

Edición 2013



Tema 4. Animaciones y efectos jQuery
Alejandro Amat Reina

ÍNDICE

1. OBJETIVOS	2
2. NOTACIÓN JSON	3
3. ANIMACIONES Y EFECTOS	6
MOSTRAR U OCULTAR ELEMENTOS.....	6
MOSTRAR U OCULTAR ELEMENTOS CON ANIMACIÓN.....	7
ANIMACIONES PREDEFINIDAS.....	9
ANIMACIONES PERSONALIZADAS.....	10
ENCOLAR ANIMACIONES.....	13
DETENER ANIMACIONES.....	14
PROBLEMAS DE LAS ANIMACIONES.....	15
4. EJERCICIOS	19
EJERCICIO 1.....	19
EJERCICIO 2.....	19
EJERCICIO 3.....	20
EJERCICIO 4.....	20
5. ÍNDICE DE EJEMPLOS Y TABLAS	22
EJEMPLOS.....	22
TABLAS.....	22

1. Objetivos

En este cuarto tema se pretenden conseguir los siguientes objetivos:

- ✚ Comprender la notación JSON.
- ✚ Ser capaz de mostrar u ocultar elementos utilizando efectos.
- ✚ Realizar animaciones personalizadas.
- ✚ Aprender a iniciar y finalizar animaciones por medio de eventos.
- ✚ Resolver problemas que pueden surgir al realizar animaciones.

2. Notación JSON

JSON (*javascript Object Notation*) es un formato sencillo para el intercambio de información. El formato JSON permite representar estructuras de datos (arrays) y objetos (arrays asociativos) en forma de texto. La notación de objetos mediante JSON es una de las características principales de javascript y es un mecanismo definido en los fundamentos básicos del lenguaje, por lo tanto, es una característica ampliamente usada en jQuery.

En los últimos años, JSON se ha convertido en una alternativa al formato XML, ya que es más fácil de leer y escribir, además de ser mucho más conciso. No obstante, XML es superior técnicamente porque es un lenguaje de marcado, mientras que JSON es simplemente un formato para intercambiar datos.

La especificación completa de JSON se puede consultar en [RFC 4627](https://tools.ietf.org/html/rfc4627) y su tipo MIME oficial es `application/json`.

La notación tradicional de los arrays es tediosa cuando existen muchos elementos:

```
var modulos = new Array();
modulos[0] = "Lector RSS";
modulos[1] = "Gestor email";
modulos[2] = "Agenda";
```

Para crear un array normal mediante JSON, se indican sus valores separados por comas y encerrados entre corchetes. Por lo tanto, el ejemplo anterior se puede reescribir de la siguiente manera utilizando la notación JSON:

```
var modulos = ["Lector RSS", "Gestor email", "Agenda"];
```

Por su parte, la notación tradicional de los arrays asociativos es igual de tediosa que la de los arrays normales:

```
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos["rss"] = "Lector RSS";
modulos.titulos["email"] = "Gestor email";
modulos.titulos["agenda"] = "Agenda";
```

En este caso, se puede utilizar la notación de puntos para abreviar ligeramente su definición:

```
var modulos = new Array();
modulos.titulos = new Array();
modulos.titulos.rss = "Lector RSS";
modulos.titulos.email = "Gestor email";
modulos.titulos.agenda = "Agenda";
```

En cualquier caso, la notación JSON permite definir los arrays asociativos de una forma mucho más concisa. De hecho, el ejemplo anterior se puede reescribir de la siguiente manera utilizando la notación JSON:

```
modulos.titulos =  
{ rss : "Lector RSS", email: "Gestor email", agenda: "Agenda" };
```

La notación JSON para los arrays asociativos se compone de tres partes:

1. Los contenidos del array asociativo se encierran entre llaves ({ y })
2. Los elementos del array se separan mediante una coma (,)
3. La clave y el valor de cada elemento se separan mediante dos puntos (:)

