serán:

- Si no se envían datos, el valor debe ser igual a `null`.
- En caso contrario, se indicará como parámetro una cadena de texto, un array de bytes o un objeto XML DOM.

## 4.8. Introducción a jQuery

jQuery es una forma de acceder a los objetos del DOM de una manera simplificada. Debéis saber que realmente es una librería de JavaScript que deberéis incluir en vuestro proyecto para poderla utilizar.

Para descargar la última versión disponible podéis acceder a su página oficial: <http://jquery.com/>.

El uso de esta librería nos va a facilitar incorporar efectos visuales tales como:

- Drag and drop.
- Auto-completar.
- Animaciones complejas.

Realizar todos estos conceptos desde cero sería muy complicado por lo que recomendamos el uso de dichas librerías para agilizar nuestro desarrollo. Además, contamos con la ventaja de despreocuparnos con la compatibilidad de navegadores, ya que la librería resolverá esto.

Una vez que ya tenemos descargada dicha librería, debemos incluir su uso en cada página HTML que creamos, agregando en su código la siguiente línea:

```
<script type="text/javascript" src="jquery-1.8.2.js" ></script>
```

Aconsejamos crear una carpeta y copiar el archivo dentro, para después ir referenciando a dicha localización en cada ejemplo que se realice.

Para descargarnos dicha librería, accedemos a la dirección:

<http://jquery.com/download/> y pulsamos sobre el botón de



Se nos abrirá una nueva página, con el código de esta librería. Para guardar dicho código, podemos realizarlo de dos maneras:

1. Seleccionamos todo el código y en un nuevo documento de texto lo pegamos y guardamos con el nombre de la versión que nos hemos descargado. En nuestro caso, el día de la confección de este temario iban por la versión v1.8.2. por lo tanto, el nombre del archivo será **jquery-1.8.2.min.js** y será el archivo que deberemos usar para la realización de nuestros ejemplos.
2. Sobre la nueva página con el código que se nos ha abierto, nos situamos encima y con el ratón y su botón derecho, seleccionamos la opción de "guardar como". Seleccionamos la ubicación donde la queremos guardar y pulsamos en aceptar.

Debemos tener en cuenta, que del sitio oficial de jQuery podemos descargar dos versiones de la librería, una es la versión descomprimida, que ocupa alrededor de 60 Kb (es el archivo jquery.js) y la otra, la comprimida, que será la que debemos subir al servidor cuando subamos nuestro sitio.

En resumen la librería jQuery nos aporta las siguientes ventajas:

- Nos va a permitir ahorrar muchas líneas de código.
- Va a dar soporte de nuestra aplicación para los principales navegadores.
- Nos ofrece mecanismos para la captura de eventos.
- De una forma sencilla podremos animar el contenido de la página.
- Integra funcionalidades para trabajar con AJAX.

Recomendamos antes de seguir con esta unidad, si no se tienen conocimientos de HTML y CSS, se revisen los anexos a dichos temarios, ya que serán necesarios conocimientos previos de ambas tecnologías para desarrollar el resto de este temario.



## 4.9. Programar JavaScript con jQuery

Como esta librería es una nueva implementación, debemos ir acostumbrándonos poco a poco a trabajar con ella para comprender correctamente su funcionamiento. Empezaremos con sencillos ejemplos, que se irán complicando según vayamos avanzando con la unidad.

En este primer ejemplo, vamos a ver como capturar un evento de un control de HTML de tipo botón. Veamos como lo hacíamos hasta ahora, accediendo a las funciones del DOM, para posteriormente hacerlo usando jQuery, y así comprenderéis las diferencias.

Todos los ejemplos que vayamos desarrollando debemos colocarlos en carpetas, para que los archivos JavaScript que se vayan creando, no afecten al resto de páginas. Para este primer ejemplo necesitaremos de 3 páginas, 2 páginas html que colocaremos en la raíz de la carpeta que se cree:

página1.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>pagina1</title>
</head>
<body>
  <h2>
    Captura del evento click de un control HTML de tipo button.</h2>
    <a href="pagina2.html">Método tradicional con las funciones del DOM </a>
    <br>
    <a href="pagina3.html">Utilizando la librería jQuery</a><br>
  </body>
</html>
```

página2.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>pagina2</title>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <input type="button" id="boton1" value="presioname">
</body>
</html>
```

Y una página donde vamos a realizar el código JavaScript. Debemos colocarlo en una subcarpeta llamada /Scripts. Si lo ponemos en la misma carpeta que el resto de archivos, deberíamos cambiar la propiedad `src` de las anteriores páginas, ya que no nos funcionaría el ejemplo. Crearemos un archivo llamado `Funciones1.js` y colocamos el siguiente código:

```
addEvent(window, 'load', inicializarEventos, false);

function inicializarEventos() {
    var boton1 = document.getElementById('boton1');
    addEvent(boton1, 'click', presionBoton, false);
}

function presionBoton(e) {
    alert('se presionó el botón');
}

function addEvent(elemento, nomevento, funcion, captura) {
    if (elemento.attachEvent) {
        elemento.attachEvent('on' + nomevento, funcion);
        return true;
    }
    else
        if (elemento.addEventListener) {
            elemento.addEventListener(nomevento, funcion, captura);
            return true;
        }
        else
            return false;
}
```

En este código, no debería haber problemas de entendimiento, si es así, deberías repasar las unidades anteriores.

Si ejecutamos dicha página, veremos su funcionamiento, que es como hasta ahora se había realizado durante todo el temario, accediendo al DOM y a sus elementos.

Ahora veamos como realizar la misma operación, pero usando la librería de jQuery. Para ello necesitamos crear un nuevo archivo HTML, que llamaremos `página3.html`, y que colocaremos en la carpeta raíz. Su código será el siguiente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>pagina3</title>
    <script src="../../Scripts/jquery-1.8.2.min.js"
type="text/javascript"></script>

    <script src="Scripts/funciones2.js" type="text/javascript"></script>
</head>
<body>
    <input type="button" id="boton1" value="presioname">
</body>
</html>
```

Debemos insertar la referencia a la librería de jQuery, en todas las páginas en las que hagamos uso de ella:

```
<script src="/Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
```



Siempre debemos importar la librería, antes de usar los archivos .js creados.

Debemos seguir siempre el orden siguiente:

```
<script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
<script src="Scripts/funciones2.js" type="text/javascript"></script>
```

En el archivo Funciones2.js insertaremos el código siguiente:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x;
    x = $("#boton1");
    x.click(presionBoton)
}

function presionBoton() {
    alert("Se presionó el botón");
}
```

Vamos a ver parte por parte que hace cada función en esta página. Lo más importante:



La función principal de esta librería se llama \$.

A esta función le podemos pasar distintos valores. En el primer caso le pasamos la referencia del objeto document del DOM y en la segunda el id del control button:

```
x = $(document);
...
x = $("#boton1");
```

La devolución de esta función siempre va a ser un objeto de tipo jQuery.

El primer método que nos interesa de esta clase es el ready:

```
var x;  
x = $(document);  
x.ready(inicializarEventos);
```

Como vemos en el ejemplo anterior, al método ready se le pasa como parámetro un nombre de función. Dicha función se ejecutará solamente cuando todos los elementos de la página estén cargados.



---

Fijaros que solo pasamos el nombre de la función y no insertamos los paréntesis abiertos y cerrados al final.

---

La función de inicializarEventos():

```
function inicializarEventos() {  
    var x;  
    x = $("#boton1");  
    x.click(presionBoton)  
}
```

Vemos como de nuevo, utilizamos la función \$ para crear un objeto de la clase jQuery, pero esta vez asociándolo al botón. Para realizarlo le pasamos el id del control button precediéndolo por el carácter # y encerrándolo entre paréntesis.

Por último, llamamos al método click, al que le pasaremos como parámetro el nombre de la función que queremos que se ejecute al presionar dicho botón. Fijaros que en este caso ocurre lo mismo, y no le pasamos los paréntesis de apertura y cierre.

En los próximos capítulos vamos a ver como realizar las operaciones más comunes con esta librería. Recomendamos crear una carpeta e ir separando las páginas en diferentes carpetas, para poder realizar la ejecución de las mismas por separa.

### 4.9.1. Selección de un elemento del DOM por el id

La forma de realizar la selección de un elemento particular de la página mediante la propiedad id es, haciéndolo mediante la siguiente sintaxis:



`$("#nombre del id")`

Realizamos un ejemplo, para ver como obtener la referencia a elementos HTML particulares mediante su id. En este ejemplo2, vamos a realizar una página que nos va a mostrar dos títulos de primer nivel, y que al ser presionados, cambiarán el color de la fuente, del fondo y el tipo de fuente del texto.

Para ello partiremos de la siguiente página1.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo2</title>
  <script src="../../Scripts/jquery-1.8.2.min.js"
type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <h1 id="titulo1">
    Primer título</h1>
  <h1 id="titulo2">
    Segundo título</h1>
</body>
</html>
```

Vemos que tenemos incluidas las referencias tanto a la librería de jQuery como al archivo donde vamos a declarar las funciones de jQuery. Este archivo funciones1.js contendrá lo siguiente:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("#titulo1");
  x.click(presionTitulo1)
  x = $("#titulo2");
  x.click(presionTitulo2)
}
```

```
function presionTitulo1() {
    var x;
    x = $("#titulo1");
    x.css("color", "#ff0000");
    x.css("background-color", "#ffff00");
    x.css("font-family", "Courier");
}

function presionTitulo2() {
    var x;
    x = $("#titulo2");
    x.css("color", "#ffff00");
    x.css("background-color", "#ff0000");
    x.css("font-family", "Arial");
}
```

Siempre que trabajemos con esta librería:

- Primero crearemos un objeto jQuery a partir de la referencia a 'document'.
- Segundo llamaremos al método ready con la que indicaremos el nombre de la función que se ejecutará una vez generado el árbol de elementos HTML para la página:

```
var x;
x = $(document);
x.ready(inicializarEventos);
```

Después tenemos la función inicializarEventos(). Esta función se ejecutará cuando se haya cargado completamente la página y estén creados todos los elementos HTML. Lo que haremos con esta función será acceder mediante \$ a través del id a los elementos h1 respectivos, asignándoles a cada uno mediante su evento click, una función distinta que se disparará cuando presionemos con el mouse:

```
function inicializarEventos() {
    var x;
    x = $("#titulo1");
    x.click(presionTitulo1)
    x = $("#titulo2");
    x.click(presionTitulo2)
}
```



Recuerde

---

Para obtener la referencia a un elemento por medio de su id, debemos anteponer el caracter # al nombre del id.

---

Y finalmente, declaramos las dos funciones que se ejecutan al presionar los títulos:

```
function presionTitulo1() {  
    var x;  
    x = $("#titulo1");  
    x.css("color", "#ff0000");  
    x.css("background-color", "#ffff00");  
    x.css("font-family", "Courier");  
}  
  
function presionTitulo2() {  
    var x;  
    x = $("#titulo2");  
    x.css("color", "#ffff00");  
    x.css("background-color", "#ff0000");  
    x.css("font-family", "Arial");  
}
```

Hasta este momento habíamos visto dos métodos del objeto jQuery:

- Ready.
- Click.

En este ejemplo, vemos un tercer método, que nos va a permitir modificar diferentes propiedades de la hoja de estilo de un elemento HTML.

