1. ¿Qué es jQuery?

jQuery es una biblioteca de JavaScript ligera, cuya filosofía es "escribir menos, hacer más". El propósito de jQuery es hacer que sea mucho más fácil de usar JavaScript en su sitio web.

jQuery posee muchas tareas que requieren gran cantidad de líneas de código JavaScript para lograrlo, y lo envuelve en los métodos que se pueden llamar con una sola línea de código.

jQuery también simplifica mucho las cosas complicadas de JavaScript, como las llamadas AJAX y Manipulación DOM.

La biblioteca jQuery contiene las siguientes características:

- Manipulación HTML / DOM
- Manipulación de CSS
- Métodos de evento HTML
- Efectos y animaciones
- AJAX
- Utilidades

Sugerencia: Además, jQuery tiene plugins para casi cualquier tarea que hay.

¿Por qué jQuery?

Hay muchos otros frameworks de JavaScript por ahí, pero jQuery es una de las más populares, y también el más extendido.

Muchas de las compañías más grandes en la Web utilizan jQuery, como:• Google • Microsoft • IBM • Netflix

¿Podrá jQuery trabajar en todos los navegadores?

El equipo de jQuery conoce los problemas entre los navegadores, y lo han tenido en cuenta en el desarrollo del mismo. jQuery funcionará exactamente igual en todos los principales navegadores, incluyendo Internet Explorer 6.

1.1 Instalación de jQuery

Adición de jQuery para sus páginas web

Para usar jQuery, es necesario descargar la biblioteca jQuery (explicado más adelante), e incluir en las páginas que deseen utilizarla.

La biblioteca jQuery es un solo archivo de JavaScript, y se referencia a él utilizando el siguiente código HTML

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

Observe que la etiqueta <script> debe estar dentro de la sección <head> de la página.

¿Se pregunta por qué no ponemos type = "text / javascript" dentro de la etiqueta <script>?

Esto no es necesario en HTML5. JavaScript es el lenguaje de scripts por defecto en HTML5 y en todos los navegadores modernos.

1.1.1. Descarga de jQuery

Al momento de esta escritura, existen 2 versiones de la librería para la descarga, la 1.10.2 y la 2.0.3. Ambas contienen el mismo API, la diferencia es que la versión 2 elimina el soporte para Internet Explorer 6, 7 y 8. Tú decides cual usar, si crees que tus usuarios todavía usan buscadores viejos como estos opta por la 1, si no, por la 2.

Dependiendo de cuál versión hayas escogido, ahora debes seleccionar si quieres la versión de **producción** o de **desarrollo**. La de desarrollo, normalmente, la utilizas mientras estas construyendo tu página web, luego cuando ya está estarás listo para subirla a tu servidor web, usas la de producción. La razón de esto es que la versión de producción esta comprimida, por lo que es más rápida de cargar por los usuarios cuando visitan tu página.

Consejo: Coloque el archivo descargado en el mismo directorio que las páginas en las que desea utilizar ella.

1.1.2. Alternativas a la descarga

Si no desea descargar y albergar jQuery usted mismo, usted puede incluir desde una red de entrega de contenidos (CDN, Content Delivery Network).

Tanto Google como Microsoft almacenan jQuery.

Para usar jQuery de Google, utilice uno de los siguientes:

Google CDN:

```
<head>
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</ script>
</ head>
```

Pruebe usted mismo »

Obtenga la última versión disponible con Google CDN:

Si nos fijamos en la URL de Google - la versión de jQuery se especifica en la URL(1.10.2). Si desea utilizar la última versión de jQuery, puede quitar un número desde el extremo de la cadena de versión (por ejemplo 1,10), entonces Google devolver la última versión disponible en la serie 1.10 (1.10.0, 1.10.1, etc), o puede llevarlo hasta el número entero (1), y Google devolverá la última versión disponible en la serie 1 (de 1.1.0 a 1.10.9).

Una gran ventaja de utilizar el jQuery alojado desde Google o Microsoft:

Muchos usuarios ya han descargado jQuery desde Google o Microsoft cuando visitar otro sitio. Como resultado, se puede cargar desde la memoria caché cuando visitan su sitio, que lleva al rápido tiempo de carga. Además, la

mayoría de CDN se asegurará de que, una vez un usuario solicita un archivo de la misma, se sirve desde el servidor más cercano a ellos, que también conduce a un menor tiempo de carga.

2. Sintaxis jQuery

La sintaxis de jQuery está hecha a medida para la **selección** de los elementos HTML y realizar alguna **acción** en el elemento o elementos.

Sintaxis básica es: **$ *(selector).acción* ()**

- Un signo $ para definir selector jQuery
- (*selector)* sera la "consulta (o encontrar)" en los elementos HTML
- Una *acción* de jQuery () para ser realizado en el elemento

Ejemplos:

$(this).hide() - oculta el elemento actual.

$("p").hide() - Oculta todos los elementos <p>.

$(".test").hide() - oculta todos los elementos con class = "test".

$("#test").hide() - oculta el elemento con id = "test".

¿Está familiarizado con los selectores CSS?

jQuery utiliza la sintaxis CSS para seleccionar elementos. Usted aprenderá más sobre el selector sintaxis en el siguiente capítulo de este tutorial.

El evento Ready del Documento

Todos los métodos de jQuery tienen que ir dentro del evento Ready del Documento, como vemos en el ejemplo:

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

Esto es para prevenir cualquier código jQuery que se ejecute antes de que el documento esté terminado de cargar.

Estos son algunos ejemplos de acciones que pueden fallar si los métodos se ejecutan antes que el documento está completamente cargado:

- Tratando de ocultar un elemento que no se crea todavía
- Tratando de obtener el tamaño de una imagen que no está cargado aún.
- etc…

Consejo: El equipo de jQuery también ha creado un método aún más corto para el documento listo:

```
$(function() {
/ / Métodos jQuery van aquí ...
});
```

Utilice la sintaxis que se prefiera. Creemos que el evento ready documento es más fácil de entender cuando la lectura del código.

3. jQuery selectores

Los selectores de jQuery son una de las partes más importantes de la biblioteca jQuery, estos permiten seleccionar y manipular los elementos de HTML.

Con los selectores de jQuery puede encontrar elementos en función de su id, clases, tipos, atributos, valores de los atributos y mucho más. Se basa en los ya existentes selectores CSS , y además, tiene algunos selectores propios.

Todo tipo de selectores de jQuery, comience con el signo de dólar y paréntesis: $().

Selector de etiqueta

El selector de etiqueta jQuery selecciona elementos en función de sus nombres de etiqueta. Puede seleccionar todas las <p> de una página de esta forma: $("p")

Ejemplo

Cuando un usuario hace clic en un botón, todos los elementos <p> estarán ocultas:

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

Selector de identificador

El selector de identificador jQuery utiliza el atributo id de una etiqueta HTML para encontrar el elemento específico.

Un identificador debe ser único dentro de una página, por lo que debe utilizar el selector de id # cuando desee encontrar un solo, único elemento.

Para encontrar un elemento con un identificador específico, escriba un carácter de almohadilla, seguida de la identificación del elemento:

```
$ ("#test")
```

Ejemplo

Cuando un usuario hace clic en un botón, el elemento con id = "test" se ocultará:

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

Selector de clase.

El selector de clase jQuery encuentra elementos con una clase específica.

Para encontrar los elementos con una clase específica, escriba un carácter de punto, seguido por el nombre de la clase: $ (".prueba")

Ejemplo

Cuando un usuario hace clic en un botón, los elementos con class = "test" se ocultarán:

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

Más ejemplos de jQuery selectores

$("*") Selecciona todos los elementos

$(this) Selecciona el elemento HTML actual

$("p.intro") Selecciona todos los elementos <p> con class = "intro"

$("p:first") Selecciona el primer elemento <p>

$("ul li:first") Selecciona el primer elemento <li> del primer

$("ul li:first-child") Selecciona el primer elemento de cada <li>

$("[href]") Selecciona todos los elementos con un atributo href

$("a [target = '_blank']") Selecciona todos los elementos <a> con un valor de atributo de destino igual a "_blank"

$("a[target! = '_blank']") Selecciona todos los elementos <a> con un valor de atributo de destino no es igual a "_blank"

$(":button") Selecciona todos los elementos <button> y <input> elementos de type = "button"

$("tr:even") Selecciona todos los elementos <tr> pares

$("tr:odd") Selecciona todos los elementos <tr> impares

Enlace a todos los selectores

:first-child	$("p:first-child")	Selecciona todos los elementos <p> que sean primeros hijos de su padre.
:first-of-type	$("p:first-of-type")	Selecciona todos los elementos <p> que sean el primer elemento <p> de su padre. (Primero del tipo)
:last-child	$("p:last-child")	Selecciona todos los elementos <p> que sean últimos hijos de su padre.
:last-of-type	$("p:last-of-type")	Selecciona todos los elementos <p> que sean el último elemento <p> de su padre.
:nth-child(*n*)	$("p:nth-child(2)")	Selecciona todos los elementos <p> que sean segundos hijos de su padre.
:nth-last-child(*n*)	$("p:nth-last-child(2)")	Selecciona todos los elementos <p> que sean penúltimos hijos de su padre.
:nth-of-type(*n*)	$("p:nth-of-type(2)")	Selecciona todos los elementos <p> que sean los segundos de los elementos <p> del padre.
:nth-last-of-type(*n*)	$("p:nth-last-of-type(2)")	Selecciona todos los elementos <p> que sean los penúltimos de los elementos <p> del padre.
:only-child	$("p:only-child")	Selecciona todos los elementos <p> que sean hijos únicos.
:only-of-type	$("p:only-of-type")	Selecciona todos los elementos <p>, que sean hijos únicos de su tipo.

parent > child	$("div > p")	Todos los elementos <p> que son hijos directos de un elemento
parent descendant	$("div p")	Todos los elementos <p> que sean descendientes de un elemento

element + next	$("div + p")	El elemento <p> siguiente al cada elemento
element ~ siblings	$("div ~ p")	Todos los elementos <p> que son hermanos siguientes de elemento

:eq(*index*)	$("ul li:eq(3)")	El cuarto element de una lista (empieza por 0)
:gt(*no*)	$("ul li:gt(3)")	Elementos de la lista cuyo índice sea mayor que 3.
:lt(*no*)	$("ul li:lt(3)")	Elemento de la lista con el índice menor que 3.
:not(*selector*)	$("input:not(:empty)")	Todos los elementos input que no estén vacíos.

:header	$(":header")	All header elements <h1>, <h2> ...
:animated	$(":animated")	All animated elements
:focus	$(":focus")	The element that currently has focus
:contains(*text*)	$(":contains('Hello')")	All elements which contains the text "Hello"
:has(*selector*)	$("div:has(p)")	All elements that have a <p> element
:empty	$(":empty")	All elements that are empty
:parent	$(":parent")	All elements that are a parent of

		another element
:hidden	$("p:hidden")	All hidden <p> elements
:visible	$("table:visible")	All visible tables
:root	$(":root")	The document's root element
:lang(*language*)	$("p:lang(de)")	All <p> elements with a lang attribute value starting with "de"
[*attribute*]	$("[href]")	All elements with a href attribute
[*attribute*=*value*]	$("[href='default.htm']")	All elements with a href attribute value equal to "default.htm"
[*attribute*!=*value*]	$("[href!='default.htm']")	All elements with a href attribute value not equal to "default.htm"
[*attribute*$=*value*]	$("[href$='.jpg']")	All elements with a href attribute value ending with ".jpg"
[*attribute*|=*value*]	$("[title|='Tomorrow']")	All elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[*attribute*^=*value*]	$("[title^='Tom']")	All elements with a title attribute value starting with "Tom"
[*attribute*~=*value*]	$("[title~='hello']")	All elements with a title attribute value containing the specific word "hello"
[*attribute**=*value*]	$("[title*='hello']")	All elements with a title attribute

		value containing the word "hello"
:input	$(":input")	All input elements
:text	$(":text")	All input elements with type="text"
:password	$(":password")	All input elements with type="password"
:radio	$(":radio")	All input elements with type="radio"
:checkbox	$(":checkbox")	All input elements with type="checkbox"
:submit	$(":submit")	All input elements with type="submit"
:reset	$(":reset")	All input elements with type="reset"
:button	$(":button")	All input elements with type="button"
:image	$(":image")	All input elements with type="image"
:file	$(":file")	All input elements with type="file"
:enabled	$(":enabled")	All enabled input elements
:disabled	$(":disabled")	All disabled input elements
:selected	$(":selected")	All selected input elements

:checked	$(":checked")	All checked input elements

Ejercicios

1. Crear un documento HTML con un encabezado, dos párrafos y un botón. Al pulsar el botón mediante el selector $("*") se debe ocultar todo. Método hide()

2. En el mismo documento HTML anterior. Al pulsar el botón debe ocultarse dicho botón.

3. En el mismo documento HTML anterior. Suponemos que el encabezado y el primer párrafo tienen el atributo class="intro".Deberás ocultar al clickar el botón, dicho párrafo.

4. Realiza el mismo código que en el script 3 pero ahora selecciona dicho párrafo mediante otro selector.

5. Selecciona el tercer elemento de la primera lista y oculta dicho elemento.

Lista 1:

- Tortilla
- Jamón
- Queso

Lista 2:

- Coca Cola
- Leche
- Té

6. Oculta los terceros elementos de la lista del script 5.

7. Crea un documento con un par de enlaces mediante el atributo href y oculta ambos enlaces.

8. Crea un documento con dos enlaces. Uno se tiene que abrir en una nueva ventana (target="_blank") y el otro no. Debes situar un botón para ocultar el enlace que se abre en una nueva ventana y otro botón para ocultar el que se abre en la misma ventana.

9. Crea un documento con un conjunto de párrafos y div. Haz un ejemplo que utilice:first-child, :first-of-type, :last-child, :last-of-type, :nth-child(n), :nth-last-child(n), :nth-of-type(n), :nth-last-of-type(n), :only-child, :only-of-type.

10. Crear un documento con una tabla. Poner el fondo de las filas pares en rojo (usar el método .css("background-color","red"). A continuación poner el fondo de las filas impares en verde .css("background-color","green")

4. Eventos jQuery

jQuery está hecho a medida para gestionar eventos de HTML / DOM.

Los controladores de eventos son métodos que se llaman cuando "algo pasa" en HTML.

