UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Javascript, JQuery y JSON avanzado.

Módulo 1

INTRODUCCIÓN A JQUERY

Creación de plugins.

Presentación de la Unidad:

En esta unidad aprenderán que es un Plugin, cual es su utilidad y como armar uno.

Nuevamente pondremos en práctica los conceptos vistos hasta ahora incorporando algunos nuevos, mas avanzados, necesarios para la creación de plugins.

Veremos algunos plugins de terceros que nos resultan interesantes.

Objetivos:

- Aprender que es un Plugin y cual es su función y utilidad.
- Aprender nuevos métodos del core de JQuery necesarios para la realización de plugins.
- Aprender cuales son las normas para su creación.

Temario:

- QUE SON LOS PLUGINS
- COMO SE CREA UN PLUGIN DE JQUERY
- NORMAS A TENER EN CUENTA EN LA CREACION DE PLUGINS
- MÉTODO EACH() DEL CORE DE JQUERY
- EJEMPLO 1: PLUGIN DE JQUERY
- EJEMPLO 2: PLUGIN CUENTACARACTERES
- COMO GESTIONAR OPCIONES EN LOS PLUGINS
- EJEMPLO 3: TIP CON OPCIONES
- PLUGIN SLIDER
- PLUGINS DE TERCEROS
- PLUGIN PARA ANIMAR COLORES

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

QUE 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 hacer 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 plugin 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.

Voy a poner un ejemplo un poco abstracto para ver si podemos llegar a la idea de cómo es un plugin. Imagina que necesitas que los elementos de la página "bailen" (parpadeen, se muevan, interactúen con el usuario de una manera concreta, o lo que sea que necesites), pues creas una función para hacer eso. Haces que esa función sea un plugin llamado "bailar" y a partir de entonces cualquier elemento de la página que lo desees podrá bailar. Para ello simplemente invocas ese método del objeto jQuery sobre el elemento o elementos que selecciones.

```
//con esto bailan todos los párrafos
$("p").bailar();

//con esto bailan los elementos de la clase "artista"
$(".artista").bailar();

//con esto baila el elemento con id="lola"
$("#lola").bailar();
```

Espero que el ejemplo no haya parecido muy tonto, pero es que los plugins no son nada del otro mundo, son simplemente eso, extensiones del framework para crear cualquier funcionalidad que podamos necesitar en los elementos de la página, por muy especial, o tonta, que sea.

Lo genial de los plugins es que tú podrás utilizar esa funcionalidad en donde desees a partir de ahora, ya que estará perfectamente a tu disposición, siempre que tengas cargado el plugin. Incluso si tu generosidad es tal, la podrás proporcionar a otras personas para que la utilicen en sus desarrollos. 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.

COMO 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.

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();
```

NORMAS A TENER EN CUENTA EN LA CREACION DE PLUGINS

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 plugin 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.

MÉTODO EACH() DEL CORE DE JQUERY

Antes de seguir avanzando vamos a ver él método each() del core de JQuery que vamos a necesitar para el desarrollo de plugins.

El método each(), perteneciente al juego de funciones del core de JQuery, es un método para realizar acciones con todos los elementos que concuerdan con una selección realizada con la función jQuery -también llamada función $()-. Útil porque nos da una manera cómoda de iterar con elementos de la página y hacer cosas con ellos más o menos complejas de una manera rápida y sin utilizar mucho código para definir el bucle.

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 variable "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.

Quiero explicar las bondades de each() con un ejemplo. Por ejemplo, veamos esta línea de código:

```
$("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 failitará esa tarea.

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");
  }
});
```

Con $("p") tengo todos los párrafos. Ahora con each puedo recorrerlos uno a uno. Para cada uno ejecutaremos la función que enviamos como parámetro a each(). En esa función recibo como parámetro una variable "i" que contiene el índice actual sobre el que estoy iterando.

Con if(i%2==0) estoy viendo si el entero del índice "i" es par o impar. Siendo par coloco un color de fondo al elemento y siendo impar coloco otro color de fondo.

Como se puede ver, con la variable "this" tenemos acceso al elemento actual. Pero OJO, que este elemento no es un objeto jQuery, así que no podríamos enviarle métodos del framework jQuery hasta que no lo expandamos con la función jQuery. Así pues, tenemos que hacer $(this) para poder invocar al método css(). Por si no queda claro este punto mirar las diferencias entre estas dos líneas de código:

```
this.css("background-color", "#ccc");
$(this).css("background-color", "#ccc");
```

En la primera línea no estaríamos extendiendo la variable this con las funcionalidades de jQuery, luego no funcionaría.

En la segunda línea, que es la que habíamos utilizado en el script de ejemplo, sí estamos extendiendo la variable "this" con la función jQuery. De ese modo, se puede invocar a cualquier método de jQuery sobre los elementos.

Pueden ver el código completo de este ejemplo en el script funcioneach.html entre los ejemplos de la unidad.

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 aprece 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());
```

```
});
```

El código completo del ejemplo lo encuentran en el script funcioneach2.html entre los ejemplos de la unidad.