Si la clave no contiene espacios en blanco, es posible prescindir de las comillas que encierran sus contenidos. Sin embargo, las comillas son obligatorias cuando las claves pueden contener espacios en blanco. Si en el ejemplo anterior cambiamos las claves por valores y viceversa:

```
modulos.titulos =  
{ "Lector RSS": "rss", "Gestor email": "email", "Agenda": "agenda" };
```

Como javascript ignora los espacios en blanco sobrantes, es posible reordenar las claves y valores para que se muestren más claramente en el código fuente de la aplicación. El ejemplo anterior se puede rehacer de la siguiente manera añadiendo nuevas líneas para separar los elementos y añadiendo espacios en blanco para tabular las claves y para alinear los valores:

```
modulos.titulos = {  
    rss: "Lector RSS",  
    email: "Gestor email",  
    agenda: "Agenda"  
};
```

Combinando la notación de los arrays simples y asociativos, es posible construir objetos muy complejos de forma sencilla. Con la notación tradicional, un objeto complejo se puede crear de la siguiente manera:

```
var modulo = new Object();  
modulo.titulo = "Lector RSS";  
modulo.objetoInicial = new Object();  
modulo.objetoInicial.estado = 1;  
modulo.objetoInicial.publico = 0;  
modulo.objetoInicial.nombre = "Modulo_RSS";  
modulo.objetoInicial.datos = new Object();
```

Utilizando JSON, es posible reescribir el ejemplo anterior de forma mucho más concisa:

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

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

Como veremos a lo largo de este tema y de los siguientes, el uso de JSON dentro de la librería jQuery y en algunos de los plugins asociados, es muy habitual. Además, como veremos en el tema 6, la notación JSON se utiliza mucho para el intercambio de datos entre el servidor y la aplicación cliente a través de Ajax.

3. Animaciones y efectos

Una de los principales usos que se le ha dado a javascript desde sus orígenes, han sido las animaciones. Pero por animación, no sólo entendemos mover elementos dentro de la página, también podemos cambiar su visibilidad, alterar sus propiedades, cambiar el tamaño, etc. Podemos hacer esto, a través de un temporizador o reaccionando a un evento.

Con la aparición de Flash, el uso de javascript para la realización de animaciones, ha ido decreciendo, sin embargo, gracias a jQuery, las cosas han cambiado y el uso de animaciones Web a través de javascript, es mucho más predominante en la actualidad.

En este punto, veremos algunos de los atajos que nos ofrece jQuery para animar componentes en nuestras aplicaciones Web.

Mostrar u ocultar elementos

Es muy habitual, mostrar u ocultar un elemento como reacción a un determinado evento. En css tenemos dos propiedades que nos permiten esta funcionalidad:

- ✚ La propiedad `visibility`: si le damos el valor `hidden`, ocultará el elemento y si le damos el valor `visible` lo mostrará.
- ✚ La propiedad `display`: si le damos el valor `none`, ocultará el elemento y si le damos el valor `block` o `inline`, lo mostrará.

Aunque, puede parecer que las dos propiedades hacen lo mismo, hay que señalar que, la aplicación de una u otra tendrá un efecto diferente en nuestra página. Si ocultamos el elemento mediante la propiedad `visibility`, el elemento no será visible, pero el espacio que ocupa permanecerá reservado en la página. Sin embargo, al ocultar con la propiedad `display`, el elemento desaparecerá totalmente de la página y, el lugar donde se encontraba, será ocupado por el resto de elementos.

Es preferible utilizar los métodos que nos ofrece jQuery, para ocultar o mostrar elementos.

Método	Funcionamiento
<code>.hide()</code>	Sin parámetros, este método oculta los elementos seleccionados
<code>.show()</code>	Sin parámetros, este método muestra los elementos seleccionados
<code>.toggle()</code>	Sin parámetros, este método muestra u oculta los elementos seleccionados. Si están ocultos,

se mostrarán y, si están visibles, se ocultarán.