Es común poner el código jQuery en los métodos de control de eventos en la sección <head>.

En el ejemplo siguiente, una función se llama cuando el evento Click para el botón es provocado:

Ejemplo

```
<! DOCTYPE html>
<html>
<head>
<script src="jquery.js"> </ script>
<script>
$(document).ready (function () {
        $("button").click (function () {
                $("p").hide();
```

```
      });
});
</ script>
</ head>
<body>
<h2> Este es un encabezado </h2>
<p> Este es un párrafo. </p>
<p> Este es otro párrafo. </p>
<button> Click me </button>
</ body>
</ html>
```

Algunos eventos de jQuery (http://prezi.com/xwylpmhtenar/eventos-jquery/)

Estos son algunos de los eventos más utilizados en jQuery:

Mouse Events	Keyboard Events	Form Events	Browser Events	Document loading
click	keypress	submit	error	load
dblclick	keydown	change	resize	unload
mouseenter	keyup	focus	scroll	ready
mouseleave		blur		
mouseup		select		
mousedown				
hover				

Eventos hover:

jQuery no solo posee los eventos del DOM sino que provee otros que combinan estos.

Es común utilizar los eventos mouseover y mouseout en común, por eso en jQuery existe un evento llamado hover que tiene dos parámetros:

$(elemento).hover([función de ingreso del mouse],[función de salida del mouse])

Es decir que al evento hover asociamos dos funciones, la primera se ejecuta cuando ingresamos la flecha del mouse dentro del elemento y la segunda cuando retiramos la flecha del mouse.

Ejemplo

Implementaremos una página que contenga tres hipervínculos, cuando ingrese la flecha del mouse al hipervínculo cambiaremos el color de fondo, retornando el color original cuando retiramos la flecha del elemento.

pagina1.html

```
<html>
<head>
<title>Problema</title>
<script type="text/javascript" src="../jquery.js"></script>
<script type="text/javascript" src="funciones.js"></script>
</head>
<body>
<a href="http://www.lavoz.com.ar">La Voz del Interior</a>
<br>
<a href="http://www.lanacion.com.ar">La nación</a>
<br>
<a href="http://www.clarin.com.ar">El clarín</a>
<br>
</body>
</html>
```

funciones.js

```
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");
}
```

Utilizamos solamente el evento hover para cambiar el color de fondo del ancla cuando se ingresa la flecha del mouse y retornarla al color original cuando se sale.

Método on

$(elements).on(events [, selector] [, data], handler);

Donde:

- **events** son los eventos que se buscan asociar al conjunto de elementos seleccionados. La novedad aquí es que pueden asociarse más de un evento separándolos con espacios. Por ejemplo: ‘click’, ‘dblclick’.
- **selector** especifica los descendientes de los elementos seleccionados que dispararán el evento. Se trata de un parámetro opcional.

- **data** indica cualquier tipo de datos que se necesite pasar al manejador cuando se dispara el evento. Es también un parámetro opcional y, por lo general, se corresponde con un objeto jQuery.
- **handler** se corresponde con el callback o acción a realizar después de que el evento se dispare.

Un ejemplo de uso completo puede ser el siguiente:

```
$('#myContainer .item').on({
  click: function() {
    event.preventDefault();
    console.log('item clicked');
  },
  mouseenter: function() {
    console.log('enter!');
  }
});
```

El resultado del código anterior es que hemos asociado dos eventos diferentes, con dos callbacks diferentes a aquellos elementos DOM que cumplan con el criterio de la selección (en este caso aquellos con la clase 'item' descendientes de un ID 'myContainer').

*Es más rápido utilizar el método on a utilizar click o cualquier otro evento, porque internamente jQuery hace una llamada a este método. Luego sería más directo, si lo utilizáramos nosotros mismos.

El método off()

La sintaxis de *off()* resulta similar a la de *on()*:

$(elements).off([events] [, selector] [, handler]);

Con *off()*, todos los parámetros son opcionales. Cuando se utiliza en su forma más simple, $(elements).off(), se eliminan todos los eventos asociados al conjunto seleccionado.

Objeto evento

Al definir un evento con jQuery, tenemos que escribir una función con el código a ejecutar cuando se produzca el evento, que podemos utilizar dentro de la función del evento y que contiene diversas utilidades que pueden ser esenciales a la hora de codificar el evento.

event.currentTarget
Devuelve el elemento sobre el que se ha lanzado el evento. Por ejemplo, si el evento es un onclick de un enlace, el currentTarget sería el enlace.

event.data
Devuelve los datos que pasan con el método on() para cada elemento seleccionado.

event.isDefaultPrevented()
Devuelve si se ha lanzado el método preventDefault() o no.

event.isImmediatePropagationStopped()
Devuelve si el método stopImmediatePropagation() se ha llamado, o no, en este objeto.

event.isPropagationStopped()
Devuelve si el método stopPropagation() ha sido llamado.

event.pageX
Devuelve la posición relativa del ratón en relación a la esquina izquierda del documento. Esta propiedad es muy útil cuando trabajamos con efectos.

event.pageY

Devuelve la posición relativa del ratón con respecto a la esquina superior del documento.

event.preventDefault()

Si llamamos a este método dentro de un evento, la acción predeterminada que se ejecutaría por este evento nunca será ejecutada.

event.stopImmediatePropagation()

Previene que se ejecuten otras acciones que pudieran estar asociadas al evento.

event.stopPropagation()

Previene que se ejecute cualquier evento que pudiera estar asociado a los padres del elemento dentro del árbol DOM.

event.target

Es el elemento DOM que inició el evento.

Ejemplo: event.target.nodeName – p, button, a….

event.type

Muestra el tipo de evento.

event.timeStamp

Número en milisegundos desde el 1 de enero de 1970, desde que el evento fue lanzado. Esto podría ayudarnos para realizar pruebas de rendimiento de nuestros scripts.

event.which

Para eventos de teclado y ratón, este atributo indica el botón o la tecla que ha sido pulsada.

Ejemplos

```
//muestra las coordenadas del ratón al hacer clic en cualquier sitio de la página
$(document).click (function(e){
        alert  ("X: " + e.pageX + " – Y: " + e.pageY)
```

```
});

//muestra la tecla pulsada  e inhabilitamos la acción, para que no lo escriba.
$("#mitexto").keypress(function (e) {
        e.preventDefault();
        alert (e.which + ": " + String.fromCharCode (e.which));
});
```

Para ver el resto de eventos y más ejemplos, pulse **aquí**.

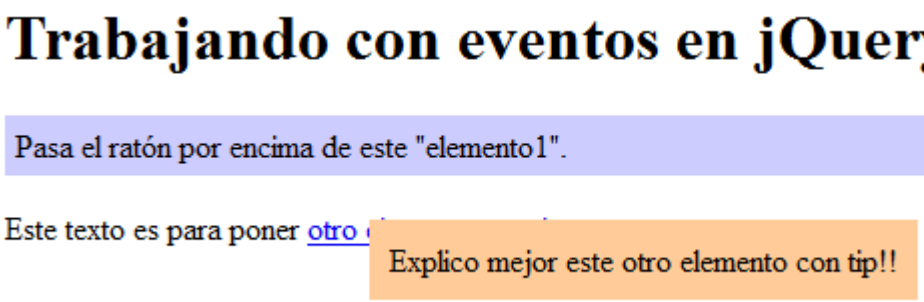
Ejercicios

11. Disponer un div de 800x70 píxeles. Al hacer doble clic redimensionarlo a 250x250 píxeles y si se hace doble clic nuevamente retornar al tamaño 800x70.

12. Crear una tabla con dos filas y dos columnas, cambiar el color del interior de la casilla cuando ingresamos con el mouse y regresarla al original cuando salimos.

13. Utiliza los métodos focus() y blur() para cambiar el color de dos cuadros de texto cuando posicionamos el foco y cuando lo retiramos.

14. Crear un tooltip (descripción emergente) básico con los eventos vistos, como el que vemos a continuación:



5. Efectos jQuery

5.1. Efectos jQuery - Ocultar y Mostrar

Hide, Show, Toggle, Slide, Fade, y Animate.

jQuery hide () y show ()

Con jQuery, puede ocultar y mostrar elementos HTML con el hide () y show ()

Ejemplo:

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

Sintaxis:

$*(selector)*.hide *(speed, callback);*

$*(selector)*.show *(speed, callback);*

El parámetro de velocidad es opcional y especifica la velocidad de la ocultación / muestra, y puede tomar los siguientes valores: "slow", "fast", o milisegundos.

El parámetro callback es opcional y es el nombre de una función a ejecutar después de completar la acción de ocultar (o mostrar).

El siguiente ejemplo muestra el parámetro de velocidad con hide ():

Ejemplo

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

jQuery toggle ()

Con jQuery, puede alternar entre los métodos hide() y show() con el método toggle(). Elementos mostrados se ocultan y elementos ocultos se muestran:

Ejemplo

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

Sintaxis:

$*(selector).toggle (speed, callback);*

El parámetro de velocidad opcional puede tomar los siguientes valores: "slow", "fast", o milisegundos.

El parámetro callback opcional es el nombre de una función a ejecutar después de completar la acción del método toggle()

Efectos jQuery - Fading

Con jQuery puede desaparecer y aparecer los elementos dentro y fuera de nuestra visibilidad.

jQuery tiene los siguientes métodos de fundido:

- fadeIn()
- fadeOut()
- fadeToggle()
- fadeTo()

Método fadeIn()

El método jQuery fadeIn() se utiliza para que aparezca elemento oculto.

Sintaxis:

$*(selector)*.fadeIn *(speed, callback);*

Método jQuery fadeOut()

El método jQuery fadeOut() se utiliza a desaparecer un elemento visible.

Sintaxis:

$*(selector)*.fadeOut *(speed, callback);*

Método fadeToggle()

El método jQuery fadeToggle() alterna entre la fadeIn() y fadeOut().

Sintaxis:

$*(selector).*fadeToggle *(speed, callback)*

Método fadeTo()

El método de jQuery fadeTo() permite la desaparición a una opacidad dada (valor entre 0 y 1).

Sintaxis:

$*(selector)*.fadeTo *(velocidad, opacidad, callback);*

Ejemplo: $("#div1").fadeTo("slow",0.15);

Efectos jQuery - Sliding

Con jQuery puede crear un efecto de deslizamiento en elementos para mostrar u ocultar.

jQuery tiene los siguientes métodos:

- slideDown()
- slideUp()
- slideToggle()

Método slideDown()

El método de jQuery slideDown () se utiliza para mostrar un elemento deslizándolo hacia abajo.

Sintaxis:

$*(selector)*.slideDown *(speed, callback).;*

Método slideUp ()

El método de jQuery slideUp () se utiliza para ocular un elemento deslizándolo hacia arriba.

Sintaxis:

$*(selector)*.slideUp *(speed, callback).;*

Método slideToggle ()

El método jQuery slideToggle() alterna entre la slideDown () y slideUp () métodos.

$*(selector)*.slideToggle *(speed, callback).;*

Efectos jQuery - Animate

El método jQuery animate() permite crear animaciones personalizadas.

Sintaxis:

$*(selector)*.animate *({params}, speed, callback).;*

El parámetro params requerido define las propiedades CSS para ser animados.

El siguiente ejemplo muestra un uso sencillo del método, que mueve un elemento hacia la izquierda, hasta que se alcanza una propiedad izquierda de 250px:

Ejemplo

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

Por defecto, todos los elementos HTML tienen una posición estática, y no se pueden mover.

Para manipular la posición, recuerde establecer primero la propiedad position del CSS relative, fixed o absolute!

jQuery animate () - Manipular múltiples propiedades

Tenga en cuenta que múltiples propiedades pueden ser animados a la vez:

Ejemplo

```
$("button").click (function () {
	$("div").animate ({
		left: '250px ',
		opacity: '0 .5 ',
		height: '150px ',
		width: '150px '
});
```

```
});
```

¿Es posible manipular todas las propiedades CSS con el método () animate?

Sí, casi! Sin embargo, hay una cosa importante para recordar: todos los nombres deben ser una sola palabra cuando se usa con lo el método animate: Usted necesitará escribir paddingLeft lugar de padding-left, marginRight lugar de margin-right, y así sucesivamente.
Además, la animación del color no está incluido en la biblioteca núcleo jQuery. Si desea animar el color, usted necesita descargar el plug-in de color Animaciones desde jQuery.com.

jQuery animate () - El uso de valores relativos

También es posible definir valores relativos (el valor es entonces con respecto al elemento de valor actual). Esto se hace poniendo += o -= delante del valor:

Ejemplo

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

jQuery animate () - Uso de valores predefinidos

Incluso puede especificar el valor de animación de una propiedad como "show", "hide" o "toogle":

Ejemplo

```
$("button").click (function () {
	$("div").animate ({
		heigth: 'toggle'
	});
```

});

jQuery animate () – Funcionalidad de cola

De forma predeterminada, jQuery viene con la funcionalidad de cola para las animaciones.

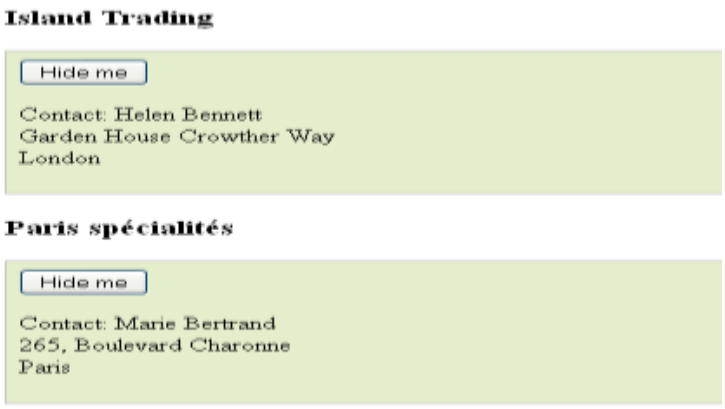
Esto significa que si usted escribe múltiples llamadas animate() unas después de la otra, jQuery crea una cola "interna" con estas llamadas a métodos. Por lo tanto, si usted desea llevar a cabo diferentes animaciones después de la otra, aprovechamos la funcionalidad de colas:

Ejemplo 1

```
$("button").click (function () {
        var div = $ ("div");
        div.animate ({height: '300px ', opacity: 0,4 '0'}, "slow");
        div.animate ({width: '300px ', opacity: 0,8 '0'}, "slow");
        div.animate ({height: '100px ', opacity: 0,4 '0'}, "slow");
        div.animate ({width: '100px ', opacity: 0,8 '0'}, "slow");
});
```

Ejercicios

15. Crea un documento con dos tag div, dentro de cada uno de los cuales debe haber un botón para esconder dichos tag.



16. Crea un documento con un botón que al pulsarlo, oculte un párrafo con velocidad rápida.

17. Crea un documento con un botón que al pulsarlo, agregue la opacidad de tres elementos div. El primero ejecútalo sin parámetros, el segundo de manera lenta y el tercero en tres milisegundos.