Una vez que ya tenemos la referencia a un elemento HTML vamos a llamar al método **css** al que le pasaremos dos parámetros:

- El primero indica el nombre de la propiedad.
- El segundo el valor a asignarle.

Podemos ver las tres asignaciones:

```
x.css("color", "#ff0000");  
x.css("background-color", "#ffff00");  
x.css("font-family", "Courier");
```

Ya veis lo sencillo que es acceder al estilo de un elemento HTML y poder actualizarlo de una manera dinámica después que la página haya sido cargada.

## 4.9.2. Selección de elementos por el tipo de elementos

Hasta ahora habíamos visto como seleccionar un elemento mediante su id de elemento, para cambiar ciertas propiedades de una forma dinámica. En este punto veremos como seleccionar varios elementos dependiendo de su tipo. La sintaxis para tener la referencia de todos los elementos de cierto tipo (a, p, h1, etc.):



`$("nombre del elemento")`

Si os fijáis, se obtiene de forma muy similar a la anterior, pero sin insertar el carácter #. De esta manera, podremos definir funciones comunes a un conjunto de elementos.

Veámoslo con un ejemplo. Será el ejemplo3 en el cual vamos a crear una tabla con 5 filas, y mediante el uso de la librería de jQuery, haremos que cambie de color la celda que se presiona con el mouse. Para ello, tendremos que obtener la referencia a todos los elementos 'td'. Necesitamos crear una página HTML con una tabla con 5 filas:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>pagina1</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <table style="padding: 1px; border-width: 1px; border-style: solid;">
    <tr>
      <td>
        1111111111
      </td>
      <td>
        1111111111
      </td>
      <td>
        1111111111
      </td>
      <td>
        1111111111
      </td>
    </tr>
    <tr>
      <td>
        2222222222
```



```
        </td>
        <td>
            2222222222
        </td>
        <td>
            2222222222
        </td>
        <td>
            2222222222
        </td>
    </tr>
    <tr>
        <td>
            3333333333
        </td>
        <td>
            3333333333
        </td>
        <td>
            3333333333
        </td>
        <td>
            3333333333
        </td>
    </tr>
    <tr>
        <td>
            4444444444
        </td>
        <td>
            4444444444
        </td>
        <td>
            4444444444
        </td>
        <td>
            4444444444
        </td>
    </tr>
    <tr>
        <td>
            5555555555
        </td>
        <td>
            5555555555
        </td>
        <td>
            5555555555
        </td>
        <td>
            5555555555
        </td>
    </tr>
</table>
</body>
</html>
```

De esta forma, hemos creado nuestra tabla. Ahora vamos a realizar las funciones necesarias para realizar lo descrito anteriormente. Las insertaremos en el archivo `funciones1.js`:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x;
    x = $('td');
    x.click(presionCelda);
}

function presionCelda() {
    var x;
    x = $(this);
    x.css("background-color", "lightgray");
}
```

Vamos a realizar las mismas operaciones siempre, primero creamos el objeto jQuery, y después inicializaremos la función que queremos que se ejecute una vez que se carguen todos los elementos del documento, mediante el método `ready` del objeto.

```
var x;
x = $(document);
x.ready(inicializarEventos);
```

Con la siguiente sintaxis obtenemos la referencia a todos los elementos HTML de tipo 'td':

```
var x;
x = $('td');
x.click(presionCelda);
```

Y a todos ellos los unimos con la función `presionCelda()` en el evento `click`. Cuando presionemos cualquiera de las celdas de la tabla se ejecutará la siguiente función:

```
function presionCelda() {
    var x;
    x = $(this);
    x.css("background-color", "lightgray");
}
```

Vemos que la forma de obtener la referencia al elemento que dispara el evento lo hacemos con la palabra reservada `this`:

```
x = $(this);
```

Y una vez obtenida dicha referencia modificaremos la propiedad css de ese elemento:

```
x.css("background-color", "lightgray");
```

Si vemos en ejecución nuestro ejemplo, cada vez que pulsemos con el ratón sobre una celda, se pondrá de color gris.

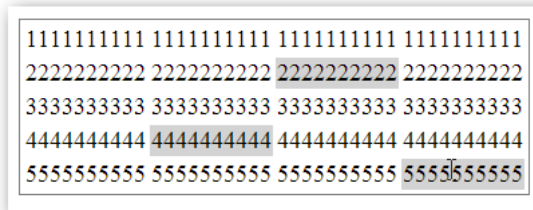


Figura 4.16. Ejecución del ejemplo3.

### 4.9.3. Selectores

En los puntos anteriores hemos visto que el parámetro que se le pasa a la función jQuery es un selector que nos determina los elementos de la página a los que se desea acceder. Como a través del parámetro vamos a controlar gran parte del potencial de jQuery, nos vamos a detener examinando algunas de sus posibilidades.

Dicha sintaxis se apoya en CSS3 y XPath. Tenéis que saber que no todos los navegadores soportan estos estándares, pero aun así, jQuery tiene su propio motor para interpretar los selectores, por lo que independientemente del navegador empleado, su compatibilidad no se ve afectada.

Los primeros de los que vamos a hablar va a ser de los considerados "básicos", que nos permiten seleccionar los elementos en función de:

- Su id.
- Su clase.
- Sus atributos.
- Sus relaciones de dependencia entre ellos.

La tabla siguiente nos presenta unos cuantos ejemplos.

Selector	Descripción
<code>\$('.laClase')</code>	Selecciona todos los elementos que tienen <code>class='laClase'</code>
<code>\$('#elId')</code>	Selecciona todos los elementos que tienen <code>id='elId'</code>
<code>\$('div')</code>	Selecciona todos los elementos de tipo <code>&lt;div&gt;</code>
<code>\$('p a')</code>	Selecciona todos los elementos de tipo <code>&lt;a&gt;</code> que se encuentren dentro de un <code>&lt;p&gt;</code>
<code>\$('img[alt]')</code>	Selecciona todos los elementos de tipo <code>&lt;img&gt;</code> que tengan un atributo <code>alt</code>
<code>\$('p#mild')</code>	Selecciona todos los elementos de tipo <code>&lt;p&gt;</code> que tengan <code>id='mild'</code>
<code>\$('li&gt;div')</code>	Selecciona todos los elementos de tipo <code>&lt;div&gt;</code> que estén contenidos directamente dentro de un <code>&lt;li&gt;</code>
<code>\$('a[href\$.xls]')</code>	Selecciona todos los hiperenlaces ( <code>&lt;a&gt;</code> ) cuyo atributo <code>href</code> tiene un valor que termina en <code>.xls</code>

**Figura 4.17. Selectores básicos.**

La siguiente tabla presenta una serie de ejemplos de selectores que se basan en la posición relativa de los elementos para seleccionarlos.

Selector	Descripción
<code>\$('p:first')</code>	Selecciona el primer <code>&lt;p&gt;</code> de la pagina
<code>\$('div.laClase:last')</code>	Selecciona el ultimo <code>&lt;div&gt;</code> que exista en la pagina que tenga la clase <code>laClase</code>
<code>\$('li:first-child')</code>	Selecciona todos los elementos del tipo <code>&lt;li&gt;</code> que sean el primer hijo de su contenedor (es decir, el primer ítem de cada lista)
<code>\$('li:nth-child(2)')</code>	Selecciona todos los elementos del tipo <code>&lt;li&gt;</code> que sean el hijo número 2 dentro de su contenedor (es decir, el segundo ítem de cada lista)
<code>\$('ul:only-child')</code>	Selecciona todos los elementos del tipo <code>&lt;ul&gt;</code> que sean los únicos elementos dentro de su contenedor
<code>\$('table&gt;tr:odd')</code>	Selecciona todos los elementos <code>&lt;tr&gt;</code> que ocupen una posición impar dentro de una tabla. Se utiliza para crear efectos de "papel pijama"
<code>\$('.laClase:lt(4)')</code>	Selecciona los 4 primeros elementos de la pagina que tengan la clase <code>laClase</code> (es decir, los que ocupan las posiciones 0, 1, 2 y 3)

**Figura 4.18. Selectores que se basan en la posición relativa de los elementos para seleccionarlos.**

Y por último, existen selectores hechos "a medida" para jQuery, que además no están basados en CSS como los anteriores. Al ser muy extensa la lista completa, y siendo que se pueden encontrar en la Web de jQuery, en la dirección <http://api.jquery.com/category/selectors/> no vamos a hablar extensamente de ellos.

Por comentar alguno de ellos:

- `$('input:hidden')` este selector seleccionará todos los campos `<input>` que se encuentran ocultos (hidden)
- `$('img:animated')` con este seleccionaremos todas las imágenes que están siendo animadas desde jQuery,
- `$('p:contains(texto)')` seleccionamos todos los párrafos que contienen el texto indicado.

Veamos un ejemplo donde aplicamos algunos de los selectores vistos anteriormente. En este ejemplo4, vamos a ocultar todos los list ítem de una lista primera lista, de una pagina que contiene dos listas desordenadas.

Partimos de la siguiente pagina XHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>pagina1</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <input type="button" value="ocultar item primer lista" id="boton1">
  <h2>
    Lista 1</h2>
  <ul id="lista1">
    <li>Opción número 1</li>
    <li>Opción número 2</li>
    <li>Opción número 3</li>
    <li>Opción número 4</li>
  </ul>
  <h2>
    Lista 2</h2>
  <ul id="lista2">
    <li>Opción número 1</li>
    <li>Opción número 2</li>
    <li>Opción número 3</li>
    <li>Opción número 4</li>
  </ul>
</body>
</html>
```

Cuando pulsemos sobre el botón, ocultaremos la “lista1”. Para ello necesitamos insertar el siguiente código en el archivo funciones1.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

/*obtenemos la referencia del botón mediante
'su id y llamamos al método click */
```

```
function inicializarEventos() {
    var x;
    x = $("#boton1");
    x.click(ocultarItem);
}

function ocultarItem() {
    var x;
    x = $("#lista1 li");
    x.hide();
}
```

Como en todos los ejemplos, en la función `inicializarEventos()` vamos a obtener la referencia del botón mediante su id para posteriormente llamar al método `click`.

Con el método `ocultarItem()` obtenemos la referencia de todos los elementos `li` que pertenecen a la primera lista:

```
var x;
x = $("#lista1 li");
x.hide();
```

Pasamos a la función `$` el selector CSS que accede a todos los elementos de tipo `li` contenidos en `#lista1` y los ocultamos.



Todas las reglas de CSS están disponibles para seleccionar los elementos del DOM

Si ejecutamos nuestro ejemplo, nos saldrán las dos listas y un botón, que al ser pulsado nos ocultará la primera de ellas:

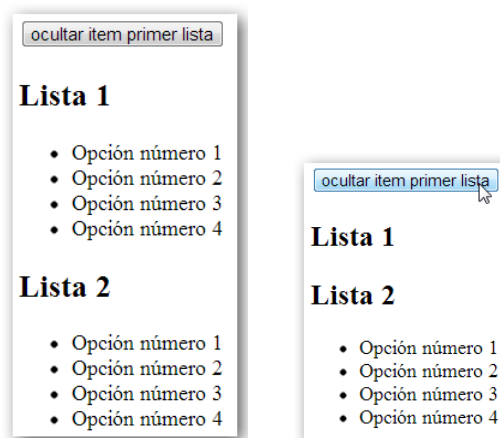


Figura 4.19. Ejecucion del ejemplo 4.