Tabla 1: Efectos básicos en jQuery

Cuando utilizamos estos métodos sin pasar ningún parámetro, el efecto obtenido será el mismo que cuando modificamos el valor de la propiedad `display` para ocultar o mostrar un elemento.

```
<!DOCTYPE html>
<html>
  <head>
    <title>efectos básicos</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("#show").click(function ( event )
        {
          event.preventDefault();
          $("p").show();
        });
        $("#hide").click(function ( event )
        {
          event.preventDefault();
          $("p").hide();
        });
        $("#toggle").click(function ( event )
        {
          event.preventDefault();
          $("p").toggle();
        });
      });
    </script>
  </head>
  <body>
    <p>Párrafo</p>
    <a href="#" id="show">Show</a> | <a href="#" id="hide">Hide</a> | <a
href="#" id="toggle">Toggle</a>
  </body>
</html>
```

Ejemplo 1: [efectosBasicos.html](#)

En el ejemplo, se puede observar cómo, al ocultar el párrafo, los enlaces ocupan el lugar que ha quedado libre. Además, el párrafo, simplemente, se oculta o se muestra inmediatamente, pero no se realiza ningún tipo de animación.

Mostrar u ocultar elementos con animación

Los métodos anteriores pueden recibir por parámetros una duración y/o una función, de forma, que se convertirán en métodos de animación:

- 🚦 Duración: El parámetro duración indicará el número de milisegundos que durará la animación.

- Completado: El parámetro completado será una función que se ejecutará cuando la animación haya terminado.

Estos métodos animarán el ancho, el alto y la opacidad de los elementos que ocultan o muestran.

```
<!DOCTYPE html>
<html>
  <head>
    <title>efectos básicos animados</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style>
      p { background:yellow; width: 300px; height:300px;}
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("#show").click(function ( event )
        {
          event.preventDefault();
          $("p").show(600, function()
          {
            alert("mostrado");
          });
        });
        $("#hide").click(function ( event )
        {
          event.preventDefault();
          $("p").hide(600, function()
          {
            alert("oculto");
          });
        });
        $("#toggle").click(function ( event )
        {
          event.preventDefault();
          $("p").toggle(600, function()
          {
            alert("cambiado");
          });
        });
      });
    </script>
  </head>
  <body>
    <p>Hola</p>
    <a href="#" id="show">Show</a> | <a href="#" id="hide">Hide</a> | <a
href="#" id="toggle">Toggle</a>
  </body>
</html>
```

Ejemplo 2: [efectosBasicosAnimados.html](#)

En el ejemplo, podemos ver como se produce una animación para cambiar la visibilidad del párrafo y, al terminar la animación, se ejecuta la función que le pasamos.

Animaciones predefinidas

Además de los métodos `.show()` y `.hide()` con animación, vistos en el punto anterior, jQuery dispone de otros métodos que nos van a permitir mostrar u ocultar elementos, realizando para ello animaciones predefinidas. Los parámetros `duración` y `completado` de los siguientes métodos, tienen la misma finalidad que en los métodos anteriores.

Método	Funcionamiento
<code>.fadeIn([duración], [completado])</code>	Muestra los elementos seleccionados animando su opacidad. La animación irá de totalmente transparentes a totalmente opacos.
<code>.fadeOut([duración], [completado])</code>	Oculto los elementos seleccionados animando su opacidad. La animación irá de totalmente opacos a totalmente transparentes.
<code>.fadeToggle([duración], [completado])</code>	Muestra u oculta los elementos seleccionados animando su opacidad. Si están ocultos, se mostrarán y, si están visibles, se ocultarán.
<code>.fadeTo(duración, opacidad, [completado])</code>	Ajusta la opacidad de los elementos seleccionados. El parámetro <code>opacidad</code> , será un número entre 0 y 1 que indicará el porcentaje de opacidad al que queremos ajustar los elementos.
<code>.slideDown([duración], [completado])</code>	Muestra los elementos seleccionados con una animación de deslizamiento (va cambiando el alto de los elementos).
<code>.slideUp([duración], [completado])</code>	Oculto los elementos seleccionados con una animación de deslizamiento.
<code>.slideToggle([duración], [completado])</code>	Muestra u oculta los elementos seleccionados con una animación de deslizamiento. Si están ocultos, se mostrarán y, si están visibles, se ocultarán.