Ahora que conocemos las reglas para hacer plugins y que conocemos el método each() de JQuery vamos a comenzar a ver algunos ejemplos sencillos de cómo crear nuestros propios plugins.

EJEMPLO 1: PLUGIN DE JQUERY

Ahora que ya sabemos las reglas básicas para hacer plugins y que conocemos el método each() 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 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.

Tal como lo indica una de las reglas para crear plugins colocaremos el código en un archivo aparte con el nombre jquery.parpadea.js. El ejemplo completo lo encuentran en la carpeta Ejemplo1

EJEMPLO 2: PLUGIN CUENTACARACTERES

El objetivo del ejemplo es la creación de un plugin para conseguir que un campo textarea de formulario informe en todo momento de caracteres que ha escrito el usuario. Es decir, vamos a hacer un método del objeto jQuery que servirá para decirle a los campos de texto textarea que se expandan para convertirse en un textarea que cuente los caracteres en una capa de texto de al lado.

Para hacer los textareas que cuenten caracteres nosotros queremos hacer algo como esto en jQuery.

```
$("textarea").cuentaCaracteres();
```

Con eso queremos conseguir que a todos los textareas del documento HTML les aparezca una información al lado con el número de caracteres que tenga el textarea escrito dentro. Esa cuenta de caracteres debe mostrarse al cargarse la página y actualizarse cuando se escriba algo dentro. Todo eso se automatizará, para que no tengamos que hacer nada, salvo la anterior llamada al plugin.

Entonces, dentro del plugin tenemos que hacer varias cosas.

1. Un bucle con each para recorrer todos los objetos que pueda haber en el objeto jQuery que reciba el método para activar este plugin. Este paso es igual

en todos los plugins.

2. Dentro de ese bucle podemos iterar con todos los elementos que haya en el objeto jQuery, que vamos a suponer son textareas. Vamos a crear un nuevo elemento DIV sobre la macha y vamos a iniciarlo con el texto de la cuenta de caracteres actual del textarea. Ese elemento creado "on the fly" lo añadiremos al cuerpo de la página, justo después de la etiqueta del textarea.
3. Además, haremos un evento, para que cuando el usuario escriba algo en el textarea, el texto con la cuenta de caracteres se actualice automáticamente.

Estos tres pasos serían un resumen del funcionamiento del plugin, cuyo código completo podemos ver a continuación.

```
//creo el plugin cuentaCaracteres
jQuery.fn.cuentaCaracteres = function() {
  //para cada uno de los elementos del objeto jQuery
  this.each(function(){
    //creo una variable elem con el elemento actual, suponemos un textarea
    elem = $(this);
    //creo un elemento DIV sobre la marcha
    var contador = $('<div>Contador caracteres: ' + elem.prop("value").length +
'</div>');
    //inserto el DIV después del elemento textarea
    elem.after(contador);
    //guardo una referencia al elemento DIV en los datos del objeto jQuery
    elem.data("campocontador", contador);

    //creo un evento keyup para este elemento actual
    elem.keyup(function(){
      //creo una variable elem con el elemento actual, suponemos un textarea
      var elem = $(this);
      //recupero el objeto que tiene el elemento DIV contador asociado al textarea
      var campocontador = elem.data("campocontador");
      //modifico el texto del contador, para actualizarlo con el número de caracteres
escritos
      campocontador.text('Contador caracteres: ' + elem.prop("value").length);
    });
  });
  //siempre tengo que devolver this
  return this;
};
```

El código está comentado para que se pueda entender mejor. Quizás nos pueda llamar más la atención la línea donde se utiliza la función jQuery para generar sobre la marcha un objeto jQuery con el campo DIV con el que vamos a seguir la cuenta. Vemos que a través del método prop() accedemos al value del textarea y con la propiedad

length a su longitud en caracteres.

```
var contador = $('<div>Contador caracteres: ' + elem.prop("value").length + '</div>');
```

Luego también puede que nos llame la atención el funcionamiento del método data(), que nos permite almacenar y recuperar datos que se guardarán en el propio objeto jQuery de cada textarea.