### 4.9.4. Métodos del objeto jQuery

jQuery dispone de métodos interesantes y útiles que son importantes conocer. En los siguientes puntos, vamos a profundizar un poco más en ellos.

#### 4.9.4.1. Métodos `tex()`, `text(valor)`

Este método nos va a permitir conocer el contenido de un elemento del formulario. Por ejemplo, para conocer el contenido de un párrafo con `id="parrafo1"`. Primero obtenemos la referencia al párrafo:

```
var x = $("#parrafo1");
```

Y después, mediante el método `x.text()` obtenemos el contenido del párrafo. Una vez realizado esto, por ejemplo, podemos cambiar el texto del párrafo:

```
var x = $("#parrafo1");
x.text("Este es el texto nuevo");
```

Es muy importante, que cuando utilicéis jQuery, tengáis mucho cuidado, ya que si no definimos correctamente el elemento a cambiar, podríamos cambiar todos los elementos de ese tipo en el formulario:

```
var x = $("p");
x.text("Este texto aparece en todos los párrafos del documento");
```

En el código anterior hemos creado una referencia a todos los párrafos contenidos en el documento. Cuando después realizamos la llamada al método `text()` y le enviamos la cadena, esta afectará a todo el documento, remplazando el contenido de todos los párrafos.

Veámoslo mejor con un ejemplo, en el que realizaremos el acceso y modificación unitaria y múltiple de contenidos de elementos del formulario. Partimos de la `página1.htm`, que contendrá el siguiente código:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Problema</title>
  <script src="../../Scripts/jquery-1.8.2.min.js"
type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>

<body>
  <input type="button" value="Obtener el texto contenido en un párrafo"
id="boton1"><br>
  <input type="button" value="Modificar el texto de un párrafo"
id="boton2"><br>
```

```

    <input type="button" value="Modificar el texto de los elementos td de una
tabla"
    id="boton3"><br>
    <p id="parrafo1">
        Texto del primer párrafo</p>
    <table style="border-width: 1px; border-style: double">
        <tr>
            <td>celda 1,1</td>
            <td>celda 1,2</td>
            <td>celda 2,1</td>
            <td>celda 2,2</td>
        </tr>
    </table>
</body>
</html>

```

En esta página tenemos 3 botones, un párrafo con texto y una tabla. Para definir las funciones que realizará cada uno de los botones, las definiremos en el archivo funciones1.js:

```

var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x = $("#boton1");
    x.click(extraerTexto);
    x = $("#boton2");
    x.click(modificarTexto);
    x = $("#boton3");
    x.click(modificarDatosTabla);
}

function extraerTexto() {
    var x = $("#parrafo1");
    alert(x.text());
}

function modificarTexto() {
    var x = $("#parrafo1");
    x.text("Nuevo texto del párrafo");
}

function modificarDatosTabla() {
    var x = $("td");
    x.text("texto nuevo");
}

```

Al presionar el primer botón, se ejecutará la función `extraerTexto()`. Con este obtenemos la referencia al párrafo mediante su id (recordemos que en una página todos los valores de los id son distintos), después mostramos en un `alert` el contenido extraído mediante el método `text()`.



Con la función `modificarTexto()` cambiaremos el contenido del párrafo:

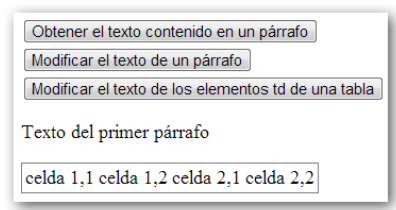
```
function modificarTexto() {
    var x = $("#parrafo1");
    x.text("Nuevo texto del párrafo");
}
```

Hemos obtenido la referencia del párrafo mediante su id y posteriormente llamamos al método `text` enviándole el nuevo string a mostrar.

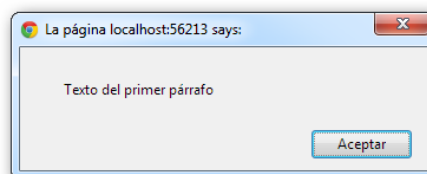
Por último, con la función `modificarDatosTabla()` creamos un objeto de la clase `jQuery` con la referencia a todos los elementos `td` del documento (es decir los `td` de todas las tablas) y posteriormente mediante el método `text` modifica el contenido de todos ellos (todos los `td` del documento se cambian por el string "nuevo texto")

```
function modificarDatosTabla() {
    var x = $("td");
    x.text("texto nuevo");
}
```

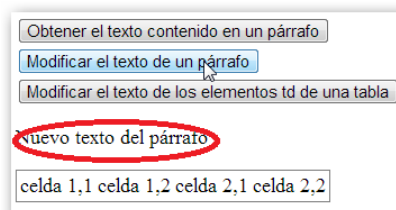
Veamos en ejecución el funcionamiento del mismo:



**Figura 4.20.** Imagen de la página cuando carga.



**Figura 4.21.** Ventana cuando pulsamos el primer botón.



**Figura 4.22.** Resultado de pulsar en el segundo botón.

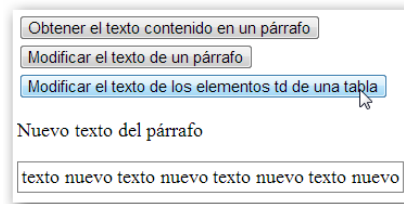


Figura 4.23. Cambio de texto en las celdas al pulsar el tercer botón.

#### 4.9.4.2. Métodos `attr(nombre de propiedad)`, `attr(nombre de propiedad,valor)` y `removeAttr(nombre de propiedad)`

Estos métodos nos permitirán agregar propiedades a un elemento HTML y también recuperar el valor de una propiedad. Para recuperar el valor de una propiedad lo haremos de la siguiente manera:

```
$(elemento).attr(nombre de propiedad)
```



Debéis saber que si hay muchos elementos que recupera la función `$`, solo retorna la propiedad del primero.

La forma de fijar el valor de una propiedad será como se ve en este código: `()`:

```
$(elemento).attr(nombre de propiedad, valor)
```



En este caso, si hay muchos elementos que recupera la función `$`, se inicializan para todos.

Y por último, para eliminar una propiedad de un elemento o conjunto de elementos tenemos:

```
$(elemento).removeAttr(nombre de propiedad)
```

Veamos estos casos implementándolos en un ejemplo. Vamos a definir una tabla sin el atributo `border`. Implementaremos tres botones, el primero le añadirá la propiedad `border` con el valor 1. El segundo botón recuperará y mostrará el valor del atributo `border` y por último, si se presiona el ultimo botón eliminaremos la propiedad `border`. Partimos de la página1.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Problema</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <input type="button" id="boton1" value="Añadir propiedad border"><br>
  <input type="button" id="boton2" value="Recuperar valor de la propiedad
border"><br>
  <input type="button" id="boton3" value="Eliminar la propiedad border">
  <table id="tabla1">
    <tr>
      <td>1111111111</td>
      <td>1111111111</td>
      <td>1111111111</td>
      <td>1111111111</td>
    </tr>
    <tr>
      <td>2222222222</td>
      <td>2222222222</td>
      <td>2222222222</td>
      <td>2222222222</td>
    </tr>
    <tr>
      <td>3333333333</td>
      <td>3333333333</td>
      <td>3333333333</td>
      <td>3333333333</td>
    </tr>
  </table>
</body>
</html>
```

Implementemos ahora las funciones necesarias para realizar lo anteriormente comentado en el archivo funciones1.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x = $("#boton1");
  x.click(agregarPropiedadBorder);
  x = $("#boton2");
  x.click(recuperarPropiedadBorder);
  x = $("#boton3");
  x.click(eliminarPropiedadBorder);
}
```

```

}

function agregarPropiedadBorder() {
    var x = $("#tabla1");
    x.attr("border", "1");
}

function recuperarPropiedadBorder() {
    var x = $("#tabla1");
    if (x.attr("border") != undefined)
        alert(x.attr("border"));
    else
        alert("No está definida la propiedad border en la tabla");
}

function eliminarPropiedadBorder() {
    var x = $("#tabla1");
    x.removeAttr("border");
}

```

En este ejemplo, cuando cargamos la página y llamamos al método de `inicializarEventos()`, asignamos a cada botón la función que van a realizar.

```

function inicializarEventos() {
    var x = $("#boton1");
    x.click(agregarPropiedadBorder);
    x = $("#boton2");
    x.click(recuperarPropiedadBorder);
    x = $("#boton3");
    x.click(eliminarPropiedadBorder);
}

```

Mediante su id les hemos asignado cada función al evento click. Cuando se presiona el primer botón:

```

function agregarPropiedadBorder() {
    var x = $("#tabla1");
    x.attr("border", "1");
}

```

La función `agregarPropiedadBorder()` obtendrá la referencia de la tabla mediante su id y después llamaremos al método `attr`, pasándole como primer parámetro el nombre de la propiedad a agregar y como segundo el valor de la propiedad.

Cuando presionemos el segundo botón:

```

function recuperarPropiedadBorder() {
    var x = $("#tabla1");
    if (x.attr("border") != undefined)
        alert(x.attr("border"));
    else
        alert("No está definida la propiedad border en la tabla");
}

```

La función `recuperarPropiedadBorder()` obtendrá la referencia a la tabla y después llamamos al método `attr`, enviándole como parámetro el nombre de la propiedad que queremos recuperar. Si devuelve `undefined`, significa que no tiene dicha propiedad el elemento HTML, en caso contrario retorna su valor y procedemos a mostrarlo mediante un `alert`.

Cuando presionemos el tercer botón:

```
function eliminarPropiedadBorder() {
    var x = $("#tabla1");
    x.removeAttr("border");
}
```

La función `recuperarPropiedadBorder()` obtendrá la referencia a la tabla mediante su id y llamará al método `removeAttr` con el nombre de la propiedad a eliminar.

Si ejecutamos la página, veremos el siguiente resultado:

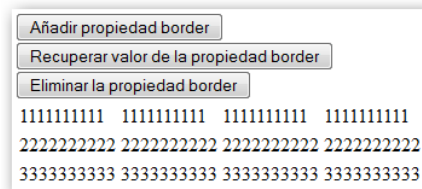


Figura 4.24. Página cuando carga el navegador.

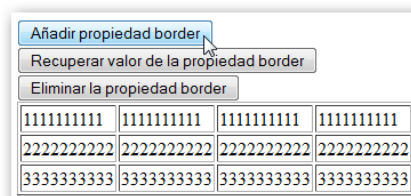


Figura 4.25. Cuando pulsamos el botón añadimos el borde.

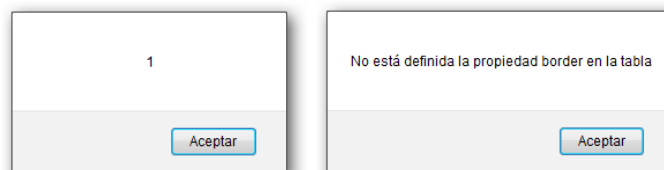


Figura 4.26. Pulsación del segundo botón, con la propiedad definida y si borde.

Si pulsamos sobre el tercer botón, dejaremos la página tal y como estaba cuando carga la página por primera vez.

### 4.9.4.3. Métodos addClass y removeClass

Con estos dos métodos proporcionados por el objeto jQuery, podremos asociar y desasociar una clase a un elemento o conjunto de elementos HTML. Para que lo veáis mas claro, vamos a implementar un ejemplo en el que tenemos un div con un conjunto de párrafos y dos botones, que cuando se presione el primero, le asociaremos una clase y cuando se presione el otro desasociarlo de dicha clase.

Para ello, insertaremos en la página1.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo07</title>
  <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <input type="button" id="boton1" value="Asociar clase">
  <input type="button" id="boton2" value="Desasociar clase">
  <div id="descripcion">
    <p>
      HTML es el lenguaje que se emplea para el desarrollo de páginas de
      internet.</p>
    <p>
      Este lenguaje está constituido de elementos que el navegador
      interpreta y las
      despliega en la pantalla de acuerdo a su objetivo. Veremos que hay
      elementos para
      disponer imágenes sobre una página, hipervínculos que nos permiten
      dirigirnos a
      otra página, listas, tablas para tabular datos, etc.</b>
    <p>
      Para poder crear una página HTML se requiere un simple editor de
      texto (en nuestro
      caso emplearemos este sitio) y un navegador de internet
      (IEExplorer, FireFox etc.),
      emplearemos el navegador que en este preciso momento está
      utilizando (recuerde que
      usted está viendo en este preciso momento una página HTML con su
      navegador).</p>
    <p>
      Lo más importante es que en cada concepto desarrolle los
      ejercicios propuestos
      y modifique los que se presentan ya resueltos.<p>
    <p>
      Este curso lo que busca es acercar el lenguaje HTML a una persona
      que nunca antes trabajó con el mismo. No pretende mostrar todas los elementos
      HTML en forma alfabética.</p>
    <p>
      Como veremos, de cada concepto se presenta una parte teórica, en
      la que se da una explicación completa, luego se pasa a la sección del
      ejercicio resuelto donde podemos
```

ver el contenido de la página HTML y cómo la visualiza el navegador. Por último y tal vez la sección más importante de este tutorial es donde se propone que usted haga páginas en forma autónoma (donde realmente podrá darse cuenta si el concepto quedó firme).</p>

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

Una vez insertada la página html, necesitamos implementar las funciones que realizarán lo anteriormente descrito. Las insertaremos en el archivo funciones.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x = $("#boton1");
    x.click(asociarClase);
    x = $("#boton2");
    x.click(desasociarClase);
}

function asociarClase() {
    var x = $("#descripcion");
    x.addClass("recuadro");
}

function desasociarClase() {
    var x = $("#descripcion");
    x.removeClass("recuadro");
}
}
```

Y por último, necesitaremos insertar la definición de la clase en el archivo estilos.css:

```
.recuadro {
    background-color:#ffffcc;
    font-family:verdana;
    font-size:14px;

    border-top-width:1px;
    border-right-width:3px;
    border-bottom-width:3px;
    border-left-width:1px;

    border-top-style:dotted;
    border-right-style:solid;
    border-bottom-style:solid;
    border-left-style:dotted;

    border-top-color:#ffaa00;
    border-right-color:#ff0000;
    border-bottom-color:#ff0000;
    border-left-color:#ffaa00;
}
```

Es muy importante que en el archivo HTML deberemos indicar donde se encuentran los archivos js y css:

```
<link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
<script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
<script src="Scripts/funciones1.js" type="text/javascript"></script>
```

Como en todos los casos, cuando cargamos la página y llamamos al método de inicializarEventos(), asignamos a cada botón la función que van a realizar.

```
function inicializarEventos() {
    var x = $("#boton1");
    x.click(asociarClase);
    x = $("#boton2");
    x.click(desasociarClase);
}
```

Después, cuando presionemos el primer botón se ejecuta la función que asociará la clase de la hoja de estilo, mediante la llamada al método addClass(), pasándole como parámetro el nombre de la clase (dicha clase debe estar definida en la hoja de estilo (css)).

```
function asociarClase() {
    var x = $("#descripcion");
    x.addClass("recuadro");
}
```

Para desasociar dicha clase, pulsaremos el botón correspondiente, y se ejecutará la función:

```
function desasociarClase() {
    var x = $("#descripcion");
    x.removeClass("recuadro");
}
```

En este método, obtenemos la referencia al objeto div mediante su id para posteriormente llamar al método removeClass(), pasándole como parámetro la clase que queremos desasignar.



Veamos la ejecución de dicho ejemplo:

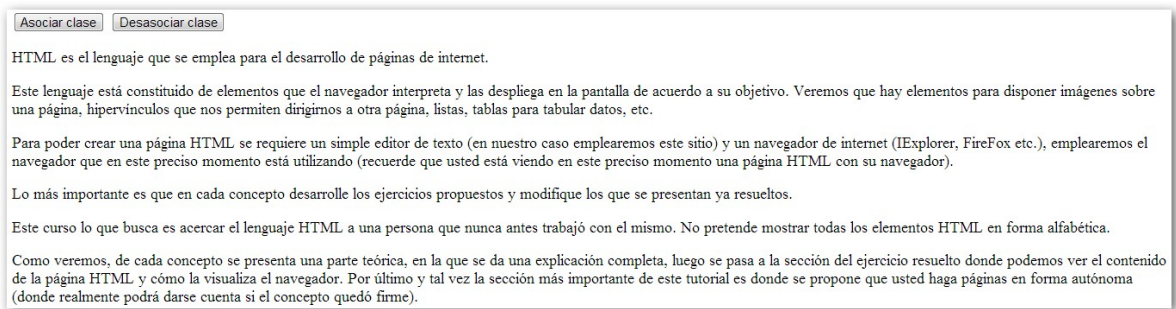


Figura 4.27. Vista del navegador cuando se carga la página.

Si pulsamos sobre el botón de “Asociar clase”, veremos como se encuadra el párrafo, sin que se refresque la página:

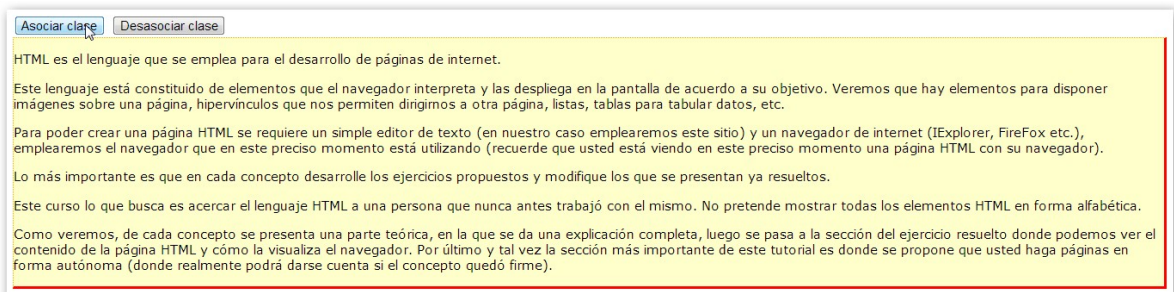


Figura 4.28. Vista cuando se ha pulsado el botón de asociar la clase.

Si pulsamos sobre el botón de “Desasociar clase”, volveremos a dejar como estaba el formulario de la página.

#### 4.9.4.4. Métodos `html()` y `html(valor)`

Con estos métodos vamos a poder insertar y recuperar el código html de una página:

- Con el método **`html([bloque html])`** vamos a poder agregar un bloque de html a partir de un elemento de la página. Es como la propiedad `innerHTML` del DOM.
- Y el método **`html()`** nos va a devolver el bloque html contenido en la página a partir del elemento html al que hace referencia el objeto jQuery.

Veámoslo mejor con un ejemplo, el ejemplo08, en el que tenemos dos botones, que al presionar el primero creara un formulario de una forma dinámica, que solicite el nombre de usuario y su clave. Cuando presionemos el segundo, mostraremos todos los elementos HTML del formulario previamente creado. Para ello debemos partir de la siguiente página1.html:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo08</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <input type="button" id="boton1" value="Mostrar formulario">
  <input type="button" id="boton2" value="Mostrar elementos html del
formulario"><br>
  <div id="formulario">
  </div>
</body>
</html>

```

Posteriormente, debemos definir las funciones en el archivo funciones.js:

```

var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("#boton1");
  x.click(presionBoton1);
  x = $("#boton2");
  x.click(presionBoton2);
}

function presionBoton1() {
  var x;
  x = $("#formulario");

  x.html('<form>Ingrese nombre:<input type="text" id="nombre"><br>Ingrese
clave:<input type="text" id="clave"><br><input type="submit"
value="confirmar"></form>');
}

function presionBoton2() {
  var x;
  x = $("#formulario");
  alert(x.html());
}

```

Como podréis ver, cuando presionemos el primer botón se va a crear un objeto jQuery que tendrá la referencia del div y con el método html creará un bloque en su interior:

```

function presionBoton1() {
  var x;
  x = $("#formulario");

```

```
x.html('<form>Ingrese nombre:<input type="text" id="nombre"><br>Ingrese
clave:<input type="text" id="clave"><br><input type="submit"
value="confirmar"></form>');
}
```

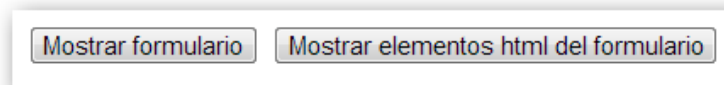
Cuando pulsemos sobre el segundo botón, se mostrará en un alert el contenido HTML actual del div:

```
function presionBoton2() {
    var x;
    x = $("#formulario");
    alert(x.html());
}
```



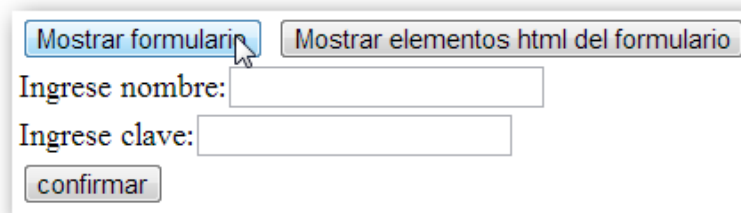
Debéis tener especial cuidado y diferenciar bien los métodos **html(valor)** y **text(valor)**, recordad que el segundo solo agrega texto a un elemento HTML, el primero agrega un bloque completo de código html.

Si ejecutamos nuestro ejemplo, cuando se carga el código veremos únicamente dos botones:



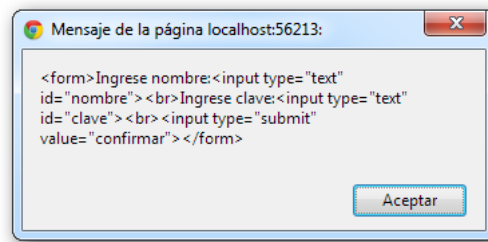
**Figura 4.29.** Vista de la página cuando carga por primera vez.

Si pulsamos sobre el botón “Mostrar formulario”, se cargarán en la página los elementos que hemos pasado mediante el método html(“codigoHTML”):



**Figura 4.30.** Vista de la página tras pulsar el primer boton.

Si pulsamos sobre el botón de “Mostrar elementos html del formulario”, un alert nos mostrará el código insertado:



**Figura 4.31. Vista de la ventana de alerta con el código insertado.**



Debéis tener en cuenta que, en el desarrollo de sitios web, se debe tener especial atención con la creación de bloques en forma dinámica, por lo que, si los usuarios finales de dicho sitio no tienen activado JavaScript, les sería imposible acceder a dichas características.