Tabla 2: Métodos con animaciones predefinidas

A continuación, veremos un ejemplo de uso de los métodos anteriores.

```
<!DOCTYPE html>
<html>
  <head>
    <title>efectos predefinidos fade</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("#incrementa").click(function ( event )
        {
          event.preventDefault();
          $("p").fadeOut(600, function()
          {
            var valor = parseInt($(this).text());
            valor++;
            $(this).html(valor);
            $(this).fadeIn(600);
          });
        });
      });
    </script>
  </head>
  <body>
    <p>10</p>
    <a href="#" id="incrementa">Incrementa</a>
  </body>
```

</html>

[Ejemplo 3: efectosPredefinidosFade.html](#)

En el ejemplo, estamos incrementando el valor del número contenido en el párrafo, para mejorar el efecto, estamos realizando dos animaciones. En primer lugar, ocultamos el número con el método `.fadeOut()`, realizando una animación de 600 ms de duración. Cuando la animación acaba, se invoca a una función que incrementa el número mientras está oculto, y vuelve a mostrarlo mediante el método `.fadeIn()` con una animación de 600 ms.

Como podemos ver en el ejemplo, desde la función de callback que le pasamos al método de animación, podemos acceder al elemento que estamos animando, utilizando el objeto `$(this)`.

Animaciones personalizadas


Cuando estamos realizando animaciones sencillas, que, simplemente, muestran u ocultan elementos, los métodos vistos en el punto anterior son más que suficientes, pero para realizar animaciones más complejas, en las que podamos animar la propiedad css que queramos, o incluso varias propiedades css, utilizamos el método `.animate()`.

Cualquier propiedad que utilice un valor numérico es susceptible de ser animada mediante este método. Por ejemplo, la posición (`left`, `right`, `top` o `bottom`), el ancho (`width`), el alto (`height`), etc.

El método `.animate()` tiene la siguiente definición:

```
$(selector).animate(propiedades, [duración], [easing], [completado])
```

Veamos el significado de los parámetros:

 **Propiedades:** Será un objeto de propiedades css y los valores hacia los que la animación se mueve. Ejemplos de este tipo de objetos:

- `{width: 0}`: Se realiza una animación que irá desde el ancho actual hasta 0.
- `{left: "+=100"}`: Cuando el valor de una propiedad se pasa como string, precedido por los caracteres `+=` o `-=`, dicho valor será relativo al valor actual de la propiedad. Por lo tanto, en este caso, se incrementará el valor de la propiedad `left` en 100 píxeles.
- `{height: "toggle"}`: Cuando utilizamos `"toggle"` como valor de una propiedad, si el elemento está visible se ocultará, animando para ello

la propiedad height y, si está oculto, se mostrará animando la misma propiedad.

- {opacity: 0.25, left: "+=50", height: "toggle"}: Podemos animar un elemento utilizando varias propiedades. En este caso, la opacidad del elemento se configurará en un 25%, será desplazado 50 píxeles a la izquierda y se mostrará u ocultara de forma alterna.

🚦 Easing: Este parámetro representará el nombre de una función de aceleración. Una función de aceleración especifica la velocidad, a la cual, la animación progresa en diferentes puntos dentro de la animación. La librería jQuery dispone de dos funciones implementadas:

- swing: la utilizada por defecto.
- linear: el progreso se realiza a un ritmo constante.

Con el uso de plug-ins, como jQuery UI, tenemos disponibles más funciones de aceleración.