18. Crea un documento con un botón que al pulsarlo, nos quite la opacidad de tres elementos div. El primero ejecútalo sin parámetros, el segundo de manera lenta y el tercero en tres milisegundos.

19. Crea un documento con un botón que al pulsarlo, nos agregue o quite la opacidad de tres elementos div. El primero ejecútalo sin parámetros, el segundo de manera lenta y el tercero en tres milisegundos.

20. Crea un documento con un botón que al pulsarlo, agregue o quite la opacidad de tres elementos div. El primero ejecútalo en modo lento con una opacidad de 0,15, el segundo en modo lento con una opacidad de 0,4 y el tercero, en modo lento con una opacidad de 0,7.

21. Crea un documento con dos capas, tal que al pulsar sobre la primera se despliegue hacia abajo la segunda.

22. Crea un documento con dos capas, tal que al pulsar sobre la primera se despliegue hacia arriba la segunda.

23. Crea un documento con dos capas, tal que al pulsar sobre la primera fija se despliegue hacia arriba o abajo la segunda.

24. Crea un programa que al pulsar en un botón anime un cuadrado moviéndolo a la izquierda 375 px y lo haga más pequeño 150 px.

25. Crea un programa que al pulsar en un botón se muestre y se oculte la anchura de un cuadrado.

26. Crear un programa que al pulsar en un botón se mueva un cuadrado 100 px a la derecha y si dentro del cuadrado pone HELLO aumente el tamaño de la letra.

HACER Y ENTREGAR PRÁCTICA 1

5.2. jQuery Detener Animaciones

El método stop de jQuery () se utiliza para detener las animaciones o efectos antes de que esté terminado.

El método stop () funciona para todas las funciones de efectos jQuery, incluyendo deslizamiento, la decoloración y animaciones personalizadas.

Sintaxis:

$*(selector).stop (stopAll, gotoEnd);*

El parámetro stopAll es opcional y especifica si también la cola de la animación debe ser borrado o no. El valor predeterminado es falso, lo que significa que sólo la animación activa será la que se detenga, lo que permite que las animaciones en cola se realicen después.

El parámetro gotoEnd es opcional y especifica si se desea o no completar la animación actual inmediatamente. El valor predeterminado es falso.

Así que, por defecto, el método stop () para la animación actual, pero se siguen realizando el resto de animaciones.

El siguiente ejemplo muestra el método stop (), sin parámetros:

Ejemplo

```
$("#stop").click (function () {
        $("#panel").stop();
});
```

5.3. Callback de funciones jQuery

Con callback de jQuery podemos hacer una secuencia de llamadas a funciones o una pila de funciones que se ejecutarán una detrás de otra**.**

A menudo cuando hacemos aplicaciones enriquecidas del lado del cliente con jQuery nos vemos en la necesidad de encadenar varias llamadas a funciones, para que una se ejecute detrás de otra, creando un efecto más elaborado. En este artículo veremos lo sencillo que es realizar lo que en inglés se llama "callback", es decir una función que se ejecuta después de otra.

Apilar funciones, para que se ejecuten una detrás de otra, nos servirá para hacer muchas cosas. En nuestro día a día con jQuery iremos encontrando la utilidad, pero de momento para explicar un caso en el que nos resultará imprescindible, se me ocurre que deseemos hacer una secuencia de efectos y cambios dinámicos en un elemento.

Por ejemplo imaginemos que se desea ocultar una capa con un efecto de fundido (de opaco a transparente), luego moverla a otra posición y volverla a mostrar (ya en la nueva posición) con otro efecto de fundido (en este caso de transparente a opaco). En principio podríamos pensar en hacer un código como este:

```
$("#micapa").fadeOut(2000);
$("#micapa").css({top:300,left:200});
$("#micapa").fadeIn(2000);
```

En este caso estamos alterando las propiedades de una capa con id="micapa". Primero llamamos a fadeOut() para ocultarla con un fundido, que durará 2 segundos (véase el parámetro 2000, que son los milisegundos que durará el efecto). Luego alteramos la posición de la capa, cambiando sus atributos CSS. Para acabar la volvemos a mostrar con un fundido de otros 2000 milisegundos.

Si lanzamos la ejecución de estas sentencias, tal como aparece en el código, será como si se ejecutasen todas a la vez. Como los fadeOut y fadeIn tardarán 2 segundos en ejecutarse y los cambios de las propiedades CSS top y left son inmediatos, lo que ocurrirá será que primero veremos la capa moverse a la nueva posición y luego veremos los dos efectos de fundido.

Lo mejor para darse cuenta de este caso es **verlo en marcha**.

Cómo realizar una pila de ejecución de funciones

Ahora que ya hemos visto uno de los casos en los que necesitaríamos ejecutar funciones en una pila, una después de otra, esperando a que termine completamente la ejecución de cualquier efecto o acción antes de comenzar con la siguiente. Vamos a ver cómo hacerlo con jQuery.

Simplemente tenemos que saber que **todas las funciones o métodos de jQuery pueden recibir un parámetro adicional con el nombre de la función que se tiene que ejecutar después que termine el procesamiento de la primera**. Esa segunda función que se ejecuta después de la primera es la que se conoce en inglés por callback. Un ejemplo sencillo para entenderlo.

```
miFuncion ("parametros de la función", funcionCallback);
```

En ese esquema de llamada a miFuncion(), se le están pasando dos parámetros. El primero sería un supuesto parámetro que necesitase miFuncion() y el segundo, que es el que nos interesa en este caso, el nombre de la función que se tiene que ejecutar después que acabe. Con este código, primero se ejecuta miFuncion() y cuando acaba completamente, se ejecuta funcionCallback(). Pero atención que este ejemplo lo hemos simplificado para que se pueda entender fácilmente y esta sintaxis sólo valdrá si funcionCallback **no recibe parámetros**, **porque no los podemos indicar con el nombre de la función**. Veamos entonces una forma de hacer este callback que funcione siempre:

```
miFuncion ("parametros de la funcion", function(){
  funcionCallback();
});
```

Con este código, que funcionaría igual que el anterior, lo bueno es que sí podemos indicar los parámetros que se necesiten para la llamada a funcionCallback().

Ejemplo real de callback con jQuery

Ahora que hemos aprendido toda la teoría, veamos un ejemplo práctico que solucionaría el problema comentado anteriormente sobre el procesamiento de diversos efectos y cambios en las propiedades de los objetos, para que se hagan siempre en la secuencia que deseamos. Se trata simplemente de aplicar las llamadas con callback que hemos antes.

```
$("#micapa").fadeOut(1000, function(){
  $("#micapa").css({'top': 300, 'left':200});
  $("#micapa").fadeIn(1000);
});
```

Como se puede ver, en la llamada a fadeOut() estamos pasando como parámetros el valor 1000, que son los milisegundos tiene que durar el efecto fade out (fundido hacia transparente), y luego la función callback, que se ejecutará después de que fadeOut() haya terminado.

Como el método css() (se encuentra como primera instrucción de la función callback) es instantáneo, no necesita hacerse un callback para ejecutar el fadeIn(), sino que se puede escribir directamente en la siguiente línea de código. Así pues, se ve que el callback, al menos en este ejemplo, sólo es necesario hacerlo cuando se ejecutan funciones que realizarán un procesamiento prolongado.

Podemos **ver este ejemplo de callback en una página aparte.**

5.4. jQuery método de encadenamiento

Con jQuery, puede encadenar acciones / métodos. Esto nos va a permitir ejecutar múltiples métodos de jQuery (en el mismo elemento) dentro de una declaración individual.

De esta manera, los navegadores no tienen que encontrar el mismo elemento (s) más de una vez.

Para encadenar una acción, sólo tiene que anexar la acción a la acción anterior.

El elemento de "p1" primero cambia a rojo, luego se desliza hacia arriba, y luego se desliza hacia abajo:

Ejemplo

```
$("#p1").css ("color", "red").slideUp (2000).slideDown (2000);
```

También podríamos haber añadido más llamadas de método, si es necesario.

Consejo: Cuando el encadenamiento, la línea de código podría llegar a ser bastante larga. Sin embargo, no es jQuery muy estrictos en la sintaxis; puede darle formato como usted desee, incluyendo los saltos de línea y tabuladores. Esto también funciona:

Ejemplo

```
$("#p1").css ("color", "red")
.slideUp(2000)
.slideDown(2000);
```

27. Mejora el ejercicio 26 añadiendo cinco botones.
    1. Empezar: Comienza la animación
    2. Parar: Para la animación actual pero continúa con el resto.
    3. Parar todo: Detiene todas las animaciones de inmediato.
    4. Para pero terminar: Detiene la animación activa, pero acabándola primero y luego continua el resto.
    5. Parar todo, pero terminar: Detiene las animaciones pero acabando primero la que se encuentra en ejecución.

28. Crea una aplicación que contenga una caja negra y dos enlaces. Un enlace será expandir y otro Contraer. Cuando se pulse uno de los dos botones, debe preguntar al usuario que cantidad lo quiere expandir o contraer. Una vez realizado la acción, debe cambiar de color.

6. HTML y jQuery

6.1. jQuery - Obtener contenido y atributos

jQuery contiene métodos poderosos para cambiar y manipular los elementos HTML y atributos.

jQuery – Manipulación DOM

Una parte muy importante de jQuery, es la posibilidad de manipular el DOM. jQuery viene con un montón de métodos DOM relacionados, que hace que sea fácil de acceder y manipular elementos y atributos.

Recibe contenido - text(), html(), y val()

Tres simples, pero útiles, métodos de jQuery para la manipulación de DOM son:

- text() - Establece o devuelve el contenido de texto de los elementos seleccionados.
- html() - Establece o devuelve el contenido de elementos seleccionados, incluyendo etiquetas HTML.
- val() - Establece o devuelve el valor de los campos del formulario.

El siguiente ejemplo muestra cómo obtener el contenido con los métodos text y html de jQuery.

Ejemplo

```
$("#btn1").click (function () {
        alert ("Texto:" + $("#test").text());
});
$("#btn2").click (function () {
        alert ("HTML" + $("#test").html ());
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_html_get

El siguiente ejemplo muestra cómo obtener el valor de un campo de entrada con el método val() de jQuery.

Ejemplo

```
$ ("#btn1").click (function () {
        alert ("Valor:" + $("#test").val());
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_val_get

Obtenga Atributos - attr ()

El método attr() de jQuery se utiliza para obtener valores de los atributos.

El siguiente ejemplo muestra cómo obtener el valor del atributo href en un enlace:

Ejemplo

```
$("button").click (function () {
        alert ($ ("#w3s").attr("href").);
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_attr_get

6.2. jQuery - Establecer Contenido y Atributos

Establecer contenido - text(), html(), y val()

Vamos a utilizar los mismos tres métodos de la página anterior para **establecer el contenido:**

- text() - Establece o devuelve el contenido de texto de los elementos seleccionados
- html() - Establece o devuelve el contenido de elementos seleccionados (incluyendo HTML)
- val() - Establece o devuelve el valor de los campos del formulario

Ejemplo

```
$("#btn1").click (function () {
        $("#test1").text("Hola mundo");
```

```
});

$("#btn2").click (function () {
      $("#test2").html("<b> Hola mundo </ b>");
});

$("#btn3").click (function () {
      $("#test3).val("Pato Dolly").;
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_html_set

Una llamada de retorno (callback) para el text(), html () y val ()

Los tres métodos de jQuery anteriores: text(), html(), y val (), también vienen con una función de devolución de llamada. La función de devolución de llamada tiene dos parámetros: el índice del elemento de la lista de elementos seleccionados y el valor original. A continuación, devuelve la cadena que desea utilizar como el nuevo valor de la función.

El siguiente ejemplo muestra text() y html() con una función de devolución de llamada:

Ejemplo

```
$("#btn1").click (function () {
      $("#test1").text (function (i, origText) {
            return "Texto viejo:" + origText + "Nuevo texto: Hola mundo”
            (index: "+ i +") ";
      });
});

$("#btn2").click (function () {
      $("#test2").HTML (function (i, origText) {
            return "Viejo html:" + origText + "Nuevo html: Hola <b> mundo
            </ b>
```

```
        (index: "+ i +") ";
        });
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_html_callback

Atributos Set - attr ()

El método attr() también se utiliza para establecer / cambiar valores de atributos.

El siguiente ejemplo muestra cómo cambiar (set) el valor del atributo href en un enlace:

Ejemplo

```
$("button").click (function () {
        $("#w3s").attr("href", "http://www.w3schools.com/jquery");
});
```

El método attr () también le permite establecer múltiples atributos al mismo tiempo.

El siguiente ejemplo muestra cómo establecer tanto el href y atributos title en el mismo tiempo:

Ejemplo

```
$("button").click (function () {
        $("#w3s").attr ({
                "href": "http://www.w3schools.com/jquery",
                "title": "W3Schools jQuery Tutorial"
        });
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_attr_set2

Una llamada de retorno para attr ()

El método attr(), también viene con una función de devolución de llamada. La función de devolución de llamada tiene dos parámetros: el índice del elemento actual en la lista de elementos seleccionados y el valor del atributo

originales. A continuación, regresa la cadena que desea utilizar como nuevo valor de la función.