#### 4.9.4.5. Métodos show, hide, fadeOut, fadeIn

Aparte de los métodos que hemos visto hasta ahora, jQuery nos facilita una serie de efectos visuales para implementar en nuestras páginas.

En este punto vamos a aprender como utilizar los métodos hide() y show(), para ocultar y mostrar elementos HTML. Y los métodos fadeOut() y fadeIn() para hacerlo con una pequeña animación.

Veámoslo a través de un ejemplo, en el que vamos a realizar una página que muestre un recuadro con texto. Vamos a insertar 4 botones, dos botones, uno que oculte lentamente el cuadro y el otro que lo muestre rápidamente, y otros dos botones, uno que oculte utilizando el método fadeOut() y el otro que lo muestre mediante el uso del método fadeIn(). Empezaremos insertando el siguiente código en la página1.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo20</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
  <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <input type="button" id="boton1" value="Ocultar recuadro (hide)">
  <input type="button" id="boton2" value="Mostrar recuadro (show)">
  <input type="button" id="boton3" value="Ocultar recuadro (fadeOut)">
  <input type="button" id="boton4" value="Mostrar recuadro (fadeIn)">
  <div id="descripcion" class="recuadro">
```

```

<p>
    HTML es el lenguaje que se emplea para el desarrollo de páginas de
internet.</p>
<p>
    Este lenguaje está constituido de elementos que el navegador
interpreta y las
    despliega en la pantalla de acuerdo a su objetivo. Veremos que hay
elementos para
    disponer imágenes sobre una página, hipervínculos que nos permiten
dirigirnos a
    otra página, listas, tablas para tabular datos, etc.</b>
<p>
    Para poder crear una página HTML se requiere un simple editor de
texto (en nuestro
    caso emplearemos este sitio) y un navegador de internet
(IEExplorer, FireFox etc.),
    emplearemos el navegador que en este preciso momento está
utilizando (recuerde que
    usted está viendo en este preciso momento una página HTML con su
navegador).</p>
<p>
    Lo más importante es que en cada concepto desarrolle los
ejercicios propuestos
    y modifique los que se presentan ya resueltos.<p>
<p>
    Este curso lo que busca es acercar el lenguaje HTML a una persona
que nunca antes
    trabajó con el mismo. No pretende mostrar todas los elementos HTML
en forma alfabética.</p>
<p>
    Como veremos, de cada concepto se presenta una parte teórica, en
la que se da una
    explicación completa, luego se pasa a la sección del ejercicio
resuelto donde podemos
    ver el contenido de la página HTML y cómo la visualiza el
navegador. Por último
    y tal vez la sección más importante de este tutorial es donde se
propone que usted
    haga páginas en forma autónoma (donde realmente podrá darse cuenta
si el concepto
    quedó firme).</p>
</div>
</body>
</html>

```

En esta página, lo único que hemos hecho ha sido insertar 4 botones y un párrafo con texto al que le vamos a dar el efecto deseado. Para ello necesitamos insertar las funciones en el archivo funciones1.js

```

var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x = $("#boton1");
    x.click(ocultarRecuadroHide);
    x = $("#boton2");
    x.click(mostrarRecuadroShow);
}

```

```
x = $("#boton3");
x.click(ocultarRecuadroFadeOut);
x = $("#boton4");
x.click(mostrarRecuadroFadeIn);
}

function ocultarRecuadroHide() {
    var x = $("#descripcion");
    x.hide("slow");
}

function mostrarRecuadroShow() {
    var x = $("#descripcion");
    x.show("fast");
}

function ocultarRecuadroFadeOut() {
    var x = $("#descripcion");
    x.fadeOut("slow");
}

function mostrarRecuadroFadeIn() {
    var x = $("#descripcion");
    x.fadeIn("slow");
}
```

Más adelante explicaremos cada una de las funciones. Además, en este ejemplo necesitamos un archivo de estilos que lo vamos a insertar en el archivo estilos.css:

```
.recuadro {
    background-color:#ffffcc;
    font-family:verdana;
    font-size:14px;

    border-top-width:1px;
    border-right-width:3px;
    border-bottom-width:3px;
    border-left-width:1px;

    border-top-style:dotted;
    border-right-style:solid;
    border-bottom-style:solid;
    border-left-style:dotted;

    border-top-color:#ffaa00;
    border-right-color:#ff0000;
    border-bottom-color:#ff0000;
    border-left-color:#ffaa00;
}
```

Vamos a ir explicando cada una de las funciones contenidas en el archivo de funciones.js. La primera función después de inicializar los eventos y asignárselos a cada uno de los botones es la función de ocultarRecuadroHide() que se ejecutará cuando presionemos el botón de "Ocultar recuadro (hide)":

```
function ocultarRecuadroHide() {
    var x = $("#descripcion");
    x.hide("slow");
}
```

En esta obtenemos la referencia del div mediante su id y procedemos a llamar al método **hide()**, pasándole el string "slow", con esto logramos que se oculte el recuadro lentamente.

El funcionamiento de la función mostrarRecuadroShow() es muy similar a la anterior:

```
function mostrarRecuadroShow() {
    var x = $("#descripcion");
    x.show("fast");
}
```

En esta llamamos a la función show, pero en este caso le pasamos como parámetro el string "fast". De esta manera, aparecerá de una manera más rápida.

Hay varias formas para llamar a los métodos **show y hide, fadeOut y fadeIn**:

- Lo muestra u oculta de forma instantánea: show(), hide(), fadeOut() y fadeIn().
- Lo muestra u oculta con una animación rápida: show("fast"), hide("fast"), ...
- Lo muestra u oculta con una animación normal: show("normal"), hide("normal"), ...
- Lo muestra u oculta con una animación lenta: show("slow"), hide("slow"), ...
- Lo muestra u oculta con una animación que tarda tantos milisegundos como le indicamos: show([cantidad de milisegundos]) igual con el resto de metodos...
- Lo muestra u oculta con una animación que tarda tantos milisegundos como le indicamos y ejecuta al final la función que le pasamos como segundo parámetro: show([cantidad de milisegundos],[función]) igual con el resto de metodos...

La función ocultarRecuadroFadeOut() se ejecutará cuando presionemos el botón de "Ocultar recuadro":

```
function ocultarRecuadroFadeOut() {
    var x = $("#descripcion");
    x.fadeOut("slow");
}
```

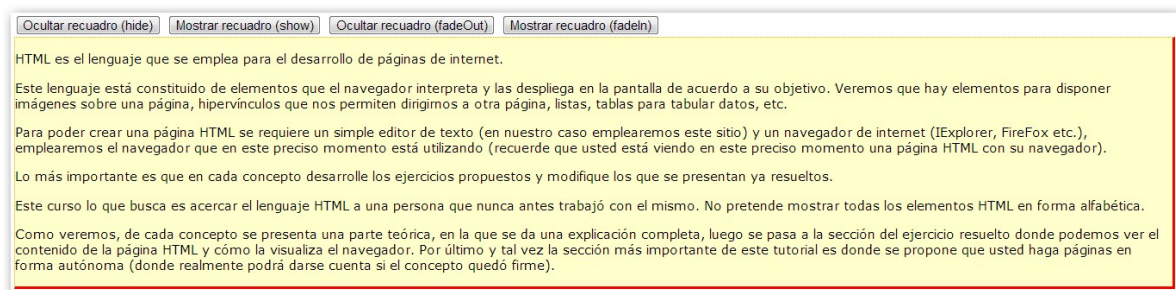
En esta función, obtendremos de nuevo la referencia del div mediante su id y procederemos a llamar al método **fadeOut()** pasándole el string "slow", de esta manera lograremos que se oculte el recuadro, decolorándose lentamente hasta desaparecer.

Por último, la función mostrarRecuadroFedeln:

```
function mostrarRecuadroFadeIn() {
    var x = $("#descripcion");
    x.fadeIn("slow");
}
```

Llamará a la función **fadeIn()** pasando como parámetro el string "slow".

Si ejecutamos nuestra página, veremos la siguiente imagen:



**Figura 4.32. Vista de la página cuando carga por primera vez.**

Cada uno de los recuadros, muestra u oculta dicho recuadro de maneras diferentes. Aconsejo que se hagan diferentes pruebas con cada uno de los métodos, cambiando los parametros al realizar las llamadas a cada uno de los métodos.

#### 4.9.4.6. Método fadeTo

Este es otro de los metodos de jQuery, que nos va a facilitar dar efectos en nuestras páginas.

El método **fadeTo()** sirve para modificar la opacidad de un elemento, y el efecto es llevar la opacidad actual hasta el valor que nosotros le pasamos al método fadeTo.





El valor de la opacidad es un número real entre 0 y 1. 1 significa sin opacidad y 0 es transparente.

Este método podemos inicializarlo de dos formas:

- **fadeTo([velocidad],[valor de opacidad])** aquí le indicaremos la velocidad de transición entre el estado actual y el estado final, y un segundo parámetro, que será el valor de la opacidad. Podemos hacerlo mediante:
  - ✓ Sslow/normal/fast.
  - ✓ Valor indicado en milisegundos.
- **fadeTo([velocidad],[valor de opacidad],[función])** Esta segunda estructura de la función podemos pasarle un tercer parámetro, que ejecutará una función cuando finalice la transición.



Hay que tener en cuenta que por más que indiquemos el valor 0 en opacidad al método fadeTo, el espacio que ocupa el elemento en la página seguirá ocupado por un recuadro vacío.

Veamos un ejemplo de su implementación. Vamos a realizar una página que muestre un recuadro con texto, y tendremos dos botones, uno que va a cambiar la opacidad lentamente hasta el valor 0.5 y el otro que lo muestre lentamente hasta el valor 1. Insertamos el siguiente código en la página1.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
  <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <input type="button" id="boton1" value="Reducir opacidad">
  <input type="button" id="boton2" value="Aumentar opacidad">
  <div id="descripcion" class="recuadro">
    <p>
      HTML es el lenguaje que se emplea para el desarrollo de páginas de
      internet.</p>
    <p>
```

Este lenguaje está constituido de elementos que el navegador interpreta y los despliega en la pantalla de acuerdo a su objetivo. Veremos que hay elementos para disponer imágenes sobre una página, hipervínculos que nos permiten dirigirnos a otra página, listas, tablas para tabular datos, etc.

<p>

Para poder crear una página HTML se requiere un simple editor de texto (en nuestro caso emplearemos este sitio) y un navegador de internet (IEExplorer, FireFox etc.), emplearemos el navegador que en este preciso momento está utilizando (recuerde que usted está viendo en este preciso momento una página HTML con su navegador).

<p>

Lo más importante es que en cada concepto desarrolle los ejercicios propuesto y modifique los que se presentan ya resueltos.

<p>

Este curso lo que busca es acercar el lenguaje HTML a una persona que nunca antes trabajó con el mismo. No pretende mostrar todas los elementos HTML en forma alfabética.

<p>

Como veremos, de cada concepto se presenta una parte teórica, en la que se da una explicación completa, luego se pasa a la sección del ejercicio resuelto donde podemos ver el contenido de la página HTML y cómo la visualiza el navegador. Por último y tal vez la sección más importante de este tutorial es donde se propone que usted

haga páginas en forma autónoma (donde realmente podrá darse cuenta si el concepto

quedó firme).