A continuación, veremos algunos ejemplos.

```
<!DOCTYPE html>
<html>
  <head>
    <title>animación personalizada</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
      div { position: relative; width:10px; height:10px; background:blue; }
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $(document).click( function(event)
        {
          $( "div#box" ).animate({ left: '+=100' }, 200);
        });
      });
    </script>
  </head>
  <body>
    <div id="box">
    </div>
  </body>
</html>
```

Ejemplo 4: [animacionPersonalizada.html](#)

En este ejemplo, el <div> se desplaza 100 píxeles a la izquierda cada vez que hacemos click con el ratón en cualquier parte del documento.

```
<!DOCTYPE html>
<html>
```

```

<head>
  <title>animación de varios atributos</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <style type="text/css">
    img#alien { position: relative; left:0; }
  </style>
  <script src="http://code.jquery.com/jquery-1.9.0.min.js"
    type="text/javascript"></script>
  <script type="text/javascript">
    $(function(){
      $(document).click( function( event ){
        $( "#alien" ).animate({
          opacity: 0,
          width: 10,
          height: 10,
          left: '+=1000px'
        }, 2000 , function () {
          $("body").append("<p>BYE!!!</p>");
        });
      });
    });
  </script>
</head>
<body>
  
</body>
</html>

```

Ejemplo 5: [animacionVariosAtributos.html](#)

En este otro ejemplo, se están animando varias propiedades simultáneamente, en concreto, se realiza una animación que cambia la opacidad, el tamaño y la posición de la imagen #alien. La animación tiene una duración de 2000 ms y, al terminar, se ejecuta una función que añade un párrafo al documento.

```

<!DOCTYPE html>
<html>
  <head>
    <title>animación con toggle</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
      img#alien { position: relative; left:0; }
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function(){
        $(document).click( function( event ){
          $( "#alien" ).animate({
            height: "toggle",
            left: '+=100'
          }, 300);
        });
      });
    </script>
  </head>
  <body>
    

```

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

Aquí vemos como se animan dos propiedades: la altura (height) y la posición (left). Si el elemento está visible, al hacer click con el ratón sobre el documento, se animará disminuyendo su altura hasta quedar invisible y, si no está visible, hará lo contrario.

Encolar animaciones

jQuery dispone de una cola de efectos, de manera que cada vez que iniciamos una animación, podemos indicar si queremos que la animación entre en la cola para ser ejecutada cuando le llegue su turno.

Todos los métodos de animación vistos, disponen de una definición alternativa, que les permite recibir un objeto de propiedades. Estas propiedades tendrán unos valores por defecto que podremos cambiar a nuestro antojo (ver la documentación de los distintos métodos en la página <http://api.jquery.com/>). El nombre de la propiedad que nos interesa en este punto es `queue`; será un valor booleano que indicará si queremos que la animación sea encolada o no; por defecto, esta propiedad tiene el valor `true`. Veamos un ejemplo:

```
<!DOCTYPE html>
<html>
<head>
  <title>animación encolada</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <style>
    div { background:#bca; width:200px; height:1.1em; text-align:center;
          border:2px solid green; margin:3px; font-size:14px; }
    button { font-size:14px; }
  </style>
  <script src="http://code.jquery.com/jquery-1.9.0.min.js"
          type="text/javascript"></script>
  <script type="text/javascript">
    $(function ()
    {
      $( "#go1" ).click(function()
      {
        $( "#block1" ) .animate({width:"90%"}, {queue: false, duration:3000})
                          .animate({ fontSize: "24px" }, 1500 )
                          .animate({ borderRightWidth: "15px" }, 1500 );
      });
      $( "#go2" ).click(function()
      {
        $( "#block2" ) .animate({ width: "90%" }, 1000 )
                          .animate({ fontSize: "24px" }, 1000 )
                          .animate({ borderLeftWidth: "15px" }, 1000 );
      });
      $( "#go3" ).click(function()
      {
```

```
$( "#go1" ).add( "#go2" ).click();
});
$( "#go4" ).click(function()
{
    $( "div" ).css({ width: "", fontSize: "", borderWidth: "" });
});
});
</script>
</head>
<body>
    <button id="go1">&raquo; Animate Block1</button>
    <button id="go2">&raquo; Animate Block2</button>
    <button id="go3">&raquo; Animate Both</button>
    <button id="go4">&raquo; Reset</button>
    <div id="block1">Block1</div>
    <div id="block2">Block2</div>
</body>
</html>
```

Ejemplo 6: [animacionEncolada.html](#)



En el ejemplo, tenemos dos bloques, sobre los cuales, se aplicarán tres animaciones sucesivas. El primer bloque, se animará sin introducir las animaciones en la cola, por lo tanto, se ejecutarán todas simultáneamente. Sin embargo, el segundo bloque, sí que introducirá las animaciones en la cola, así que las animaciones se irán realizando una tras otra.