El siguiente ejemplo muestra attr () con una función de devolución de llamada:

Ejemplo

```
$("button").click (function () {
        $("#w3s").attr("href", function (i, origValue) {
                return origValue + "/ jquery";
        });
});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_attr_callback

6.3. jQuery – Añadir elementos

Con jQuery, es fácil añadir nuevos elementos / contenido.

Añadir nuevo contenido HTML

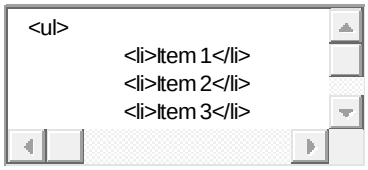
Veremos cuatro métodos de jQuery que se utiliza para añadir nuevos contenidos:

- append() - Inserta contenido al final de los elementos seleccionados.
- prepend() – Inserta contenido al comienzo de los elementos seleccionados.
- after() – Inserta contenido después de los elementos seleccionados
- before() - Inserta contenido antes de los elementos seleccionados

Método jQuery append()

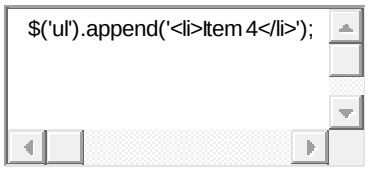
Con la función **append()** podremos añadir contenido al último elemento hijo del elemento seleccionado.

Supongamos que disponemos de una lista desordenada <ul> que contiene 3 elementos.



```
1 <ul>
2     <li>Item 1</li>
3     <li>Item 2</li>
4     <li>Item 3</li>
5 </ul>
```

A través de la función **append()** podremos añadir un nuevo elemento a la lista y situarlo en la posición 4. Veamos cómo.

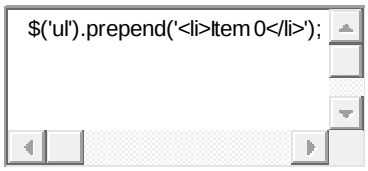



```
1 $('ul').append('<li>Item 4</li>');
```

Método jQuery prepend()

El funcionamiento de **prepend()** es el mismo que el de *append()*, con la diferencia de que el contenido se añade justo después de la etiqueta seleccionada.

Supongamos una lista como la descrita en la función anterior. Si queremos añadir un ítem nuevo al principio de la lista haríamos lo siguiente:




```
1 $('ul').prepend('<li>Item 0</li>');
```

Añadir varios nuevos elementos con append() y prepend()

En los dos ejemplos anteriores, sólo hemos introducido un texto / HTML al principio / final de los elementos HTML seleccionados.

Sin embargo, tanto el append() y prepend() pueden tomar un número infinito de nuevo elementos como parámetros. Los nuevos elementos se pueden generar con el texto / HTML (como nosotros lo hemos hecho en los ejemplos anteriores), con jQuery, o con código JavaScript y elementos DOM).

En el siguiente ejemplo, creamos varios elementos nuevos. Los elementos se crean con el texto / HTML, jQuery y JavaScript / DOM. Luego añadimos los nuevos elementos al texto con el método append()

Crear elemento con HTML

$("p").append (" Texto ");

Crear elemento con jQuery

$("p").append($("h2"));

Crear elemento con el DOM

var txt = document.createElement ("p");

txt.innerHTML = "Texto";

$("p").append(txt);

$("p").append(document.createTextNode("Hello"));

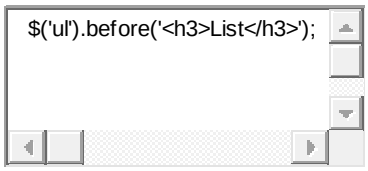
También se pueden guarder en variables y añadir todas ellas con una única sentencia.

$("p").append(txt1,txt2,txt3);

Métodos after() y before() de jQuery

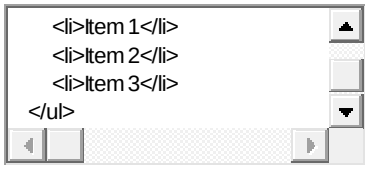
Con la función **before()** podremos añadir código antes del elemento seleccionado (fuera del mismo). Veamos cómo:

Si tomamos la lista de 3 elementos que presentamos desde un principio como ejemplo, de referencia y aplicamos la siguiente regla de jQuery:



1 $('ul').before('<h3>List</h3>');

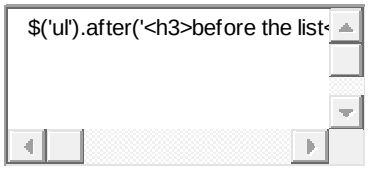
El resultado sería el siguiente:



```
1 <h3>before the list</h3>
2 <ul>
3     <li>Item 1</li>
4     <li>Item 2</li>
5     <li>Item 3</li>
6 </ul>
```

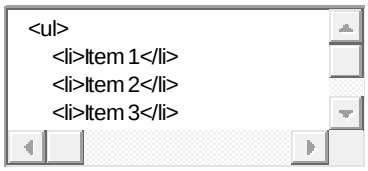
Por ultimo y contrario la función **after()** funciona exactamente igual que la función *before()* con la diferencia de que el argumento de la función se añade después del elemento seleccionado (fuera del mismo).

De esta forma y siguiendo el mismo procedimiento que en el ejemplo anterior. Aplicamos la siguiente regla a la lista:




```
1 $('ul').after('<h3>before the list</h3>');
```

Y el resultado que obtendríamos es el siguiente:




```
1 <ul>
2     <li>Item 1</li>
3     <li>Item 2</li>
4     <li>Item 3</li>
5 </ul>
6 <h3>after the list</h3>
```

Añadir varios nuevos elementos con after() y before()

Además, tanto el after() y before() pueden tomar un número infinito de nuevos elementos como parámetros. Los nuevos elementos se pueden generar con el texto / HTML (como nosotros lo hemos hecho en el ejemplo anterior), con jQuery, o con código JavaScript y elementos DOM.

En el siguiente ejemplo, creamos varios elementos nuevos. Los elementos se crean con el texto / HTML, jQuery y JavaScript / DOM. A continuación insertamos los nuevos elementos al texto con el método after():

Ejemplo

```
funcionar afterText ()
{
var txt1 = "<b> I </ b>"; // Crear elemento con HTML
var txt2 = $("<i></ i>").text("love"); / / Crear con jQuery
var txt3 = document.createElement ("jQuery"); / / Se crea con el DOM
txt3.innerHTML = "jQuery";
$("img").after (txt1, txt2, txt3); // Insertar nuevos elementos después img
}
```

Resultado

I *love* jQuery!

6.4. jQuery – Quitar elementos

Con jQuery, es fácil de quitar elementos HTML existentes.

Para eliminar elementos y contenido, hay principalmente dos métodos de jQuery:

- remove() - Elimina el elemento seleccionado (y sus elementos secundarios)
- empty() - Elimina los elementos secundarios del elemento seleccionado

Ejemplo

```
$("#div1").remove();
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_remove

$("#div1").empty();

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_dom_empty

Filtra los elementos que deberán ser eliminados

El método remove() también acepta un parámetro, que permite filtrar los elementos a eliminar.

El parámetro puede ser cualquiera de las sintaxis de selectores de jQuery.

El ejemplo siguiente elimina todos los elementos <p> con class = "cursiva":

Ejemplo

$("p").remove(".italic");

Método removeAttr()

El método removeAttr () elimina uno o más atributos de los elementos seleccionados.

Sintaxis

$(selector).removeAttr(attribute)

Ejemplo:

Quite el atributo de estilo de todos los elementos <p>:

```
$("button").click(function(){
  $("p").removeAttr("style");
});
```

EJERCICIOS

29. Crear un programa con un botón, tal que al hacer pasar sobre él, le establezca un fundido de 0,4 en modo lento, ponga las letras de color rojo, lo oculte con un desplazamiento hacia arriba y lo visualice con un desplazamiento hacia abajo.

30. Crear un documento con cuatro botones, un cuadro de texto, un enlace a http://www.thehobbit.com/ y un párrafo *"Tres anillos para los Reyes Elfos bajo el cielo.Siete para los **Señores Enanos** en casas de piedra. Nueve para los Hombres Mortales condenados a morir.Uno para el Señor Oscuro, sobre el*

*trono oscuro. Un Anillo para gobernarlos a todos. Un anillo para encontrarlos, un **Anillo** para atraerlos a todos y atarlos en las tinieblas en la Tierra de Mordor donde se extienden las Sombras."*

tal que,

- al pulsar en el primer botón se obtenga una alerta con el contenido de texto del párrafo indicado.
- Al pulsar en el segundo botón se debe obtener una alerta con el contenido incluyendo las etiquetas HTML del párrafo.
- Al pulsar en el tercer botón, se debe obtener una alerta con el valor del cuadro de texto.
- Al pulsar en el cuarto botón, se debe obtener una alerta con el contenido del enlace.

31. Modifica el ejercicio anterior, creando tres eventos asociados a tres botones, "SetText", "SetHTML" y "SetValue" para establecer el texto del párrafo a otro párrafo, ponerlo en cursiva y establecer un valor distinto al cuadro de texto.

32. Modifica el ejercicio anterior, de forma que muestre el texto antiguo y el nuevo una vez modificado en cada uno de los cambios realizados.

33. Crea un enlace a google y con un evento asociado a un botón, cambia su título, su enlace y su color.

34. Cambia un enlace a http://google.es mediante un evento asociado a un botón y un callback, en el que el nuevo texto introducido sea /intl/es/earth/index.html

35. Haz una aplicación con cuatro botones, cada uno asociado a un evento (prepend, append, after y before) y un div. Cada evento introduce un párrafo en el sitio correspondiente.

36. Haz lo mismo que el ejercicio anterior pero con una lista desordenada y cambiar el atributo tipo.

37. Crear un documento con un párrafo, mediante un evento asociado a un botón debemos añadirle (indiferente principio o final) un párrafo creado con HTML, un párrafo creado con jQuery y un párrafo creado con el DOM de JavaScript.

38. Crea una capa cuadrada (100 x 300 en amarillo) que contenga un par de párrafos. Borra dicha capa con el método remove() como gestor de un evento asociado a un botón.

39. Crea una capa cuadrada (100 x 300 en amarillo) que contenga un par de párrafos. Borra dichos párrafos con el método empty() como gestor de un evento asociado a un botón.

40. Tenemos una lista desordenada de elementos. Estos elementos algunos son anclas y otros son enlaces. Borrar únicamente aquellos que son enlaces mediante un evento asociado a un botón.

7. Validación de formularios

Ejemplos de validación con jQuery

Validar si hay algo escrito

```
if($(".required").val() == ""
```

```
<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
<script src="jquery.js"></script>

<script>
	$(document).ready(docLoaded);
	function docLoaded() {
		$(".required").blur(validarSiHayAlgoEscrito);
	}

	function validarSiHayAlgoEscrito() {
		if($(".required").val() == "")
			alert("Debe introducir algún valor");
	}
</script>

<form name="formulario">
	<input class="required" type="text" >
</form>
```

Validar si hay algo escrito II (Con trim)

```
if($.trim($(".required").val()) == "")
```

```
<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
<script src="jquery.js"></script>
```

```
<script>
	$(document).ready(docLoaded);
	function docLoaded() {
		$(".required").blur(validarSiHayAlgoEscrito);
	}

	function validarSiHayAlgoEscrito() {
		if($.trim($(".required").val()) == "")
			alert("Debe introducir algún valor");
	}
</script>

<form name="formulario">
	<input class="required" type="text" >
</form>
```

Validar si es un número

```
if(isNaN($(".required").val()))
```

```
<meta name="tipo_contenido"  content="text/html;" http-equiv="content-type" charset="utf-8">
<script src="jquery.js"></script>

<script>
	$(document).ready(docLoaded);
	function docLoaded() {
		$(".required").blur(validarSiHayAlgoEscrito);
	}

	function validarSiHayAlgoEscrito() {
		if(isNaN($(".required").val()))
			alert("Debe introducir un número");
	}
</script>

<input type="text" class="required">
```

Validar un carácter

Validar que hemos escrito un único carácter y que es una letra o un número.

/^ y $/ delimitan el principio y el final de la cadena.

\w representa un único carácter, que puede ser una letra o un número.

if (/^\w$/.test(valor))

Validar un conjunto de caracteres

Validar que hemos escrito una cadena de caracteres que pueden ser letras o números.

\w+ representa una cadena de caracteres, que pueden ser una letra o un número.

if(/^\w+$/.test(valor))

Validar que hemos escrito una cadena de caracteres que pueden ser letras o números seguida de un punto o un guión

[\.-] representa un punto o un guión.

if (/^\w+[\.-]$/.test(valor)){

Validar que hemos escrito una cadena de caracteres que pueden ser letras o números seguida de un punto o un guión, seguidos de más letras o números.

if (/^\w+[\.-]\w+$/.test(valor)){

Validar que hemos escrito una cadena de caracteres que pueden ser letras o números. Luego puede haber escrita una sucesión de punto o guión seguidos de más letras o números, tantas veces como sea necesario

if (/^\w+([\.-]\w+)*$/.test(valor)){

Validar que se cumplen todas las condiciones anteriores y que después hay escrita una Arroba.

if (/^\w+([\.-]\w+)*@$/.test(valor)){

Ejercicio

Construir la validación para el dominio, teniendo en cuenta que deberá empezar por letras o números y luego puede tener un punto o un guion y más

letras, luego tiene una arroba y luego texto seguido de una o varias sucesiones de un punto o guion y Texto.

Validar fechas I

Guardamos cada parte que compone la fecha en una variable diferente.

```
function validarFecha() {

var cadena = $("#fecha").val();

var dia = cadena.substring(0, cadena.indexOf("/"));

var mes = cadena.substring(cadena.indexOf("/") + 1,
cadena.lastIndexOf("/"));

var anio = cadena.substring(cadena.lastIndexOf("/") + 1, cadena.length);

}
```

Comprobamos que los días, meses y años tomen valores posibles.

```
function validarFecha() {

var cadena = $(".required").val();

var dia = cadena.substring(0, cadena.indexOf("/"));

var mes = cadena.substring(cadena.indexOf("/") + 1,

cadena.lastIndexOf("/"));

var anio = cadena.substring(cadena.lastIndexOf("/") + 1,

cadena.length);

if(isNaN(anio) || anio.length != 4 || parseFloat(anio) < 1900)

alert("valor del año incorrecto");

if(isNaN(mes) || parseFloat(mes) < 1 || parseFloat(mes) > 12)

alert("valor del mes incorrecto");

if(isNaN(dia) || parseFloat(dia) < 1 || parseFloat(dia) > 31)

alert("valor del día incorrecto");

}
```

Ejercicio. A los condicionales que ya teníamos añadir otro que evalúe si los meses 4, 6, 8 y 11 no tienen más de 30 días y otro que evalúe que el mes 2 no tenga más de 29 días.