Así guardo una referencia al objeto con la capa contador en el textarea, en un dato llamado "campocontador".

```
elem.data("campocontador", contador);
```

Y con este otro código en el evento recupero esa capa, pues luego en el evento tengo que cambiar el contenido con la cuenta de caracteres actualizada.

```
var campocontador = elem.data("campocontador");
```

Una vez creado el plugin, convierto todos los textareas en textareas-contador de caracteres, con este código:

```
$(document).ready(function(){
  $("textarea").cuentaCaracteres();
})
```

Eso es todo, el código completo lo encuentran en la carpeta Ejemplo 2.

COMO GESTIONAR OPCIONES EN LOS PLUGINS

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.

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 como las que hacemos con jQuery UI.

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.

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 siguente 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 "configuracion" todas las opciones del plugin que se va a iniciar.

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();
```

Vamos a ver a continuación un ejemplo de un plugin al cual le pasamos opciones.

EJEMPLO 3: TIP CON OPCIONES

Vamos a hacer un plugin que muestra un tip con una serie de opciones, que nos permitirán configurar de una manera más versátil el comportamiento del 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, 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
  });
});
```

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.

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
```

Para invocar al plugin del tip con opciones, podemos hacerlo de dos maneras posibles: una es con la opciones por defecto y la otra es pasándole las opciones.

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.

En la carpeta Ejemplo 3 encuentran el código completo de este ejemplo.

PLUGIN SLIDER

Vamos a volver sobre el ejercicio hecho en la anterior unidad para convertirlo en un plugin.

En nuestro body tendremos el siguiente código:

```
<div id="imagenes">
   <div><img src="foto1.jpg" id="foto1" /></div>
   <div><img src="foto2.jpg" id="foto2" /></div>
   <div><img src="foto3.jpg" id="foto3" /></div>
</div>
```

Creamos aparte nuestro plugin con el nombre jquery.slider.js entonces en nuestro archivo html tenemos que incluir los dos scripts correspondientes, el jquery.min y el js correspondiente al slider. Nos quedará de esta forma:

```
<script type="text/javascript" src="jquery-1.10.2.min.js"></script>
<script type="text/javascript" src="jquery.slider.js"></script>
```

Y luego llamamos al plugin pasándole como parámetro el div que contiene las imágenes:

```
<script type="text/javascript">
        $(document).ready(function(){
                        $("#imagenes").slider();
        });
</script>
```

El código del plugin es el siguiente:

```
jQuery.fn.slider = function() {

        this.each(function(){
                $('#imagenes div:gt(0)').hide();
                setInterval(function(){
                  $('#imagenes div:first-child').fadeOut(0)
                        .next('div').fadeIn(1000)
```

```
                .end().appendTo('#imagenes');}, 3000);
        });