Detener animaciones

Cualquier animación puede ser detenida de dos formas: utilizando el método `.finish()` o utilizando el método `.stop()`.

El método `.finish()` **elimina y completa** todas las animaciones existentes en los elementos seleccionados. Cuando el método `.finish()` es llamado, la animación que está actualmente en ejecución y todas las que estuvieran pendientes se terminarán y sus propiedades css tomarán los valores de destino especificados.

El método `.stop([limpiarCola], [saltarAlFinal])` detiene la animación que se está ejecutando actualmente en el conjunto de elementos seleccionados. Puede recibir dos parámetros:

-  `limpiarCola`: valor booleano que indica si se deben eliminar también las animaciones encoladas.
-  `saltarAlFinal`: valor booleano que indica si se debe completar la actual animación inmediatamente, es decir, si las propiedades css que se estuvieran animando deben tomar sus valores de destino. Si se pasa true y

la animación tiene una función callback de completado, esta será llamada inmediatamente.

El método `.finish()` es similar al método `.stop(true, true)`, ya que, en ambos, se eliminarán todas las animaciones de la cola y se establecerán las propiedades css de la animación actual a su valor de destino. Sin embargo, la diferencia en este caso sería, que con el método `.finish()`, también se establecerían a su valor de destino, las propiedades css de todas las animaciones de la cola.

```
<!DOCTYPE html>
<html>
  <head>
    <title>parar animación</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
      img#alien { position: relative; left:0; }
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $("a").mouseover(function(evento)
        {
          $("#alien").animate({ left: '+=1000' }, 1000);
        }).mouseout(function()
        {
          $("#alien").stop(true, false);
        });
      });
    </script>
  </head>
  <body>
    
    <a href="">mueve</a>
    <p>Ponte sobre el enlace para mover la caja. Sal para pararla</p>
  </body>
</html>
```

Ejemplo 7: [pararAnimacion.html](#)

En este ejemplo se inicia una animación en el evento `mouseover` del enlace y se detiene en el evento `mouseout`.

Problemas de las animaciones

Cuando realizamos una animación, hemos de tener en cuenta, que tendrá una determinada duración, por lo tanto, mientras la animación se realiza el código javascript seguirá su curso. Por ejemplo, si mientras la animación está en marcha se produce un evento sobre un elemento que tiene un manejador, el código del manejador será ejecutado. Si, además, desde este manejador se accede a la

propiedad que se está animando, podemos encontrarnos con problemas, ya que el valor de la propiedad será impredecible.

Veamos el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>animación con problemas</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
      div { position: relative; left: 0px; width:100px; height:100px;
background:blue; }
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
      type="text/javascript"></script>
    <script type="text/javascript">
      $(function()
      {
        $(document).click( function(event)
        {
          $( "div#box" ).animate({ left: '+=100' }, 200);
          $( "div#box" ).html($( "div#box" ).css("left"));
        });
      });
    </script>
  </head>
  <body>
    <div id="box">
    </div>
  </body>
</html>
```

Ejemplo 8: [animacionConProblemas.html](#)

Al hacer click con el botón del ratón en cualquier parte del documento, el <div> con id box se desplazará 100 píxeles a la izquierda con una animación que durará 200 segundos, además, también se escribirá dentro del <div> el valor de su propiedad left. Si probamos el ejemplo, vemos que al hacer un click funciona perfectamente, pero si hacemos varios click rápidamente, observamos que el valor de la propiedad left que se va mostrando en el <div>, no va de 100 en 100 como podríamos esperar, sino que va tomando valores intermedios, incluso, con decimales.

Esto es así, porque estamos accediendo a la propiedad que estamos animando a mientras la animación está en marcha. Tenemos dos formas de solucionar este tipo de problemas: usando una variable centinela o usando una función de completado.