Validar un formulario entero

Cuando el cuadro de texto pierda el foco, llamaremos a las siguientes funciones, según el caso:

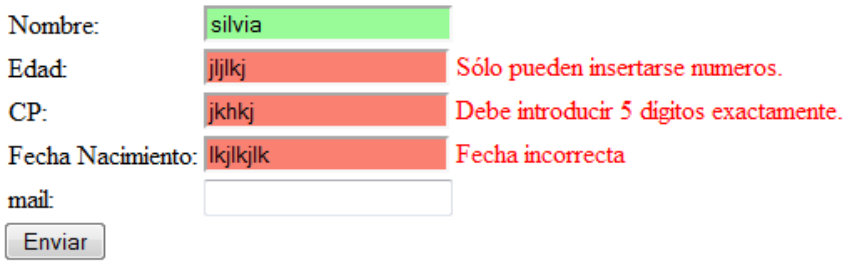
contieneValor();

soloNumeros();

cincoNumeros();

validarFecha();

validarMail();



Ver ejercicio resuelto

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

<style>
	span {
		color: red
	}
</style>

<script type="text/javascript">
	$(document).ready(docLoaded);
	function validacionFinal() {
		var formularioValido = true;
		var objetosValidacion = $(".required");
		objetosValidacion.push($(".numero"));
```

```
	objetosValidacion.push($(".fiveDigits"));
	objetosValidacion.push($(".fecha"));
	objetosValidacion.push($(".mail"));
	objetosValidacion.each(function(i) {
		if($(this).attr("valido") != "true") {
			formularioValido = false;
			return false;
		}
	});
	if(formularioValido == false) {
		alert("Revise el formulario, hay datos incorrectos");

	} else {
		alert("Eviando formulario...")
	}
}

function docLoaded() {
	$(".required").blur(contieneValor);
	$(".numero").blur(soloNumeros);
	$(".fiveDigits").blur(fiveDigits);
	$(".fecha").blur(validarFecha);
	$(".mail").blur(validarMail);
	$("#enviar").click(validacionFinal);
}

function soloNumeros() {
```

```
		if(isNaN($(this).val()) || $.trim($(this).val()) == "") {
			pintarError($(this), "Sólo pueden insertarse numeros.");
		} else {
			limpiar($(this));
		}
	}

	function fiveDigits() {
		if($(this).val().length != 5 || $.trim($(this).val()) == "" ||
isNaN($(this).val())) {
			pintarError($(this), "Debe introducir 5 dígitos
exactamente.");
		} else {
			limpiar($(this));

		}
	}

	function validarFecha() {
		var cadena = $(this).val();
		var dia = cadena.substring(0, cadena.indexOf("/"));
		var mes = cadena.substring(cadena.indexOf("/") + 1,
cadena.lastIndexOf("/"));
		var anio = cadena.substring(cadena.lastIndexOf("/") + 1,
cadena.length);
		var fechaCorrecta = true;

		if(isNaN(anio) || anio.length != 4 || parseFloat(anio) < 1900) {
			fechaCorrecta = false;
```

```
            }
            if(isNaN(mes) || parseFloat(mes) < 1 || parseFloat(mes) > 12) {
                  fechaCorrecta = false;
            }
            if(isNaN(dia) || parseFloat(dia) < 1 || parseFloat(dia) > 31) {
                  fechaCorrecta = false;
            }

            /*Los meses 4, 6, 9 y 11 no tienen más de 30 días. Además, el
mes 2 no tendrá más de 29 días*/
            if((mes == 4 || mes == 6 || mes == 9 || mes == 11) && dia >
30)
                  fechaCorrecta = false;
            if(mes == 2 && dia > 28)
                  fechaCorrecta = false;

            if(fechaCorrecta == false) {
                  pintarError($(this), "Fecha incorrecta");
            } else {
                  limpiar($(this));

            }
      }

      function validarMail() {
            if(/^\w+([\.-]\w+)*@\w+([\.-]\w+)*$/.test($(this).val()) ==
false) {
                  pintarError("La dirección de email es incorrecta.");
```

```
            } else {
                limpiar($(this));
            }
        }

        function contieneValor() {
            if($(this).val() == "") {
                pintarError($(this), "El campo debe contener un valor.");

            } else {
                limpiar($(this));
            }
        }

        function pintarError(elemento, msg) {
            elemento.css("background", "salmon");
            elemento.next().html(msg);
            elemento.attr("valido", "false");
        }

        function limpiar(elemento) {
            elemento.css("background", "PaleGreen");
            elemento.next().html("");
            elemento.attr("valido", "true");
        }
</script>
<form name="formulario" id="pp">
```

```
<div name="rqr"></div>
<table>
	<tr name="uuu">
		<td name="psst">Nombre: </td>
		<td id="pp">
		<input id="nombre" class="required" type="text"/>
		<span></span></td>
	</tr>
	<tr>
		<td>Edad: </td>
		<td>
		<input type="text" class="numero" />
		<span></span></td>
	</tr>
	<tr>
		<td>CP: </td>
		<td>
		<input type="text" maxlength="5" class="fiveDigits" />
		<span></span></td>
	</tr>
	<tr>
		<td>Fecha Nacimiento: </td>
		<td>
		<input type="text" class="fecha" />
		<span></span></td>
	</tr>
	<tr>
```

```
                    <td>mail: </td>
                    <td>
                    <input type="text" class="mail" />
                    <span></span></td>
                </tr>
        </table>
        <input type="button" id="enviar" value="Enviar"/>
</form>
```

<u>Validación con plugin</u>

<u>http://bassistance.de/jquery-plugins/jquery-plugin-validation/</u>

```
<script>
  $(document).ready(function(){
    $("#myform").validate();
  });
</script>
```

```
class="required number email url"
minlength="5"
maxlength="5"
equalTo="#idPassword"
```

Ejemplos:

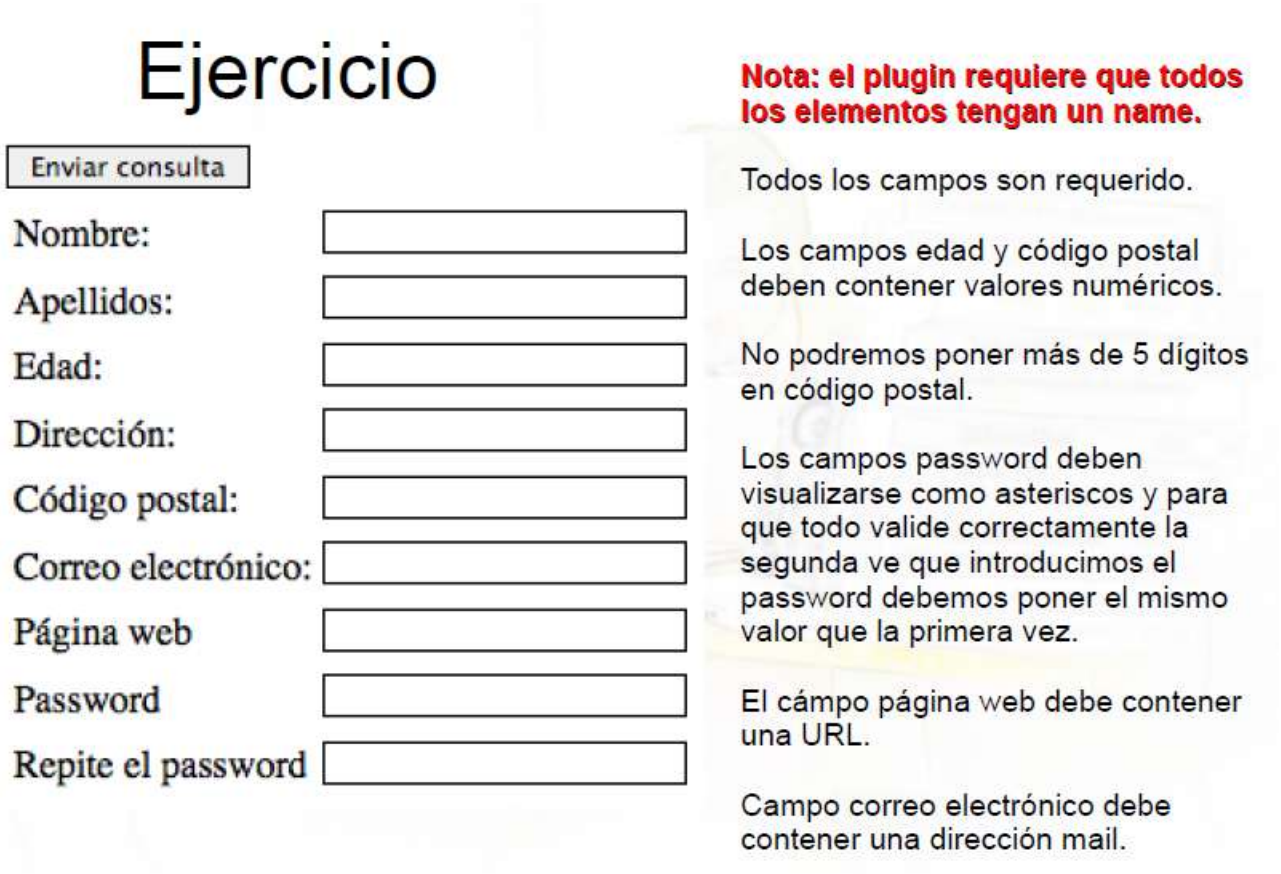
```
<label for="idFieldEdad">Edad: </label>
<input class="required number" id="idFieldEdad" name="fieldEdad" /><br/>


<label for="idFieldCP">Código postal: </label>
<input class="required number" minlength="5" maxlength="5" id="idFieldCP"
name="fieldCP" /><br/>


<label for="idFieldPagina">Página web </label>
<input class="required url" id="idFieldPagina" name="fieldPagina" />
```

```
<label for="idPassword">Password</label>
<input type="password" class="required" id="idPassword"
name="fieldPassword" />


<label for="idPassword2">Repite el password</label>
<input type="password" class="required" equalTo="#idPassword"
id="idPassword2" name="fieldPasswordb" />
```



http://pablomonteserin.com/apuntes/javascript/jquery_exercises/25-validarFormularios/conPlugin/ejercicio/validationSinRules.html

<u>Validación de formularios con el plugin de validación de jQuery y reglas</u>

Estructura

```
$(document).ready(function(){
    $("#myform").validate({
    rules: {
        fieldNombre:{//los name de los elementos
            required:true
        },
        ...
    },
    messages: {
        fieldNombre: "Por favor, inserte su nombre",
        fieldEdad: {
            required: "Por favor, inserte la edad",
            number: "Sólo puedes introducir números"
        },
        ...
    },
    submitHandler: function(){
        alert("rqr");
  }
    });
});
```

rules
required:true,
number:true,
minlength:3,
maxlength:5,
min:300,
max:600
email:true
url:true
equalTo: "#password"

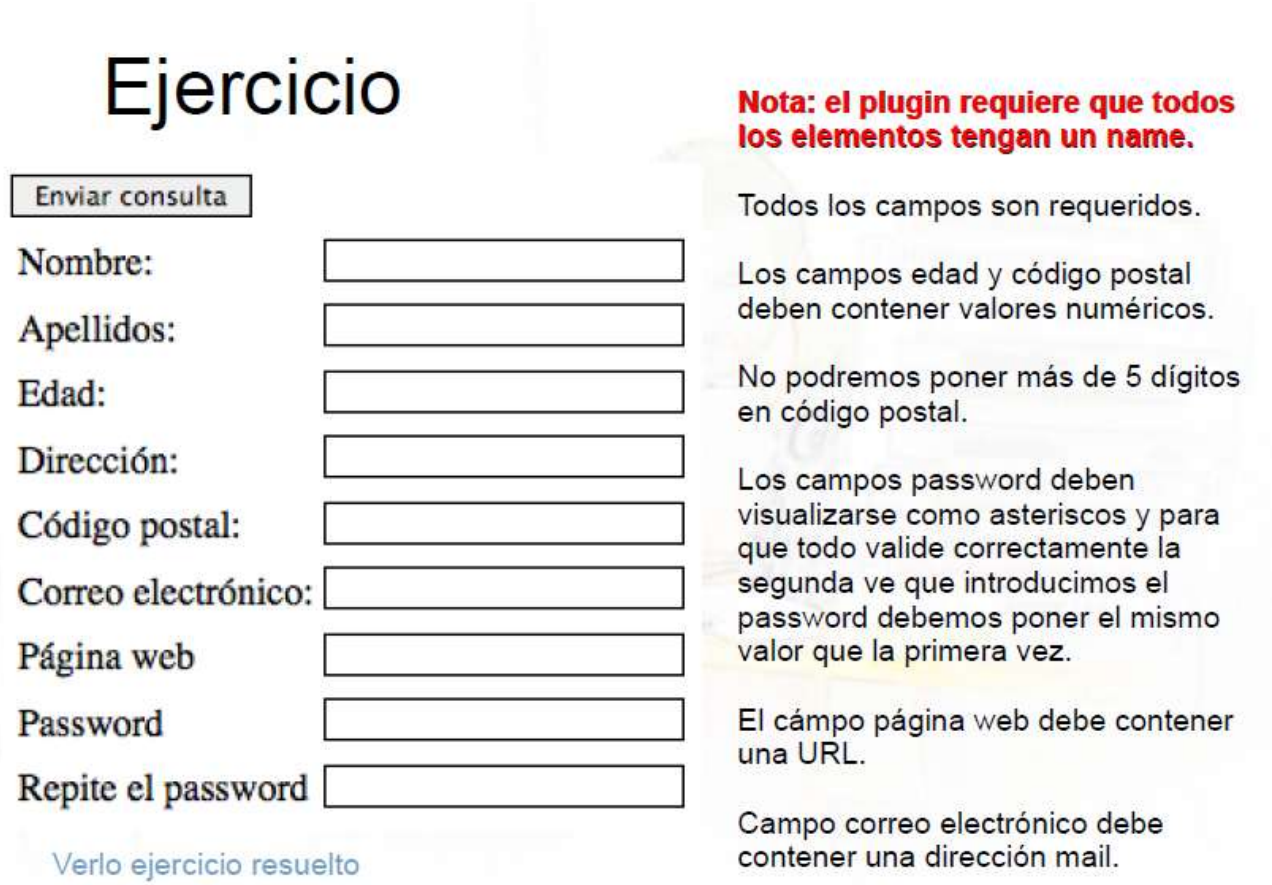
Ejemplos:

```
$(document).ready(function(){
   $("#myform").validate({
        rules: {
                //los name de los elementos
        fieldNombre:{
                required:true
        },
        fieldApellidos:{
                required:true
        },
        fieldEdad:{
                required:true,
                digits:true
        },
        fieldDireccion:{
                required:true
        },
        fieldCP:{
                required:true,
                digits:true,
                minlength:5,
```

```
                maxlength:5
        },

        fieldCorreo:{
                email:true,
                required:true
        },
        fieldPagina:{
                url:true,
                required:true
        },
        fieldPassword:{
                required:true
                },
                fieldPasswordb:{
                        required:true,
                        equalTo: "#idPassword"
        },

        },
        messages: {
                fieldNombre: "Por favor, inserte su nombre",
                fieldApellidos: "Por favor, inserte sus apellidos",
                fieldEdad: {
                        required: "Por favor, inserte la edad",
                        number: "Sólo puedes introducir números"
                },
                fieldPassword: {
                        required: "Por favor, inserte su password",
                },
                fieldPasswordb: {
                        required: "Por favor, repita su password",
                        equalTo: "Debe introducir el password introducido
previamente"
                },
                fieldCorreo: "Por inserte un mail válido",
                fieldDireccion: "Debe introducir su direccion",
                fieldCP : {
                        required: "Debe introducir su CP",
                        number: "Sólo puede escribir números",
                        minlength: "Debe introducir al menos 5 dígitos"
                },
                fieldPagina: "Debe introducir una url válida"
        },
        submitHandler: function(){
                alert("rqr");
   }
        });
 });
```



http://pablomonteserin.com/apuntes/javascript/jquery_exercises/25-validarFormularios/conPlugin/ejercicio/validationConRules.html

Ejercicio. Modifica el ejercicio 3 del boletín 2 del tema 5 de tres formas distintas, con jQuery, con plugin sin reglas y con plugin con reglas.

8. jQuery y CSS

8.1. jQuery – Obtener y establecer clases CSS

Con jQuery, es fácil de manipular elementos CSS.

jQuery tiene varios métodos para la manipulación de CSS. Vamos a ver en los siguiente métodos:

- addClass() - Añade una o más clases a los elementos seleccionados
- removeClass() - Elimina una o más clases de los elementos seleccionados
- toggleClass() - Cambia entre añadir / eliminar las clases de los elementos seleccionados.
- css() - Establece o devuelve el atributo de estilo.

Ejemplo de estilos

La siguiente hoja de estilo se utilizará para todos los ejemplos de esta página:

```
.important
{
font-weight: bold;
font-size: xx-large;
}
.blue
{
color: blue;
}
```

Método addClass()

El siguiente ejemplo muestra cómo agregar atributos de clase de diferentes elementos.

Ejemplo

```
$("button").click (function () {
        $("h1, h2, p").addClass ("blue");
        $("div").addClass ("important blue");
});
```

Método removeClass()

El siguiente ejemplo muestra cómo eliminar un atributo de clase específica de diferentes elementos:

Ejemplo

```
$("button").click (function () {
        $("h1, h2, p").removeClass ("blue");
});
```

Método toggleClass()

El siguiente ejemplo muestra cómo utilizar el método jQuery toggleClass (). Este método alterna entre añadir / eliminar las clases de los elementos seleccionados:

Ejemplo

```
$("button").click (function () {
        $("h1, h2, p").toggleClass ("blue");
});
```

8.2. jQuery – Obtener y establecer propiedades CSS

Método css()

El método css() establece o devuelve una o varias propiedades de estilo de los elementos seleccionados.

Devolver una propiedad CSS

Para devolver el valor de una propiedad CSS especificado, utilice la siguiente sintaxis:

css *("propertyname");*

Ejemplo

```
$("p").css ("background-color");
```

Establecer una propiedad CSS

Para establecer una propiedad CSS especificado, utilice la siguiente sintaxis:

css *("propertyname", "valor");*

Ejemplo

$("p").css ("background-color", "yellow");

Establecer varias propiedades CSS

Para establecer múltiples propiedades CSS, utilice la siguiente sintaxis:

css *({"propertyname": "valor", "propertyname": "valor",* ...});

Ejemplo

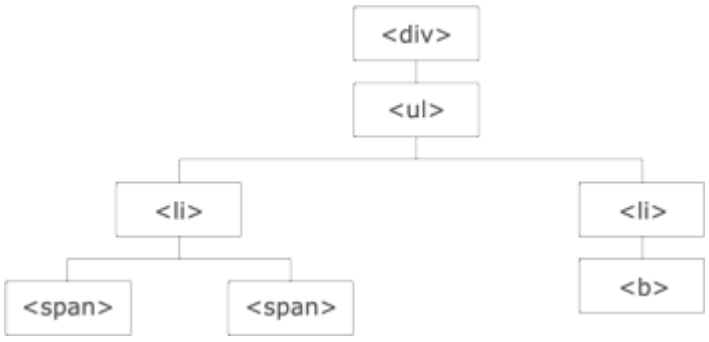
$("p").css ({"background-color": "yellow", "font-size": "200%"});

9. JQuery desplazamientos

¿Qué es el desplazamiento?

Significa "moverse a través de", se utilizan para "encontrar" (o seleccionar) elementos HTML en función de su relación con otros elementos. Comience con una selección y pasar a través de esa selección hasta llegar a los elementos que desee.

La imagen siguiente muestra un árbol. Con los métodos que jQuery proporciona para los desplazamientos, puede moverse fácilmente hacia arriba (antepasados), abajo (descendientes) y de lado (hermanos) en el árbol de familia, a partir del elemento seleccionado (actual). Este movimiento se llama desplazamiento - o en movimiento a través de - el DOM.



- El elemento es el **padre** de <ul>, y un **antepasado** de todo el interior de la misma
- El elemento <ul> es el **padre** de ambos elementos <li>, y un **niño** de
- El elemento <li> izquierda es el **padre** de , **hijo** de <ul> y un **descendiente** de
- El elemento es un **niño** de la <li> izquierda y un **descendiente** de <ul> y
- Los dos elementos <li> son **hermanos** (comparten el mismo padre)

- El elemento <li> derecha es el **padre** de <b>, **hijo** de <ul> y un **descendiente** de
- El elemento <b> es un **niño** de la <li> derecho y un **descendiente** de <ul> y

Atravesando el DOM

jQuery ofrece una variedad de métodos que nos permite recorrer el DOM.

Acceso a los antecesores

Con jQuery se puede recorrer el árbol DOM para encontrar a antepasados de un elemento. Tres métodos de jQuery útiles para recorrer el árbol DOM son:

Método parent()

El método parent() devuelve el elemento primario directo del elemento seleccionado. Este método sólo atravesar un solo nivel en el árbol DOM.

Ejemplo:

El ejemplo siguiente devuelve el elemento primario directo de cada elemento :

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

  $("span").parent();

});
```

Método parents()

El método parents() devuelve todos los elementos antecesores del elemento seleccionado, todo el camino hasta el elemento raíz del documento (<html>).

Ejemplo:

El ejemplo siguiente devuelve todos los antepasados de todos los elementos :

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

  $("span").parents();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parents

También puede utilizar un parámetro opcional para filtrar la búsqueda de antepasados.

Ejemplo: El ejemplo siguiente devuelve todos los antepasados de todos los elementos que son elementos <ul>:

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

  $("span").parents("ul");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parents2

Método parentsUntil()

El método parentsUntil () devuelve todos los elementos antecesores entre dos argumentos dados. Sin incluir a ambos.

Ejemplo: El ejemplo siguiente devuelve todos los elementos antecesores entre un y un elemento

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

  $("span").parentsUntil("div");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_parentsuntil

Acceso a los descendientes

Con jQuery se puede recorrer el árbol DOM para encontrar descendientes de un elemento. Dos métodos de jQuery útiles para recorrer el árbol DOM son:

Método children()

El método children () devuelve todos los hijos directos del elemento seleccionado. Este método sólo atravesar un solo nivel por el árbol DOM.

Ejemplo: El ejemplo siguiente devuelve todos los elementos que son hijos directos de cada elemento :

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

```
  $("div").children();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_children

También puede utilizar un parámetro opcional para filtrar la búsqueda de los niños.

El ejemplo siguiente devuelve todos <p> elementos con el nombre de la clase "1", que son hijos directos de :

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

  $("div").children("p.1");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_children2

Método find

El méotod find() devuelve los elementos descendientes del elemento seleccionado, todo el camino hasta el último descendiente.

Ejemplo: El ejemplo siguiente devuelve todos los elementos que son descendientes de :

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

  $("div").find("span");

});
```

El ejemplo siguiente devuelve todos los descendientes de :

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

  $("div").find("*");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_find2

Acceso a los hermanos

Con jQuery se puede recorrer de lado en el árbol DOM para encontrar hermanos de un elemento. Hay muchos métodos jQuery útiles para atravesar de lado en el árbol DOM:

Método siblings()

El método siblings() devuelve todos los elementos del mismo nivel del elemento seleccionado.

Ejemplo: devuelve todos los elementos del mismo nivel de <h2>:

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

  $("h2").siblings();

});
```

También puede utilizar un parámetro opcional para filtrar la búsqueda de los hermanos.

El ejemplo siguiente devuelve todos los elementos del mismo nivel de <h2> que son elementos <p>:

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

  $("h2").siblings("p");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_siblings2

Método next()

El método next () devuelve el siguiente elemento relacionado del elemento seleccionado.

El ejemplo siguiente devuelve el siguiente hermano del <h2>:

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

  $("h2").next();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_next

Método nextAll()

El método nextAll () devuelve todos los elementos del mismo nivel siguiente del elemento seleccionado.

El ejemplo siguiente devuelve todos los elementos del mismo nivel siguiente de <h2>:

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

  $("h2").nextAll();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_nextall

Método nextUntil()

El método nextUntil() devuelve todos los elementos relacionados próximos entre dos argumentos dados. Sin incluirlos.

El ejemplo siguiente devuelve todos los elementos del mismo nivel entre un <h2> y un elemento <h6>:

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

  $("h2").nextUntil("h6");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_nextuntil

Los métodos prev(), prevAll () y prevUntil () funcionan igual que los métodos anteriores, pero con la funcionalidad inversa: vuelven elementos relacionados anteriores (recorrer hacia atrás a lo largo de elementos del mismo nivel en el árbol DOM, en lugar de hacia delante).

Acotar la búsqueda de elementos

Los tres métodos de filtrado más básicas son first(), last() y la eq(), que le permite seleccionar un elemento específico en función de su posición en un grupo de elementos.

Otros métodos de filtrado, como filter() y not() le permiten seleccionar los elementos que coinciden o no coinciden, un cierto criterio.

Método first()

Devuelve el primer elemento de los elementos seleccionados.

El ejemplo siguiente selecciona el primer elemento <p> dentro del primer elemento :

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

  $("div p").first();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_first

Método last()

Devuelve el último elemento de los elementos seleccionados.

El ejemplo siguiente selecciona el último elemento <p> dentro del último elemento :

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

  $("div p").last();

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_last

Método eq()

Devuelve un elemento con un número de índice específico de los elementos seleccionados.

Los números de índice comenzarán a 0, por lo que el primer elemento tendrá el número de índice 0 y no 1. El ejemplo siguiente selecciona el segundo elemento <p> (número de índice 1):

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

  $("p").eq(1);

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_eq

Método filter()

El método filter() le permite especificar un criterio. Se devolverán los elementos que no coinciden con los criterios se eliminan de la selección, y los que coinciden los devuelven.

El ejemplo siguiente devuelve todos los elementos <p> con nombre de la clase "intro":

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

  $("p").filter(".intro");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_filter

Método not()

El método no () devuelve todos los elementos que no coinciden con los criterios.

El ejemplo siguiente devuelve todos los elementos <p> que no tienen nombre de la clase "intro":

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

  $("p").not(".intro");

});
```

http://www.w3schools.com/jquery/tryit.asp?filename=tryjquery_not

Ejercicios

41. Añade una clase css a distintos elementos de HTML mediante un evento asociado a un botón y como gestor del evento el método addClass. Mediante otro evento asociado a un segundo botón elimina la clase css de los elementos utilizando como gestor el evento removeClass().

42. Mediante un evento asociado a un botón gestionado con el método toggleClass añade y elimina una clase css a distintos elementos HTML.

43. Tenemos un documento con 4 párrafos cada uno con un color de fondo. mediante un evento asociado a un botón devuelve mediante una alerta el color de fondo del primer párrafo.

44. En el mismo documento anterior, mediante un evento asociado a un botón. Cambia el color de fondo de los cuatro párrafos a amarillo y aumenta el tamaño de la letra.

45. Selecciona de una lista de 5 ciudades las tres últimas, debes resaltarla con un color de fondo rojo.

46. Crea una tabla de 8 filas, a continuación pon el fondo rojo a todas aquellas que estén por encima de la tercera y pon el fondo azul a todas aquellas que estén por debajo de la tercera.

10. Varios jQuery

Método each

each() es un método que se utiliza sobre un conjunto de elementos que hayamos seleccionado con la función jQuery. Con each() realizamos una **iteración por todos los elementos** del DOM que se hayan seleccionado.

El método each() **recibe una función que es la que se tiene que ejecutar para cada elemento** y dentro de esa función con la variables **"this"** tenemos una referencia a ese elemento del DOM. Adicionalmente, **la función** que se envía a each(), **puede recibir un parámetro que es el índice actual sobre el que se está iterando**.

Por ejemplo, si tenemos:

```
$("p").css("background-color", "#eee");
```

Como ya sabemos, con $("p") seleccionamos todos los párrafos de la página. Luego con el método CSS asignamos un estilo, en este caso para cambiar el color del fondo. Esto en realidad jQuery lo hace con una iteración con todos los párrafos de la página, sin que tengamos que hacer nosotros nada más y es genial que se permita en el uso del framework. ¿Pero qué pasa si queremos cambiar el fondo de los párrafos utilizando colores alternos?

En este caso no podemos hacerlo en una sola línea de código, pero each nos vendrá como anillo al dedo.

Imaginemos que tenemos una serie de párrafos en la página y queremos cambiar el color de fondo a los mismos, de manera que tengan colores alternos, para hacer dinámicamente un típico diseño para los listados.