        return this;
};
```

Vamos a explicar como funciona:

$('#slider div:gt(0)').hide();
Con gt(x) seleccionamos todos los div a partir del número x. En este caso 0 es el primero, así que lo que hacemos con esta línea es esconder
(hide) todas las cajas excepto la primera, que será la imagen inicialmente visible.

setInterval(function(){ [lo que haremos] }, 3000);
Necesitamos reiterar una serie de cosas cada cierto tiempo y eso lo hacemos con setInterval, indicando el tiempo de retardo entre cada serie.

$('#slider div:first-child').fadeOut(0)
Dentro de cada uno de esos intervalos hacemos desaparecer (fadeOut) la primera caja (div:first-child) que haya en la relación de imágenes...

.next('div').fadeIn(1000)
... y hacemos que la siguiente caja (next) aparezca poco a poco (fadeIn).

.end().appendTo('#slider');
Por último tomamos la que hasta ese momento es la primera imagen y la situamos al final (appendTo) de la "lista".

end() resetea el contador de elementos que hicimos avanzar con next(). De esa manera, el primer hijo que antes hicimos desaparecer es el que enviamos al final de la pila y no la imagen que ahora tenemos visible. Para el siguiente ciclo la que hasta este momento era visible (que era la segunda) será la primera y por tanto la que haremos desaparecer.

Referencias sobre las funciones usadas:

- :gt()
- .hide()
- setInterval()
- :first-child
- .fadeOut()
- .fadeIn()
- .end()
- .appendTo()

El código completo del ejemplo se encuentra en la carpeta Plugin slider.

PLUGINS DE TERCEROS

Como dijimos al principio de este apunte 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.

Los desarrolladores de plugins en general ponen a disposición el código de estos para que puedan ser utilizados por cualquiera de nosotros en nuestros desarrollos, es así que existen infinidad de plugins de terceros los cuales podemos bajar y utilizar libremente.

En la carpeta plugins de terceros tienen una cantidad de ellos como ejemplo.

A continuación vemos uno para animar colores basado en el método animate() de JQuery.

Espero encontrar próximamente un plugin desarrollado por ustedes que pueda utilizar! ☺.

PLUGIN PARA ANIMAR COLORES

En jQuery podemos hacer muchos tipos de animaciones a partir del método animate(), que nos sirve para variar de una manera suavizada una gran gama de propiedades CSS. De hecho, ese es uno de los métodos más importantes del día a día en la realización de efectos con jQuery.

Pero con animate() no podemos hacer animaciones de color, es decir, hacer un gradiente suavizado para pasar de un color a otro con una animación. Quizás nunca encuentren un inconveniente en esa carencia del framework, pero si algún día deciden hacer una animación de color, tendréis que teclear bastante código para conseguirlo por vuestra cuenta.

Sin embargo, como en muchas otras ocasiones, los plugins de terceras personas nos pueden ahorrar mucho trabajo y horas de ingeniería. En este caso comentamos una de esas joyitas que nos permitirá hacer animaciones de color directamente con el método animate() que estamos familiarizados.

El plugin de jQuery que vamos a mostraros a continuación se llama Color animation jQuery-plugin y lo puedes encontrar en la ruta: http://www.bitstorm.org/jquery/color-

animation/

Lo cierto es que hay poco que explicar sobre este plugin, pues simplemente se trata de incluirlo en las páginas y a partir de ese momento simplemente utilizar el conocido método animate(), no obstante, hacemos una descripción paso por paso sobre cómo se utilizaría.
Nota: Las propiedades CSS que podrás animar usando este plugin son las siguientes:

- color
- backgroundColor
- borderColor
- borderBottomColor
- borderLeftColor
- borderRightColor
- borderTopColor
- outlineColor

1) Incluir jQuery y el plugin
Comenzamos por incluir los scripts del framework jQuery y del plugin para animación de colores.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.2/jquery.min.js"
type="text/javascript"></script>
<script src="jquery.animate-colors.js" type="text/javascript"></script>
```

2) Crear la animación
Ahora es tan sencillo como invocar a animate() con los parámetros necesarios, que fueron explicados en la unidad 2 del presente módulo.

```
$("h1").animate({
  color: "#f86"
}, 3000);
```

Con esto estamos haciendo que los elementos H1 de la página tengan una animación que durará 3 segundos en la que pasarán del color que tuvieran definido normalmente hasta el color #f86.

Como hemos comprobado, no tiene mucho misterio, pero el efecto puede resultar interesante. Si tenemos un poco de creatividad todavía podemos conseguir efectos un poco más atractivos, como el que vamos a ver a continuación.

Se trata de hacer una animación de color de un fondo (atributo background-color), pero donde estamos utilizando un patrón de imagen que se repite en un mosaico. Al haber un fondo de imagen, da la sensación que la animación se realiza cambiando esa imagen, pero realmente solo está cambiando el color del fondo. Esto lo conseguimos gracias a una imagen de fondo que tiene transparencia.

Veamos el estilo que hemos definido para los elementos H2:

```
h2{
  padding: 30px;
  background-color: #ffc;
  background-image: url("fondo-h2.png");
  color: #009;
}
```

La imagen de fondo que hemos colocado "fondo-h2.png" es parcialmente transparente, para obtener el efecto deseado.

Ahora este pequeño código nos servirá para iluminar y oscurecer el fondo del H2 al hacer clic sobre él.

```
var iluminado = false;
$("h2").click(function(){
  var elem = $(this);
  if(iluminado){
    elem.animate({
      "background-color": "#ffc"
    }, 500);
  }else{
    elem.animate({
      "background-color": "#9f9"
    }, 500);
  }
  iluminado = !iluminado;
})
```

Como se puede comprobar, se ha utilizado una variable "iluminado" para saber cuando el elemento está encendido y cuando apagado. Luego creamos un evento click, para colocar la funcionalidad descrita. Si estaba iluminado, hago una animación del atributo background-color hacia un color y si estaba oscurecido paso el background-color hacia otro color.

El efecto no es nada del otro mundo, pero es bastante versátil y si tienen un bonito fondo con un patrón interesante, más atractivo será el resultado.

La ventaja del plugin que hemos explicado es que no tienen que aprender nada nuevo, sino simplemente incluir su código y estarán en disposición de hacer animaciones de color, como si jQuery siempre lo hubiera soportado en su conocido método animate().