</div>

</body>

</html>

Hasta ahora, tenemos una página que contiene un div con un parrafo con texto, y dos botones, para aumentar y reducir la opacidad. Definamos las funciones necesarias en el archivo funciones.js

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x = $("#boton1");
    x.click(reducirOpacidadRecuadro);
    x = $("#boton2");
    x.click(aumentarOpacidadRecuadro);
}

function reducirOpacidadRecuadro() {
    var x = $("#descripcion");
    x.fadeTo("slow", 0.5);
}

function aumentarOpacidadRecuadro() {
    var x = $("#descripcion");
    x.fadeTo("slow", 1);
}
```

Un poco más adelante explicaremos sus funciones, pero ahora necesitamos crear un archivo de estilos, para poder aplicarlos. El archivo lo llamaremos estilos.css:

```
.recuadro {  
  background-color:#ffffcc;  
  font-family:verdana;  
  font-size:14px;  
  
  border-top-width:1px;  
  border-right-width:3px;  
  border-bottom-width:3px;  
  border-left-width:1px;  
  
  border-top-style:dotted;  
  border-right-style:solid;  
  border-bottom-style:solid;  
  border-left-style:dotted;  
  
  border-top-color:#ffaa00;  
  border-right-color:#ff0000;  
  border-bottom-color:#ff0000;  
  border-left-color:#ffaa00;  
}
```

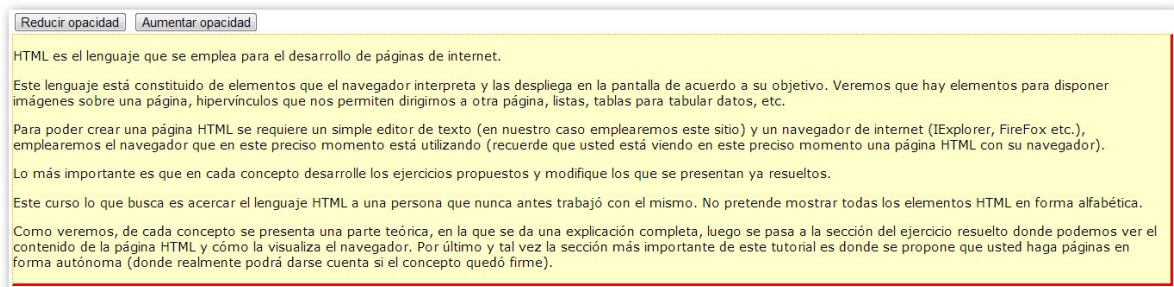
En el archivo de funciones.js, hemos declarado dos funciones. Una para reducir la opacidad, al que le pasaremos como valor 0.5:

```
function reducirOpacidadRecuadro() {  
  var x = $("#descripcion");  
  x.fadeTo("slow", 0.5);  
}
```

La segunda función declarada es la de aumentarOpacidadRecuadro(), que será llamada cuando presionamos el botón de aumentar la opacidad. En esta función vamos a llamar al método fadeTo pasándole como valor la velocidad "slow" y un valor 1 en opacidad (significa opacidad total):

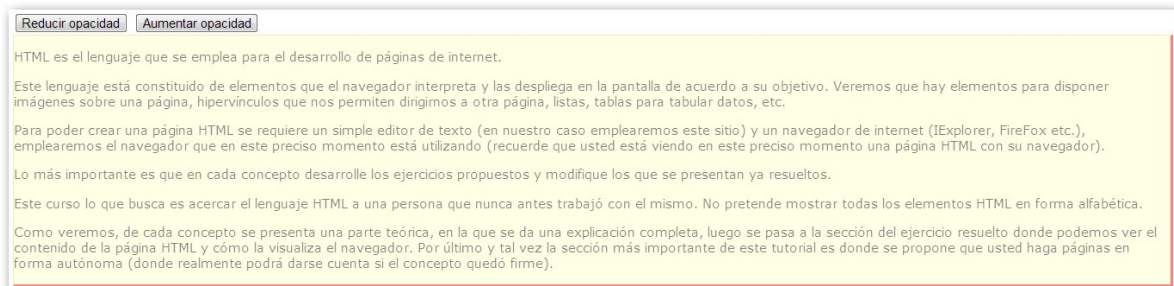
```
function aumentarOpacidadRecuadro() {  
  var x = $("#descripcion");  
  x.fadeTo("slow", 1);  
}
```

Veamos nuestro ejemplo funcionando, cuando cargue la página, observaremos lo siguiente:



**Figura 4.33. Vista de la página cuando carga por primera vez.**

Si pulsamos sobre el botón de Reducir opacidad, vemos como disminuye el color o aumenta la transparencia del mismo:



**Figura 4.34. Vista de la página cuando pulsamos sobre el botón de “Reducir opacidad”.**

Si pulsamos sobre el botón de “Aumentar opacidad”, volveremos a situar su valor en 1 y se mostrará de nuevo como cuando cargó por primera vez.

#### 4.9.4.7. Método toggle

Si recordáis unos puntos anteriores, teníamos dos metodos, `hide()` y `show()`, que nos servían para ocultar y mostrar respectivamente. El método **`toggle()`** realiza la misma acción, pero en una sola declaración, es decir, si está visible pasa a oculto y si se encuentra oculto pasa a visible.

Vamos a verlo con un sencillo ejemplo, en el que el mismo bloque de texto de anteriores ejemplos, pasará de visible a oculto lentamente y viceversa al presionar un único botón.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
```

```

    <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <input type="button" id="boton1" value="Mostrar/Ocultar">
    <div id="descripcion" class="recuadro">
        <p>
            HTML es el lenguaje que se emplea para el desarrollo de páginas de
            internet.</p>
        <p>
            Este lenguaje está constituido de elementos que el navegador
            interpreta y las despliega en la pantalla de acuerdo a su objetivo. Veremos
            que hay elementos para disponer imágenes sobre una página, hipervínculos
            que nos permiten dirigirnos a otra página, listas, tablas para tabular
            datos, etc.</b>
        <p>
            Para poder crear una página HTML se requiere un simple editor de
            texto (en nuestro caso emplearemos este sitio) y un navegador de internet
            (IEExplorer, FireFox etc.), emplearemos el navegador que en este preciso
            momento está utilizando (recuerde que usted está viendo en este preciso
            momento una página HTML con su navegador).</p>
        <p>
            Lo más importante es que en cada concepto desarrolle los
            ejercicios propuestos y modifique los que se presentan ya
            resueltos.</p>
        <p>
            Este curso lo que busca es acercar el lenguaje HTML a una persona
            que nunca antes trabajó con el mismo. No pretende mostrar todas los elementos HTML en
            forma alfabética.</p>
        <p>
            Como veremos, de cada concepto se presenta una parte teórica, en
            la que se da una explicación completa, luego se pasa a la sección del
            ejercicio resuelto donde podemos ver el contenido de la página HTML y cómo
            la visualiza el navegador. Por último y tal vez la sección más importante de
            este tutorial es donde se propone que usted haga páginas en forma autónoma
            (donde realmente podrá darse cuenta si el concepto
            quedó firme).</p>
    </div>
</body>
</html>

```

Declaramos nuestra función necesaria en el archivo funciones.js:

```

var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x = $("#boton1");
    x.click(ocultarMostrarRecuadro);
}

function
    ocultarMostrarRecuadro() {
        var x = $("#descripcion");
        x.toggle("slow");
    }
}

```

Y por último, definimos nuestra hoja de estilos en el archivo estilos.css

```
.recuadro {  
  background-color:#ffffcc;  
  font-family:verdana;  
  font-size:14px;  
  
  border-top-width:1px;  
  border-right-width:3px;  
  border-bottom-width:3px;  
  border-left-width:1px;  
  
  border-top-style:dotted;  
  border-right-style:solid;  
  border-bottom-style:solid;  
  border-left-style:dotted;  
  
  border-top-color:#ffaa00;  
  border-right-color:#ff0000;  
  border-bottom-color:#ff0000;  
  border-left-color:#ffaa00;  
}
```

La función `ocultarMostrarRecuadro()`, será la que directamente realizará el cambio de estado, dependiendo en el que se encuentre cuando se pulse el botón, de oculto a visible, o viceversa:

```
function ocultarMostrarRecuadro() {  
  var x = $("#descripcion");  
  x.toggle("slow");  
}
```

Hemos obtenido la referencia al div y llamamos al método `toggle()` del objeto jQuery, realizando la transición de una forma lenta ya que le pasamos como parámetro el string "slow".

#### 4.9.4.8. Método each

jQuery proporciona un método con el que podemos realizar la iteración por los elementos, es decir, asociará una función que se ejecutará por cada elemento que contenga la lista del objeto jQuery. La sintaxis del iterador **each()** es:

```
var x;
x=$([elementos]);
x.each([nombre de funcion])
```

Para verlo más claro, realizaremos un ejemplo, en el que resaltaremos con fondo amarillo todos los párrafos que tengan menos de 100 caracteres, y en rojo los que tengan más de 100 caracteres. Creamos una página que tiene un <div> y 4 párrafos de diferentes caracteres. Insertamos el siguiente código en la página1.htm:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <div id="párrafos">
    <p>
      párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 -
      párrafo 1 - párrafo
      1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 -
      párrafo 1 - párrafo
      1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 -
      párrafo 1 - párrafo
      1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 -
      párrafo 1 - párrafo
      1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1</p>
    <p>
      párrafo 2 - párrafo 2 - párrafo 2 - párrafo 2</p>
    <p>
      párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 -
      párrafo 3 - párrafo
      3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 -
      párrafo 3 - párrafo
      3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 -
      párrafo 3 - párrafo
      3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 -
      párrafo 3 - párrafo
      3</p>
    <p>
      párrafo 4 - párrafo 4 - párrafo 4 - párrafo 4 - párrafo 4</p>
  </div>
</body>
</html>
```

Declararemos las funciones necesarias en el archivo funciones.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x;
    x = $("#parrafos p");
    x.each(resaltarParrafos);
}

function resaltarParrafos() {
    var x = $(this);
    if (x.text().length < 100) {
        x.css("background-color", "#ff0");
    } else {
        x.css("background-color", "#f00");
    }
}
```

Lo primero que haremos en la función inicializarEventos(), es obtener la lista de párrafos contenidos en el div y luego mediante el método each:

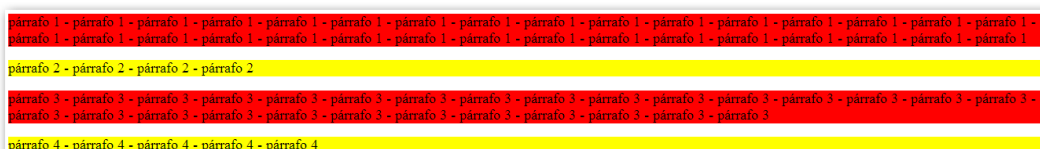
```
function inicializarEventos() {
    var x;
    x = $("#parrafos p");
    x.each(resaltarParrafos);
}
```

La función resaltarParrafos() se ejecutará por cada uno de los párrafos contenidos en el objeto jQuery:

```
function resaltarParrafos() {
    var x = $(this);
    if (x.text().length < 100) {
        x.css("background-color", "#ff0");
    } else {
        x.css("background-color", "#f00");
    }
}
```

Lo primero obtenemos la referencia del párrafo a procesar mediante la función \$ y pasando `this` como parámetro.