Entonces podríamos hacer lo siguiente:

```
$("p").each(function(i){
if (i%2==0){
    $(this).css("background-color", "#eee");
} else {
    $(this).css("background-color", "#ccc");
}
});
```

Retornando valores en la función que enviamos a each.

Ahora vamos a ver un par de posibilidades interesantes al utilizar each. Resulta que la **función que enviamos como parámetro a each() puede devolver valores** y dependiendo de éstos, conseguir comportamientos parecidos a los conocidos break o continue de los bucles JavaScript.

Si la función devuelve "**false**", se consigue **detener por completo el proceso de iteraciones** de each((). Esto es como si hiciéramos el típico **"break"**.

Si la función devuelve **"true"**, se consigue **pasar directamente a la próxima iteración del bucle**. Es como hacer el típico **"continue"**.

Para ver estos dos casos realizaremos otro ejemplo de uso de each. Tenemos varios DIV, donde cada uno tiene un texto.

```
<div>red</div>
<div>blue</div>
<div>red</div>
<div>white</div>
<div>red</div>
<div>green</div>
<div>orange</div>
<div>red</div>
<div>nada</div>
<div>red</div>
<div>blue</div>
```

Ahora queremos hacer un recorrido a esos DIV y en cada uno, mirar el texto que aparece. Entonces colocaremos como color del texto del DIV el color que aparece escrito en el DIV. Pero con dos casos especiales:

• Si el texto del DIV es "white", entonces no queremos hacer nada con ese elemento.

• Si el texto del DIV es "nada", entonces detendremos el bucle y dejaremos de colorear los siguientes elementos.

Esto lo podríamos hacer con el siguiente código:

```
$("div").each(function(i){
elemento = $(this);
if(elemento.html() == "white")
return true;
if(elemento.html() == "nada")
return false;
elemento.css("color", elemento.html());

});
```

Método size() y propiedad length

Método size() del Core de jQuery

Este método sirve, como decimos, para obtener el número de elementos seleccionados con la función jQuery.

Por ejemplo, veamos este código:

```
var parrafos = $("p");
alert ("Hay " + parrafos.size() + " párrafos en la página");
```

Con la primera línea selecciono todos los párrafos de la página y los meto en el objeto jQuery de la variable "parrafos".

Mediante la segunda línea muestro el número de párrafos encontrados, con una llamada al método size().

Propiedad length del objeto jQuery

La propiedad length, que existe en cualquier objeto jQuery, nos sirve para **obtener el número de elementos de la página que hemos seleccionado**. Lo interesante de esta propiedad es que almacena directamente este valor, por lo que es **más rápido** y más aconsejable que calcular los elementos seleccionados con el método size().

Tanto el método size() con la propiedad length devolverán el mismo valor, siendo las únicas diferencias la mencionada rapidez adicional de la propiedad y el acceso a este valor, que como es una propiedad, se accede a través del operador punto, pero sin colocar los paréntesis después de length. Por ejemplo, veamos este código:

```
var ElementosMitexto = $(".mitexto");
alert ("Hay " + ElementosMitexto.length + " elementos de la clase mitexto");
```

Con la primera línea estamos utilizando la función jQuery para seleccionar todos los elementos de la página que tienen la clase CSS "mitexto". Con la segunda línea se muestra en una caja de alerta el número de elementos seleccionados con ElementosMitexto.length.

Método data() y removeData()

Sirven para almacenar, consultar y eliminar cualquier tipo de dato en elementos de la página.

En algunas ocasiones resulta útil almacenar variables u objetos en determinados elementos de la página. Aunque quizás no es una acción muy corriente en los primeros pasos con jQuery, en el futuro encontraréis que resulta útil y veréis herramientas y plugins que utilizan este mecanismo para su operativa. De modo que conviene al menos saber que esto es posible y conocer de qué manera podemos utilizar los elementos de la página para guardar cosas en ellos.

Para ello vamos a comentar dos métodos distintos que forman parte del core de jQuery:

Método data()

Este método del objeto jQuery sirve tanto para guardar un dato en un elemento como para consultarlo. Según el número de parámetros que reciba, realiza una u otra acción.

• Si recibe un parámetro data (**nombre**): **devuelve el valor** que haya en el dato cuyo nombre se pasa por parámetro.

• Si recibe dos parámetros data (**nombre, valor**): **almacena un dato**, cuyo nombre recibe en el primer parámetro, con el valor que recibe en el segundo parámetro.

Como data() es un método que pertenece al objeto jQuery, podemos almacenar estos pares (dato, valor) en cualquiera de los elementos que seleccionemos con la función jQuery().

Veamos un caso de uso simple. Por ejemplo tenemos un elemento de la página como este:

```
<div id="capa">
En este div voy a guardar y leer datos sobre este elemento.
</div>
```

Ahora podríamos usar le método data() de la siguiente manera:

```
$("#capa").data("midato","mivalor");
```

Con esta línea hemos guardado un dato llamado "midato" con el valor "mivalor", en el elemento con identificador (atributo id) "capa".
Ahora podríamos leer ese dato en cualquier momento para acceder a su valor, de la siguiente manera:

```
alert($("#capa").data("midato"));
```

En esta línea de código extraemos el dato "midato" del elemento con identificador "capa" y lo mostramos en una caja de alerta.

Método removeData()

Este método sirve para **eliminar un dato de un elemento** y su funcionamiento es tan simple como enviar por parámetro el dato que se quiere eliminar del elemento.

```
$("#capa").removeData("midato")
```

Con esta línea habríamos eliminado el dato llamado "midato" del elemento con identificador "capa".

Ejemplo completo de los métodos data() y removeData() del Core de jQuery

Veamos un ejemplo completo del uso de estos métodos que acabamos de aprender. Se trata de una página que tiene un elemento sobre el que vamos a guardar datos. Además tiene tres botones para guardar un dato, leerlo y borrarlo. El dato que se guardará tendrá como valor lo que se haya escrito en un campo de texto que aparece también en la página.

Tenemos, para comenzar, un elemento de la página, que es donde vamos a guardar los pares dato-valor con data().

```
<div id="division">
En esta división (elemento id="division") voy a guardar datos con la función data y luego los voy a leer.
</div>
```

Luego tendremos este formulario, que contiene el campo de texto así como los tres botones de los que hemos hablado.

```
<form name="formul">
Escribe un valor a guardar, leer o eliminar:
<input type="text" name="valor" id="valor">
<br>
<input type="button" value="guardar dato" id="guardar">
<input type="button" value="leer dato" id="leer">
<input type="button" value="eliminar dato" id="eliminar">
</form>
```

Ahora se trata de asignar los comportamientos a estos botones con Javascript, haciendo uso de jQuery.

Este sería el script para agregar el evento click al botón de guardar datos.

```
$("#guardar").click(function(evento){
var valor = document.formul.valor.value;
//Esta misma línea de código se puede codificar así también con jQuery
//var valor = $("#valor").attr("value");
$("#division").data("midato",valor);
$("#division").html('He guardado en este elemento (id="division") un dato llamado "midato" con el valor "' + valor + '"');
});
```

Como se puede ver, primero se recibe el texto del campo de texto que había en el formulario. Para ello se muestran dos maneras de hacerlo:

- A través de la jerarquía de objetos del navegador, con document.formul.valor.value
- A través de su identificador, con un método de jQuery llamado attr() que sirve para recuperar el valor de un atributo HTML pasado por parámetro sobre el elemento que recibe el método. Este modo de obtener el atributo value del campo de texto está comentado, pues sólo lo quería señalar, para que se vea el modo de acceder a un elemento de formulario utilizando las funciones del framework Javascript jQuery.

Luego, se guarda el dato "midato" con el valor que se recuperó del atributo value del campo de texto. Para ello utilizamos el método data() tal como comentábamos.

Por último se muestra un mensaje en el HTML del elemento con id="division", por medio del método html() de jQuery, para informar sobre la acción que acabamos de realizar.

Ahora mostramos el código para asignar un comportamiento al evento click sobre el segundo botón:

```
$("#leer").click(function(evento){
valor = $("#division").data("midato");
$("#division").html('En este elemento (id="division") leo un dato llamado "midato" con el valor "' + valor + '"');
});
```

Como se puede ver, se recupera el valor del dato "midato" guardado sobre el elemento "#division" (etiqueta HTML con id="division"), y se almacena en una variable. Luego se crea un mensaje para mostrar el valor del dato.

Para acabar, tenemos el código del evento click sobre el botón de eliminar el contenido de un dato, que hace uso de removeData().

```
$("#eliminar").click(function(evento){
$("#division").removeData("midato");
$("#division").html('Acabo de eliminar del elemento (id="division") el dato llamado "midato"');
});
```

Como se verá, el método removeData() se invoca sobre el elemento que tiene el dato que pretendemos eliminar. Más tarde se muestra un mensaje informando sobre la acción que se ha realizado.

Para comprobar el funcionamiento de estos métodos habría que crear un dato, escribiendo el valor en el campo de texto y pulsando el botón "guardar dato". Luego podríamos leer ese dato con el botón "leer dato". Por último

podríamos eliminar el dato con el botón "eliminar dato". Si, una vez eliminado pulsamos sobre el botón de "leer dato" veremos que el valor del dato aparece como "undefined", puesto que ese dato ha sido borrado (esto también ocurre si no se ha guardado ningún dato todavía, por ejemplo cuando se acaba de cargar la página).

Consideraciones interesantes de data() y removeData()

- **Admite cualquier tipo de dato**: Podemos guardar lo que deseemos por medio del método data(). Los ejemplos anteriores hemos guardado simplemente cadenas de texto, pero soportaría cualquier tipo de variable, numérica, un array o incluso un objeto Javascript o jQuery.
- **Se guarda un dato por cada elemento del objeto jQuery selecionado**: En caso que en el objeto jQuery sobre el que estemos almacenando cosas con data() haya referencias a varios elementos de la página, el dato se almacena en todos los elementos.
- **Los objetos se almacenan por referencia**: En el caso que estemos almacenando un objeto Javascript con data() sobre uno o varios elementos, no se copia el objeto, sino que se asigna por referencia. Esto quiere decir que no se harían copias independientes del objeto a guardar, sino que permanecería tal cual y lo que se asignaría como dato es una referencia a ese único objeto.

  Otra utilidad, es que nos permite el **acceso a los atributos data de HTML5.**

  Ejemplo:

```
<div id="texto" data-author="Ricardo Miranda" data-date="2012-06-21">
   <h4>Lorem ipsum</h4>
   <p>
      Lorem ipsum dolor sit amet, ius integre eligendi et.
   </p>
</div>
```

```
<script type="text/javascript">
   $(function() {
      alert("El texto fue escrito por " +
$('#texto').data('author'));
   });
</script>
```

Método get()

El método get () obtiene los elementos DOM especificado por el selector.

Ejemplo: Obtener el nombre y el valor del primer elemento <p>:

```
$("button").click(function(){
  x=$("p").get(0);
  $("div").text(x.nodeName + ": " + x.innerHTML);
});
```

Método index()

El método index() devuelve la posición de índice de los elementos especificados en relación con otros elementos especificados. Nota: Si no se encuentra el elemento, índice () devolverá -1.

Ejemplo: Obtener el índice del elemento <li> al que hacemos clic, en relación a sus hermanos:

```
$("li").click(function(){
  alert($(this).index());
});
```

Método toArray()

El método toArray () devuelve los elementos coincidentes por el selector de jQuery como una matriz.

Ejemplo: Convertir los elementos <li> a una matriz, entonces alertar al innerHTML de los elementos de la matriz:

```
$("button").click(function(){
  x=$("li").toArray()
  for (i=0;i<x.length;i++)
    {
    alert(x[i].innerHTML);
    }
});
```

Método $.noConflict()

El método noConflict () libera el control de jQuery de la variable $.

Ejemplo: Utilice el método noConflict () para especificar un nuevo nombre para la variable de jQuery:

```
var jq=$.noConflict();
```

Consejo: Este método es útil cuando otras bibliotecas JavaScript utilizan el $ para sus funciones.

EJERCICIO

1. Haz una página web que incluya varios textarea y todos ellos incluyan a la derecha del campo información sobre el número de caracteres que tenga escrito. Debe mostrar para todos el número de caracteres escritos, excepto si es 0 que debe indicar que el campo está vacío.
2. Haz una página web que incluya varios enunciados de ejercicios, al cargar la página para cada uno de ellos tiene que incorporar enlaces al anterior y al siguiente ejercicio, excepto para el primero y el último. Si es el último debe mostrar un enlace al anterior y al primer ejercicio. Si es el primero, sólo el siguiente.

11. PLUGINS

Los plugins en jQuery nos permiten hacer desarrollos que podremos reutilizar con facilidad en diversos sitios y que también podrán usar otros desarrolladores. Los plugins te ayudarán a hacer código de calidad en jQuery.

11.1. Qué son los Plugins

Los plugins son la utilidad que pone jQuery a disposición de los desarrolladores para **ampliar las funcionalidades del framework**. Por lo general servirán para hacen cosas más complejas necesarias para resolver necesidades específicas, pero las hacen de manera que puedan utilizarse en el futuro en cualquier parte y por cualquier web.

En la práctica un pluging no es más que una **función que se añade al objeto jQuery** (objeto básico de este framework que devuelve la función jQuery para un selector dado), para que a partir de ese momento responda a nuevos métodos. Como ya sabemos, en este framework todo está basado en el objeto jQuery, así que con los plugins podemos añadirle nuevas utilidades.

Claro que, para conseguir todo esto, será necesario que programes los plugins atendiendo a una serie de normas, bastante sencillas pero importantes para asegurar que se puedan utilizar en cualquier parte y para cualquier selector de jQuery.

11.2.- Cómo se crea un plugin de jQuery

Los plugins en jQuery se crean **asignando una función a la propiedad "fn" del objeto jQuery**. A partir de entonces, esas funciones asignadas se podrán utilizar en cualquier objeto jQuery, como uno de los muchos métodos que dispone dicho objeto principal del framework.

Nota: La creación de plugins, para ampliar las funcionalidades de jQuery, es una cosa tan básica que la mayoría de las funciones con las que está dotado el propio framework están incluidas en el objeto jQuery por medio de plugins. Es decir, en la construcción del framework en muchas de las ocasiones simplemente se crean plugins para extenderlo. Así pues, esta técnica es usada,

no sólo por terceros desarrolladores, para crear nuevos componentes, sino también por el propio equipo de jQuery para el diseño base de este framework.