```
<!DOCTYPE html>
<html>
  <head>
    <title>animación solucionada función completado</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
```

```

    div { position: relative; left: 0px; width:100px; height:100px;
background:blue; }
</style>
<script src="http://code.jquery.com/jquery-1.9.0.min.js"
        type="text/javascript"></script>
<script type="text/javascript">
    $(function()
    {
        $(document).click( function(event)
        {
            $( "div#box" ).animate({ left: '+=100' }, 200, function ()
            {
                $( "div#box" ).html($( "div#box" ).css("left"));
            });
        });
    });
</script>
</head>
<body>
    <div id="box">
    </div>
</body>
</html>

```

Ejemplo 9: [animacionSolucionadaFuncionCompletado.html](#)

En este caso, sólo accedemos a la propiedad `left` cuando estamos seguros de que la animación ha terminado, es decir, en la función completado.

Otra posibilidad, es impedir la ejecución de la animación si la animación ya está iniciada. Utilizamos, para ello, una variable centinela.

```

<!DOCTYPE html>
<html>
  <head>
    <title>animación solucionada centinela</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <style type="text/css">
      div { position: relative; left: 0px; width:100px; height:100px;
background:blue; }
    </style>
    <script src="http://code.jquery.com/jquery-1.9.0.min.js"
            type="text/javascript"></script>
    <script type="text/javascript">
      var animando = false;
      $(function()
      {
        $(document).click( function(event)
        {
          if (animando == false)
          {
            animando = true;
            $( "div#box" ).animate({ left: '+=100' }, 200, function ()
            {
              animando = false;
            });
            $( "div#box" ).html($( "div#box" ).css("left"));
          }
        });
      });
    </script>
  </head>
  <body>
    <div id="box">
    </div>
  </body>
</html>

```