Después accedemos al método `text()` que nos devuelve el texto del párrafo propiamente dicho. Este método devuelve un string por lo que, mediante su atributo `length`, contaremos la cantidad de caracteres que tiene. Dependiendo de su resultado, si es inferior o superior a 100, cambiaremos el color de fondo del párrafo. Si ejecutamos la página en el navegador, veremos como los párrafos menores de 100 tienen el fondo amarillo, y los superiores, lo tienen rojo:



párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1 - párrafo 1  
 párrafo 2 - párrafo 2 - párrafo 2 - párrafo 2  
 párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3 - párrafo 3  
 párrafo 4 - párrafo 4 - párrafo 4 - párrafo 4 - párrafo 4



#### 4.9.4.9. Métodos para manipular nodos del DOM

En este punto vamos a ver otros métodos con los que poder manipular elementos del DOM mediante jQuery. Lo mejor será verlo con un ejemplo.

Tendremos una página web, que va a contener una lista con cuatro ítems, y mediante una serie de botones, vamos a manipular los elementos de dicha lista. Partimos de una página1.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo18</title>
  <script src="../../Scripts/jquery-1.8.2.min.js"
type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <h1>
    Métodos para manipular elementos del DOM con jQuery.</h1>
  <ul>
    <li>Primer elemento.</li>
    <li>Segundo elemento.</li>
    <li>Tercer elemento.</li>
    <li>Cuarto elemento.</li>
  </ul>
  <input type="button" id="boton1" value="Eliminar la lista completa."><br>
  <input type="button" id="boton2" value="Restaurar Lista"><br>
  <input type="button" id="boton3" value="Añadir un elemento al final de la
lista"><br>
  <input type="button" id="boton4" value="Añadir un elemento al principio de
la lista"><br>
  <input type="button" id="boton5" value="Eliminar el último elemento."><br>
  <input type="button" id="boton6" value="Eliminar el primer elemento."><br>
  <input type="button" id="boton7" value="Eliminar el primero y segundo
elemento."><br>
  <input type="button" id="boton8" value="Eliminar los dos últimos."><br>
</body>
</html>
```

En el archivo funciones1.js vamos a insertar el siguiente código que posteriormente iremos explicando función a función:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("#boton1");
  x.click(eliminarElementos)
  x = $("#boton2");
  x.click(restaurarLista)
  x = $("#boton3");
  x.click(anadirElementoFinal)
  x = $("#boton4");
  x.click(anadirElementoPrincipio)
```

```
x = $("#boton5");
x.click(eliminarElementoFinal)
x = $("#boton6");
x.click(eliminarElementoPrincipio)
x = $("#boton7");
x.click(eliminarPrimeroSegundo)
x = $("#boton8");
x.click(eliminarDosUltimos)
}

function eliminarElementos() {
    var x;
    x = $("ul");
    x.empty();
}

function restaurarLista() {
    $("ul").html('<li>Primer item.</li><li>Segundo item.</li><li>Tercer
item.</li><li>Cuarto item.</li>');
}

function anadirElementoFinal() {
    var x;
    x = $("ul");
    x.append("<li>otro item al final</li>");
}

function anadirElementoPrincipio() {
    var x;
    x = $("ul");
    x.prepend("<li>otro item al principio</li>");
}

function eliminarElementoFinal() {
    var x;
    x = $("li");
    var cantidad = x.length;
    x = x.eq(cantidad - 1);
    x.remove();
}

function eliminarElementoPrincipio() {
    var x;
    x = $("li");
    x = x.eq(0);
    x.remove();
}

function eliminarPrimeroSegundo() {
    var x;
    x = $("ul li:lt(2)");
    x.remove();
}

function eliminarDosUltimos() {
    var x;
    x = $("ul li").slice(-2);
    x.remove();
}
```

Lo primero, con la función `inicializarEventos()`, obtenemos la referencia a cada botón y le asignamos a su evento click cada una de las funciones declaradas.

Empecemos con la de `eliminarElementos()`. Para borrar todos los elementos contenidos en una lista, primero obtenemos la referencia de la lista mediante la función `$` y seguidamente llamamos al método **empty**:

```
function eliminarElementos()
{
    var x;
    x=$("#ul");
    x.empty();
}
```

La forma de restaurar la lista de nuevo será utilizando el método `html` e insertando directamente los ítems a partir del elemento `ul`:

```
function restaurarLista()
{
    $("#ul").html('<li>Primer item.</li><li>Segundo item.</li><li>Tercer
item.</li><li>Cuarto item.</li>');
}
```

Con el método **append**, añadiremos un elemento al final de la colección de elementos del objeto jQuery:

```
function anadirElementoFinal()
{
    var x;
    x=$("#ul");
    x.append("<li>otro item al final</li>");
}
```

Mediante el método `prepend`, añadiremos un elemento al principio de la lista:

```
function anadirElementoPrincipio()
{
    var x;
    x=$("#ul");
    x.prepend("<li>otro item al principio</li>");
}
```

Mediante la función `eliminarElementoFinal()`, por medio de la propiedad `length` obtendremos la cantidad de elementos que almacena el objeto jQuery y luego mediante el método **eq** (**equal**) le indicaremos la posición del elemento que necesitamos (debéis saber que este método devuelve otro objeto de tipo jQuery) y por último, para borrarlo llamamos al método **remove**:

```
function eliminarElementoFinal()
{
    var x;
    x=$("#li");
```

```

var cantidad=x.length;
x=x.eq(cantidad-1);
x.remove();
}

```

La función `eliminarElementoPrincipio()` funciona de una forma similar a borrar el último, pero no necesitamos saber cuantos elementos almacena el objeto jQuery, ya que debemos acceder al primero y eliminarlo mediante el método **remove**. Recordad que el primero se almacena en la posición cero:

```

function eliminarElementoPrincipio()
{
    var x;
    x=$("li");
    x=x.eq(0);
    x.remove();
}

```

La forma de eliminar el primero y segundo elemento que están almacenando el objeto jQuery será usando el método `lt` (less than), que nos devuelve todos los elementos menores a la posición que le pasamos como parámetro. En nuestro ejemplo nos está devolviendo los elementos de la posición 0 y 1:

```

function eliminarPrimeroSegundo() {
    var x;
    x = $("ul li:lt(2)");
    x.remove();
}

```

Para eliminar los dos elementos finales podemos hacerlo de varias formas. Nosotros hemos optado por usar el método `slice`, que retorna un subconjunto de elementos, que en este caso al indicarlo con el signo negativo, significa que empieza por el final de la lista, y que contendrá 2 elementos. Posteriormente llamamos al método `remove` para eliminar dichos elementos:

```

function eliminarDosUltimos() {
    var x;
    x = $("ul li").slice(-2);
    x.remove();
}

```




---

Podéis encontrar más información sobre el método `slice`, en la documentación oficial de jQuery:

<http://api.jquery.com/slice/>

---

### 4.9.5. Administración de eventos con jQuery

jQuery nos va a facilitar la administración de eventos de JavaScript y lo va a hacer transparente la diferencia en el registro de eventos entre los distintos navegadores actuales (IE Explorer, FireFox, Chrome).



Si recordáis de los puntos anteriores, hemos estado utilizado el manejador de eventos:

**`$(document).ready(nombre de función)`**

Comentamos anteriormente que esta función se ejecuta cuando el DOM del documento está en memoria. En el caso de que no utilicemos la librería jQuery esto se realiza a través del evento load.

Si recordáis, el evento click de un elemento, la sintaxis que utilizábamos era:

```
var x;
x = $("button");
x.click(presionBoton);
```

De esta manera registrábamos la función presionBoton() para todos los elementos de tipo button del documento.

Y con este otro código:

```
var x;
x = $("#boton1");
x.click(presionBoton);
```

Solo lo registrábamos para el elemento con id con valor "boton1".

Y ya por último, en el código siguiente, registramos para todos los elementos "button" que dependen del div con valor de id igual a "formulario1":

```
var x;
x = $("#formulario1 button");
x.click(presionBoton);
```

Una vez recordados estos conceptos, vamos a ver los eventos más importantes de jQuery

### 4.9.5.1. Eventos mouseover y mouseout

En este punto hablaremos de los eventos mouseover y mouseout, que son los equivalentes de JavaScript onmouseover y onmouseout. Lo normal es que estos eventos estén unidos. Debéis saber que:

- Mouseover se dispara cuando posicionamos la flecha del ratón sobre un elemento HTML
- Mouseout se dispara cuando sacamos la flecha del control.

Vamos a probar estos dos eventos; realizaremos una página (ejemplo09) que contenga tres hipervínculos y cuando se posicione la flecha del mouse sobre botón cambiaremos el color de texto del mismo, retornando el color original cuando retiramos la flecha del control.

Partiremos de la siguiente página que contiene tres enlaces. La llamaremos como en los anteriores casos *página1.html*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo09</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <a href="http://www.google.es">Google</a>
  <br>
  <a href="http://www.estudiosabiertos.com">Estudios Abiertos</a>
  <br>
  <a href="http://www.facebook.com">Facebook</a>
  <br>
</body>
</html>
```

Definimos las funciones necesarias en el archive funciones.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("a");
  x.mouseover(entraRaton);
  x.mouseout(saleRaton);
}

function entraRaton() {
  $(this).css("background-color", "#ff0");
}

function saleRaton() {
  $(this).css("background-color", "#fff");
}
```

Después de realizar la inicialización, como en los casos anteriores, asignaremos las funciones que se ejecutarán cuando se desencadenen los eventos `mouseover` y `mouseout`:

```
var x;  
x = $("a");  
x.mouseover(entraRaton);  
x.mouseout(saleRaton);
```

La función `entraRaton()` accede al elemento que recibe en su interior el puntero del ratón (la obtenemos mediante la referencia que guarda `this`) y cambiaremos el color de la propiedad `text-background` del CSS:

```
function entraRaton() {  
    $(this).css("background-color", "#ff0");  
}
```

La función `saleRaton()` volverá el formulario al color original:

```
function saleRaton() {  
    $(this).css("background-color", "#fff");  
}
```

Si vemos en ejecución nuestro ejemplo, cuando arranque la página veremos tres enlaces:

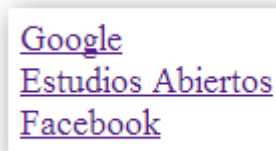


Figura 4.36. Página cuando carga.

En cuanto pasemos con el puntero del ratón por encima de cualquiera de ellos, veremos como cambian el color del fondo del mismo:



Figura 4.37. Vista cuando pasamos por encima de ellos con el ratón.