A modo de ejemplo, podemos ver a continuación un código fuente de un plugin muy sencillo:

```
jQuery.fn.desaparece = function() {
this.each(function(){
elem = $(this);
elem.css("display", "none");
});
return this;
};
```

Este plugin permitiría hacer desaparecer a los elementos de la página y podríamos invocarlo por ejemplo de la siguiente manera:

```
$("h1").desaparece();
```

11.3.- Reglas para el desarrollo de plugins en jQuery

Para construir plugins en jQuery tenemos que seguir una serie de normas, para asegurar que los plugins funcionen como deben y los pueda utilizar cualquier desarrollador en cualquier página web.

Aquí puedes ver un listado normas, que son sólo unas pocas, pero que resultan tremendamente importantes.

• El archivo que crees con el código de tu plugin lo debes **nombrar como jquery.[nombre de tu plugin].js**. Por ejemplo jquery.desaparece.js.

• **Añade las funciones como nuevos métodos por medio de la propiedad fn del objeto jQuery**, para que se conviertan en métodos del propio objeto jQuery.

• Dentro de los métodos que añades como plugins, la palabra **"this" será una referencia al objeto jQuery que recibe el método**. Por tanto, podemos utilizar "this" para acceder a cualquier propiedad del elemento de la página con el estamos trabajando.

• **Debes colocar un punto y coma ";" al final de cada método que crees como plugin**, para que el código fuente se pueda comprimir y siga

funcionando correctamente. Ese punto y coma debes colocarlo después de cerrar la llave del código de la función.

• El método debe **retornar el propio objeto jQuery sobre el que se solicitó la ejecución del plugin**. Esto lo podemos conseguir con un **return this**; al final del código de la función.

• Se debe usar **this.each para iterar sobre todo el conjunto de elementos que puede haber seleccionados**. Recordemos que los plugins se invocan sobre objetos que se obtienen con selectores y la función jQuery, por lo que pueden haberse seleccionado varios elementos y no sólo uno. Así pues, con this.each podemos iterar sobre cada uno de esos elementos seleccionados. Esto es interesante para producir código limpio, que además será compatible con selectores que correspondan con varios elementos de la página.

• **Asigna el pluging siempre al objeto jQuery, en vez de hacerlo sobre el símbolo $,** así los usuarios podrán usar alias personalizados para ese plugin a través del método noConfict(), descartando los problemas que puedan haber si dos plugin tienen el mismo nombre.

Estas reglas serán suficientes para plugins sencillos, aunque quizás en escenarios más complejos en adelante necesitaremos aplicar otras reglas para asegurarnos que todo funcione bien.

11.4.- Ejemplo de un plugin en jQuery

Ahora que ya sabemos las reglas básicas para hacer plugins podemos crear uno por nuestra cuenta que nos sirva para practicar lo que hemos aprendido. Te sugiero que identifiques los lugares donde hemos aplicado cada una de las anteriores normas de la lista, o al menos las que se puedan aplicar en este plugin tan simple que vamos a ver.

El plugin que vamos a construir sirve para hacer que los elementos de la página parpadeen, esto es, que desaparezcan y vuelvan a aparecer en un breve instante. Es un ejemplo bien simple, que quizás tenga ya alguna utilidad práctica en tu sitio, para llamar la atención sobre uno o varios elementos de la página.

Para hacerlo, utilizaremos otras funciones del framework como fadeOut() (para hacer desaparecer al elemento) y fadeIn() (para que aparezca de nuevo).

```
jQuery.fn.parpadea = function() {
this.each(function(){
elem = $(this);
elem.fadeOut(250, function(){
$(this).fadeIn(250);
});
});
return this;
};
```

Con this.each creamos un bucle para cada elemento que pueda haberse seleccionado para invocar el plugin. Con elem=$(this) conseguimos extender a this con todas las funcionalidades del framework y el objeto jQuery resultante guardarlo en una variable. Luego invocamos fadeOut(), enviando como parámetro un número que son los milisegundos que durará el efecto de desaparecer el elemento. Luego enviamos como parámetro una nueva función que es un callback, que se ejecutará cuando haya terminado fadeOut() y en esa función callback se encargará simplemente de ejecutar un fadeIn() para mostrar de nuevo el elemento.

Ahora veamos cómo podríamos invocar este plugin:

```
$(document).ready(function(){
//parpadean los elementos de class CSS "parpadear"
$(".parpadear").parpadea();
//añado evento clic para un botón. Al pulsar parpadearán los elementos de clase parpadear
$("#botonparpadear").click(function(){
$(".parpadear").parpadea();
})
})
```

Dado el código anterior, al abrir la página parpadearán los elementos de la clase "parpadear" y luego habrá un botón que repetirá la acción de parpadear cuando se pulse.

11.5.- Gestión de opciones en plugins jQuery

Manera de gestionar opciones en los plugins de jQuery, a través de un **objeto de options enviado al invocar el plugin**, para hacerlos un poco más versátiles y con configuración más fácil.

Cuando desarrollamos plugins en jQuery debemos atender a una serie de normas básicas para que estén bien creados y puedan funcionar en cualquier ámbito. Pero además tenemos una serie de patrones de desarrollo que debemos seguir de manera opcional para facilitarnos la vida a nosotros mismos y a otros desarrolladores que puedan utilizar nuestros plugins.

Una de las tareas típicas que realizaremos es la creación de un sistema para cargar opciones con las que configurar el comportamiento de los plugins. Estas opciones las recibirá el plugin como parámetro cuando lo invocamos inicialmente.

Nosotros, como desarrolladores del plugin, **tendremos que definir cuáles van a ser esas opciones de configuración y qué valores tendrán por defecto**.

La ayuda del sitio de jQuery para la creación de plugins sugiere la manera con la que realizar el proceso de configuración del plugin, por medio de un objeto de "options", que nos facilitará bastante la vida.

11.5.1.- Por qué son interesantes los options en plugins

La idea que hay detrás de la carga de opciones en los plugins ya la conocemos, que éstos sean más configurables y por lo tanto más versátiles. Pero vamos a intentar dar un ejemplo más claro sobre cómo esas opciones pueden hacer a los plugins más versátiles.

Imaginemos un plugin para mostrar una caja de diálogo. Esas cajas de diálogo permiten mostrar mensajes en una capa emergente. Esa caja podría tener diversos parámetros para configurarla, como su altura, anchura, título de

la caja, etc. Todos esos parámetros podríamos enviarlos al dar de alta la caja, con un código como este:

```
$("#capa").crearCaja(400, 200, "titulo", ...);
```

Pero eso **no es práctico**, porque el usuario debería indicar todos los parámetros para crear la caja, o al menos si no indica unos no podría indicar otros que están detrás en la lista. Luego, en el código del plugin, el desarrollador debería comprobar qué parámetros se indican, uno a uno, y darles valores por defecto si no se han indicado, etc. Todo eso ampliaría demasiado el código fuente.

Entonces, lo que se suele hacer al dar de alta el plugin, es **indicar una serie de datos con notación de objeto**:

```
$("#capa").crearCaja({
titulo: "titulo",
anchura: 400,
altura: 200,
...
});
```

El desarrollador del plugin colocará en el código fuente un objeto con las variables de configuración y sus valores por defecto. Luego, cuando se cree el plugin, **lo mezclará con el objeto de options enviado por parámetro**, con una única sentencia, con lo que obtendrá rápidamente el objeto completo de configuración del plugin que debe ser aplicado.

11.5.2.- Definir opciones por defecto en el código del plugin

Con el siguiente código podemos definir las variables de configuración por defecto de un plugin y combinarlas con las variables de options enviadas por parámetro al invocar el plugin.

```
jQuery.fn.miPlugin = function(cualquierCosa, opciones) {
//Defino unas opciones por defecto
var configuracion = {
dato1: "lo que sea",
dato2: 78
}
```

```
        //extiendo las opciones por defecto con las recibidas
        jQuery.extend(configuracion, opciones);
        //resto del plugin
        //donde tenemos la variable configuracion para personalizar el plugin
        }
```

La función principal del plugin recibe **dos parámetros**, uno "cualquierCosa" y otro "opciones". El primero supongamos que es algo que necesita el plugin, pero la configuración, que es lo que nos importa ahora, se ha recibido en el parámetro "opciones".

Ya dentro de la función del plugin, **se define el objeto con las opciones de configuración, con sus valores por defecto, en una variable llamada "configuracion".**

En la siguiente línea **se mezclan los datos de las opciones de configuración por defecto y las recibidas por el plugin al inicializarse**. Luego podremos acceder por medio de la variable "configuración" todas las opciones del plugin que se va a iniciar.

Nota: El modo en cómo se mezclan los datos por medio de **extend(),** podéis revisar en el artículo sobre el método jQuery.extend().

11.5.3.- Invocar al plugin enviando el objeto de opciones

Ahora podemos ver el código que utilizaríamos para invocar al plugin pasando las opciones que deseamos:

```
        $("#elemento).miPlugin({
        dato1: "Hola amigos!",
        dato2: true
        });
```

O podríamos enviar sólo alguno de los datos de configuración, para que el resto se tomen por defecto:

```
        $("<div></div>").miPlugin({
        dato2: 2.05
        });
```

O no enviar ningún dato al crear el plugin para utilizar los valores por defecto en todas las opciones de configuración.

```
$("p").miPlugin();
```

11.6.- Plugin Tip con opciones en jQuery

Un ejemplo de plugin en jQuery para hacer un sistema de tip más avanzado, que permite configurarse por medio de unas opciones en el plugin.

Las opciones que vamos a implementar serán las siguientes:

- Velocidad de la animación de mostrar y ocultar el tip
- Animación a utilizar para mostrar el tip
- Animación a utilizar para ocultar el tip
- Clase CSS para la capa del tip

Todas esas opciones se definen, junto con los valores por defecto que van a tomar, al crear el código del plugin.

Comenzamos por especificar, con notación de objeto, las opciones de configuración por defecto para el plugin:

```
var configuracion = {
velocidad: 500,
animacionMuestra: {width: "show"},
animacionOculta: {opacity: "hide"},
claseTip: "tip"
}
```

Ahora veamos el inicio del código del plugin, donde debemos observar que en la función que define el plugin se están recibiendo un par de parámetros. El primero es el texto del tip, que necesitamos para crear la capa del tip (Este parámetro ya aparecía en el código del plugin del artículo anterior). El segundo son las opciones específicas para configurar el plugin.

```
jQuery.fn.creaTip = function(textoTip, opciones) {
//opciones por defecto
var configuracion = {
velocidad: 500,
animacionMuestra: {width: "show"},
animacionOculta: {opacity: "hide"},
```

```
    claseTip: "tip"
    }
    //extiendo las opciones por defecto con las opciones del parámetro.
    jQuery.extend(configuracion, opciones);
    this.each(function(){
    //código del plugin
    });
    });
```

11.6.1.- Método jQuery.extend()

Quizás en este código, lo que más nos llame la atención sea el lugar donde extiendo las opciones por defecto definidas en la variable "configuracion", con las opciones específicas para el plugin concreto, recibidas por medio del parámetro "opciones".

```
    jQuery.extend(configuracion, opciones);
```

Esta sentencia es una llamada al método extend() que pertenece a jQuery. Esta función recibe cualquier número de parámetros, que son objetos, y mete las opciones de todos en el primero. Luego, después de la llamada a extend(), el objeto del primer parámetro tendrá sus propiedades más las propiedades del objeto del segundo parámetro. Si alguna de las opciones tenía el mismo nombre, al final el valor que prevalece es el que había en el segundo parámetro. Si tenemos dudas con respecto a este método, leer el artículo jQuery.extend().

Así, podemos ver cómo con extend() las propiedades por defecto del plugin se combinan con las que se envíen en las opciones. Luego, en el código del plugin, podremos acceder a las propiedades a través de la variable configuración, un punto y el nombre de propiedad que queramos acceder.

```
    configuracion.velocidad
```

11.6.2.- Código completo del plugin tip con opciones

Veamos todo el código de nuestro primer plugin en implementar el sistema de opciones:

```
jQuery.fn.creaTip = function(textoTip, opciones) {
var configuracion = {
velocidad: 500,
animacionMuestra: {width: "show"},
animacionOculta: {opacity: "hide"},
claseTip: "tip"
}
jQuery.extend(configuracion, opciones);
this.each(function(){
elem = $(this);
var miTip = $('<div class="' + configuracion.claseTip + '">' +
textoTip + '</div>');
$(document.body).append(miTip);
elem.data("capatip", miTip);
elem.mouseenter(function(e){
var miTip = $(this).data("capatip");
miTip.css({
left: e.pageX + 10,
top: e.pageY + 5
});
miTip.animate(configuracion.animacionMuestra,
configuracion.velocidad);
});
elem.mouseleave(function(e){
var miTip = $(this).data("capatip");
miTip.animate(configuracion.animacionOculta,
configuracion.velocidad);
});
});
return this;
};
```

11.6.3.- Invocar al plugin con o sin las opciones de configuración

Para acabar, vamos a invocar al plugin del tip con opciones, pero lo vamos a hacer de dos modos, uno con las opciones por defecto y otro con opciones específicas.

Así se llamaría al plugin con las opciones por defecto:

```
$("#elemento1").creaTip("todo bien...");
```

En realidad le estamos pasando un parámetro, pero no son las opciones, sino es el texto que tiene que aparecer en el tip.

Como no se indican opciones, ya que no hay segundo parámetro, se toman todas las definidas por defecto en el plugin.

Las opciones, según se puede ver en el código del plugin, se deberían enviar en un segundo parámetro cuando se llama al plugin, tal como se puede ver a continuación:

```
$("#elemento2").creaTip("Otra prueba...", {
velocidad: 1000,
claseTip: "otroestilotip",
animacionMuestra: {
opacity: "show",
padding: '25px',
'font-size': '2em'
},
animacionOculta: {
height: "hide",
padding: '5px',
'font-size': '1em'
}
});
```

Ahora hemos indicado varias opciones específicas, que se tendrán en cuenta al crear el plugin con este segundo código.

EJERCICIO

Desarrollar página web que incluya dos plugins, uno descargado de internet y uno desarrollado por ti.