```
});  
</script>  
</head>  
<body>  
  <div id="box">  
  </div>  
</body>  
</html>  
Ejemplo 10: animacionSolucionadaCentinela.html
```

La diferencia entre este método y el anterior, es que en el anterior, siempre que hacemos un click sobre el documento, se produce un desplazamiento, aunque el valor de `left` en el `<div>` se actualizará al final del mismo. Sin embargo, con este método, algunos de los clicks del ratón se perderán y no tendrán efecto sobre el `<div>`.

En este punto, se ha explicado una posible situación problemática que puede surgir cuando realizamos animaciones, la casuística es muy amplia y, según el caso, las soluciones que deberemos adoptar serán variadas, pero podemos tomar como norma que no debemos modificar valores de elementos que son susceptibles de ser animados, a no ser que estemos seguros de que la animación está completada o no está en ejecución.

4. Ejercicios

Hemos hablado durante este tema sobre los efectos y las animaciones. Haremos ahora una serie de mejoras en nuestra aplicación para practicar los conceptos estudiados.

Ejercicio 1

Cuando el carrito este vacío, nuestra barra de navegación va a estar oculta, así que, en nuestro `document.ready` ocultaremos los botones comprar, siguiente, anterior y vaciar. Después, controlaremos la visibilidad de los mismos de la siguiente forma:

- ✚ En el momento que se introduzca un artículo en el carrito, se mostrarán los botones comprar y vaciar. Si el carrito se vuelve a quedar vacío, se volverán a ocultar.
- ✚ En el momento que hayan más de cuatro artículos en el carrito, los botones anterior y siguiente se mostrarán. Si en el carrito vuelven a haber menos de cuatro artículos porque se elimine alguno de ellos, se volverán a ocultar.

Ejercicio 2

A continuación, introduciremos algunos efectos cuando se pulse sobre un artículo para añadirlo al carrito:

- ✚ Al añadir la copia del artículo al carrito, primero la ocultaremos con `.hide()`, después, la añadiremos y, finalmente, animaremos la visualización de la misma.
- ✚ Para realizar la animación, utilizaremos el método `.animate()` y animaremos la propiedad `width` con el valor `"toggle"`. El efecto resultante será que el artículo se despliega dentro del carrito, haciéndose cada vez más ancho y desplazando el resto de artículos.
- ✚ Animaremos la actualización del stock disponible, ya sea para incrementarlo o para decrementarlo. En este caso, en lugar de animar la propiedad `width` con el método `.animate()`, vamos a utilizar el método `.fadeIn()`.
- ✚ Por último, animaremos la actualización del número de ítems comprados y el precio total de la compra. El mecanismo será el mismo que en los dos

puntos anteriores. En este caso, utilizaremos también el método `.fadeIn()` para volver a mostrar los valores actualizados.

✚ Cuando sea necesario se mostrarán los botones de navegación.

Ejercicio 3

Vamos a animar también la eliminación de un artículo del carrito. El comportamiento debe ser el siguiente:

✚ Al pulsar sobre el botón `delete` del artículo a eliminar, se iniciará una animación `.fadeOut()`.

✚ Una vez la animación haya terminado, se actualizará el stock disponible, el precio total de la compra y el número de ítems comprados, con las animaciones indicadas en el ejercicio anterior.

✚ También se ocultarán los botones de navegación, en caso de ser necesario.

Todos los efectos que introduzcamos para resolver los ejercicios tendrán una duración de 600 ms, excepto los de `.show()` y `.hide()` que no tendrán duración.

Ejercicio 4

Para finalizar, vamos a cambiar el comportamiento de los botones de navegación siguiente y anterior. Como ya habréis observado, es algo tedioso tener que estar haciendo varios clicks con el ratón sobre los botones de navegación para ir desplazándonos por los artículos del carrito. Sería mucho más cómodo, poder desplazarnos por él, simplemente, situando el ratón encima del botón correspondiente y parar el desplazamiento, sacando el ratón del mismo.

Para implementar este comportamiento tendremos que sustituir el manejador de evento `click` de los botones siguiente y anterior, por dos manejadores: uno para el evento `mouseover` y otro para el evento `mouseout`. En el manejador del evento `mouseover`, iniciaremos una animación que incrementará el valor de la propiedad `left` hasta llegar al borde izquierdo o derecho del carrito (según estemos en el botón siguiente o anterior). En el manejador de evento `mouseout` detendremos la animación.

Una cosa a tener en cuenta es la duración que le demos a la animación, ya que, si la duración es corta y el desplazamiento largo (hay muchos artículos), la animación puede ser demasiado rápida, pero si la duración es larga y el desplazamiento corto (hay pocos artículos), la animación puede ser demasiado lenta.

Una posible solución es tener una variable global en la que guardamos la duración de la animación. Inicialmente la duración será 0, pero a medida que vamos añadiendo artículos al carrito, podemos ir incrementando su valor, por ejemplo, de 500 ms en 500 ms. Evidentemente, al eliminar artículos del carrito tendremos que disminuir la duración.

Con esto, terminaríamos el cuarto tema, en el aula virtual, encontraréis un vídeo en el que se explica cómo tiene que quedar nuestra aplicación después de realizar los ejercicios indicados.

5. Índice de ejemplos y tablas

Ejemplos

EJEMPLO 1: EFECTOSBASICOS.HTML.....	7
EJEMPLO 2: EFECTOSBASICOSANIMADOS.HTML	8
EJEMPLO 3: EFECTOSPREDEFINIDOSFADE.HTML	10
EJEMPLO 4: ANIMACIONPERSONALIZADA.HTML.....	11
EJEMPLO 5: ANIMACIONVARIOSATRIBUTOS.HTML	12
EJEMPLO 6: ANIMACIONENCOLADA.HTML.....	14
EJEMPLO 7: PARARANIMACION.HTML	15
EJEMPLO 8: ANIMACIONCONPROBLEMAS.HTML.....	16
EJEMPLO 9: ANIMACIONSOLUCIONADAFUNCIONCOMPLETADO.HTML	17
EJEMPLO 10: ANIMACIONSOLUCIONADACENTINELA.HTML	18

Tablas

TABLA 1: EFECTOS BÁSICOS EN JQUERY	7
TABLA 2: MÉTODOS CON ANIMACIONES PREDEFINIDAS	9