### 4.9.5.2. Evento hover

Con la librería jQuery además de mapear los eventos del DOM posee eventos propios que combinan estos. En el punto anterior hemos visto, que lo mas común es combinar los eventos `mouseover` y `mouseout`, por lo que podemos utilizar un evento de jQuery que combina estos. Es el evento **hover**, al que se le pasan dos parámetros, la función cuando se posiciona el ratón sobre el elemento en cuestión, y el otro, que es la función que se desencadenará cuando se quite el ratón. Su sintaxis es la siguiente:

```
$(elemento).hover([función ingreso ratón],[función salida ratón])
```

Vamos a aplicarlo en el mismo ejemplo anterior en el que ya trabajamos con ambos eventos (ahora será el ejemplo10). La página desde la que partiremos será igual que la anterior.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>ejemplo10</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <a href="http://www.google.es">Google</a>
  <br>
  <a href="http://www.estudiosabiertos.com">Estudios Abiertos</a>
  <br>
  <a href="http://www.facebook.com">Facebook</a>
  <br>
</body>
</html>
```

En cambio, el archivo `funciones.js` lo vamos a modificar e insertaremos el siguiente código:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("a");
  x.hover(entraMouse, saleMouse);
}

function entraMouse() {
  $(this).css("background-color", "#ff0");
}

function saleMouse() {
  $(this).css("background-color", "#fff");
}
```



Cuando inicialicemos los eventos, únicamente utilizaremos el evento hover para cambiar el color de fondo del enlace cuando se posicione el puntero del ratón y devolverla al color original cuando se quite:

```
function inicializarEventos() {
    var x;
    x = $("a");
    x.hover(entraMouse, saleMouse);
}
```

El funcionamiento de la página será igual que en el apartado anterior, por lo que no insertaremos las capturas de pantalla.

### 4.9.5.3. Evento mousemove

Este evento se desencadenará cuando se mueva el puntero del ratón dentro del elemento HTML respectivo. Nos servirá para recuperar la coordenada donde se encuentra en ese momento el puntero del ratón, pasando el objeto event como parámetro.

Veamos un ejemplo de cómo realizarlo y capturaremos el evento mousemove a nivel del objeto document. Mostraremos la coordenada en la que se encuentre el puntero del ratón. Para ello, necesitamos la siguiente página1.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
    <title>Ejemplo13</title>
    <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
    <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
    <p id="corx">
        coordenada x=</p>
    <p id="cory">
        coordenada y=</p>
</body>
</html>
```

Únicamente tendremos dos párrafos en los que iremos escribiendo las coordenadas donde se encuentra el ratón. Para ello, insertamos el siguiente código en el archivo funcione.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
    var x;
    x = $(document);
```

```
x.mousemove(moverMouse);  
}  
  
function moverMouse(event) {  
    var x;  
    x = $("#corx");  
    x.text("coordenada x=" + event.clientX);  
    x = $("#cory");  
    x.text("coordenada y=" + event.clientY);  
}
```

Como en todos los ejemplos, obtendremos la referencia del objeto document y le registraremos la función a ejecutar cuando se dispare el evento:

```
var x;  
x = $(document);  
x.mousemove(moverMouse);
```

Después declaramos la función que se va a ejecutar cada vez que se mueva el cursor del ratón en el documento:

```
function moverMouse(event) {  
    var x;  
    x = $("#corx");  
    x.text("coordenada x=" + event.clientX);  
    x = $("#cory");  
    x.text("coordenada y=" + event.clientY);  
}
```

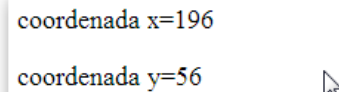
Aquí hemos creado un objeto jQuery a partir del id del primer párrafo y asignamos el texto del mismo insertando el valor del atributo event.clientX, que es el que tiene almacenado la coordenada x del cursor del ratón:

```
x = $("#corx");  
x.text("coordenada x=" + event.clientX);
```

Realizaremos la misma operación con la coordenada "y" y la pintaremos como texto del párrafo con id "cory":

```
x = $("#cory");  
x.text("coordenada y=" + event.clientY);
```

Si ejecutamos nuestra página en el navegador, veremos que según se va moviendo el ratón, se va cambiando:



```
coordenada x=196
coordenada y=56
```

Figura 4.38. Cambio de coordenadas al mover el ratón.

#### 4.9.5.4. Eventos mousedown y mouseup

Estos dos eventos normalmente van unidos y se usan en la misma pagina:

- Mousedown se disparará cuando presionemos alguno de los botones del ratón.
- Mouseup se disparará cuando dejemos de presionar el botón.

Veamos con un ejemplo como implementarlo (ejemplo12). Realizaremos una página que va a contener una tabla con una fila y dos columnas. Cuando presionemos el botón del ratón dentro de una celda, esta cambiará el color de fondo a amarillo y cuando levantemos el dedo del ratón volverá a blanco la celda. Todo esto lo vemos en la página1.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo12</title>
  <script src="../../Scripts/jquery-1.8.2.min.js"
type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
</head>
<body>
  <table style="padding: 1px; border: 1px solid #000000;">
    <tr>
      <td>1111111111</td>
      <td>1111111111</td>
    </tr>
  </table>
</body>
</html>
```

Definiremos todas las funciones en el archive funciones.js

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $('td');
  x.mousedown(presionaRaton);
  x.mouseup(sueltaRaton);
}
```

```
function presionaRaton() {
    $(this).css("background-color", "#ff0");
}

function sueltaRaton() {
    $(this).css("background-color", "#fff");
}
```

Lo primero que hacemos después de obtener la referencia al documento es asociar los eventos mousedown y mouseup a todos los elementos "td" del documento:

```
var x;
x = $('td');
x.mousedown(presionaRaton);
x.mouseup(sueltaRaton);
```

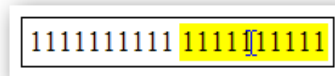
En el momento que presionemos el botón del ratón dentro de una celda de la tabla se disparará la función presionaRaton():

```
function presionaRaton() {
    $(this).css("background-color", "#ff0");
}
```

Cuando soltemos el botón del ratón, dejaremos el color blanco en la celda ejecutando la función sueltaRaton():

```
function sueltaRaton() {
    $(this).css("background-color", "#fff");
}
```

Si ejecutamos nuestra página en el navegador, veremos la siguiente tabla, y cuando presionemos con el ratón dentro de la celda de la tabla, veremos como cambia a color amarillo:



1111111111	1111111111
1111	1111

**Figura 4.39.** Vista cuando presionamos sobre una celda de la tabla.

### 4.9.5.5. Evento dblclick

En este capítulo vamos a ver el evento “dblclick” que se disparará cuando se presione dos veces seguidas el botón izquierdo del ratón.

Veamos como implementar este evento mediante un ejemplo. Para eso, vamos a implementar un div en una coordenada absoluta y lo ocultaremos al hacer doble click en su interior. Partimos de la página.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo13</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
  <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <div id="recuadro">
    <h1>
      Haz doble clic aqui para ocultar este recuadro</h1>
    </div>
</body>
</html>
```

Implementamos las funciones necesarias en el archivo funciones.js

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x;
  x = $("#recuadro");
  x.dblclick(dobleClic);
}

function dobleClic() {
  var x;
  x = $(this);
  x.hide()
}
```

Y además de los archivos anteriores, necesitaremos de una archivo de estilos.css donde insertaremos el siguiente código:

```
#recuadro {
  color:#aa0;
  background-color:#ff0;
  position:absolute;
  text-align:center;
  left:40px;
```

```

top:30px;
width:800px;
height:70px;
}

```

Pasemos a explicar las funciones declaradas en el archivo js. En la función inicializarEventos vamos a registrar el evento dblclick para el div:

```

var x;
x = $("#recuadro");
x.dblclick(dobleClic);

```

Después, cuando presionemos dos veces seguidas (gesto de doble click del ratón) dentro del div se disparará la función dobleClic():

```

function dobleClic() {
    var x;
    x = $(this);
    x.hide()
}

```

En la que obtendremos la referencia del elemento que emitió el evento y llamaremos al método hide del objeto jQuery creado, para que lo oculte. Si ejecutamos la página en el navegador, se nos mostrará un recuadro, en el que cuando se haga doble click, veremos como desaparece:



Figura 4.40. Recuadro en el que hacer doble click.

#### 4.9.5.6. Evento focus y blur

Estos eventos suelen ir unidos en las páginas webs.

- Evento **focus**: se produce cuando dicho control recibe el foco o se activa.
- Evento **blur**: es el evento contrario al evento focus y se dispara cuando un control pierde el foco.

Dichos eventos los podemos capturar en los controles de tipo:

- text.
- textarea.
- button.
- checkbox.

- fileupload.
- password.
- radio.
- reset.
- submit.

Veamos un ejemplo donde tendremos dos controles de tipo text con algún contenido. Ambos controles tendrán fijados de color azul su fuente. Al tomar foco el control cualquiera de los dos, cambiará a color rojo y al perder el foco volverá a color azul. Comenzaremos viendo el código html de la página1.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
  <title>Ejemplo14</title>
  <script src="Scripts/jquery-1.8.2.min.js" type="text/javascript"></script>
  <script src="Scripts/funciones1.js" type="text/javascript"></script>
  <link href="Styles/estilos.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <form action="#" method="post">
    <input type="text" name="text1" id="text1" value="Hola" size="20">
    <br>
    <input type="text" name="text2" id="text2" value="Hola" size="20">
  </form>
</body>
</html>
```

Después tendremos que declarar las funciones en el archivo funciones1.js:

```
var x;
x = $(document);
x.ready(inicializarEventos);

function inicializarEventos() {
  var x = $("#text1");
  x.focus(tomaFoco);
  x.blur(pierdeFoco);
  x = $("#text2");
  x.focus(tomaFoco);
  x.blur(pierdeFoco);
}

function tomaFoco() {
  var x = $(this);
  x.css("color", "#f00");
}

function pierdeFoco() {
  var x = $(this);
  x.css("color", "#00f");
}
```

Y por ultimo, definiremos los estilos en el archivo estilos.css

```
#text1, #text2 {  
    color: #00f;  
}
```

Veamos como asociar los eventos focus y blur a los controles de tipo text mediante la función inicializarEventos():

```
function inicializarEventos() {  
    var x = $("#text1");  
    x.focus(tomaFoco);  
    x.blur(pierdeFoco);  
    x = $("#text2");  
    x.focus(tomaFoco);  
    x.blur(pierdeFoco);  
}
```

Y a continuación declaramos la función tomaFoco(), que será la encargada de cambiar de color a rojo al texto del control seleccionado:

```
function tomaFoco() {  
    var x = $(this);  
    x.css("color", "f00");  
}
```

La función pierdeFoco(), será la que cambiará de nuevo a azul, cuando el control ya no tenga el foco:

```
function pierdeFoco() {  
    var x = $(this);  
    x.css("color", "#00f");  
}
```



## ● Resumen

---

- En esta unidad hemos realizado una introducción a Ajax, y hemos visto con su uso como podemos mejorar la interacción que se produce entre los usuarios con las aplicaciones web mediante AJAX, evitando que se recargue constantemente la página, ya que este intercambio de información con el servidor se va a realizar en un segundo plano.
- Hemos conocido la nomenclatura de JSON, necesaria para poder realizar correctamente nuestras aplicaciones en AJAX y analizar las peticiones al servidor.
- Hemos visto aspectos más avanzados de JavaScript, y hemos visto como tratar las excepciones y como funciona la reflexión.
- Hemos realizado un ejemplo de una aplicación Ajax, montando un pequeño servidor en nuestro propio equipo.
- Hemos aprendido a utilizar jQuery desde el principio, a insertarla en nuestras páginas mediante sus referencias y a utilizarla combinándola con JavaScript. Ahora ya sabemos como seleccionar un elemento mediante su Id, por su tipo de elemento y los selectores que se pueden usar para recoger los elementos de las páginas.
- Ya conocemos los métodos y eventos más importantes del objeto jQuery y los hemos implementado cada uno de ellos en un ejemplo totalmente funcional.