Temporizadores

1-setInterval()

Este método es utilizado para ejecutar repetidamente una función en un intervalo establecido.
El formato de este método es:

```
window.setInterval("functionName()",tiempo);
```

El primer parámetro (“functionName()”) es el nombre de la función que desea ejecutar.
Observa que el nombre de la función está entre comillas. Es tratado como una cadena para evitar que se ejecute de inmediato.
El segundo parámetro (tiempo) es la cantidad de la demora en milisegundos, entre cada vez que la función se ejecuta (1 minuto = 60000 milisegundos).
Esto es útil en la animación, para la rotación de imágenes en una galería o tal vez refrescar la pantalla.
Por ejemplo, para volver a cargar una página después de un intervalo de 10 minutos (600.000 milisegundos), prueba este script:

```
function reFresh() {

location.reload(true)

}

window.setInterval("reFresh()",600000);
```

2- clearInterval()

Este método se utiliza para detener el bucle cronometrado que se inició con el método setInterval () anterior.
El formato es:

```
window.clearInterval(varName);
```

Con el fin de utilizarlo, el bucle debe ser asignado a una variable.
Volvamos a nuestro script actualiza la página anterior.
Sólo tenemos que añadir la palabra reservada var delante del bucle setInterval()

```
function reFresh()
{

window.open(location.reload(true))


}
var repeticion = window.setInterval("reFresh()",600000);
```

Entonces podríamos crear un botón para detener la actualización de la página.

```
<form>
<input type="Button" value="Parar Refresco
Página"onclick="window.clearInterval(repeticion);">
</form>
```

3 - setTimeout()

Este método se utiliza para ejecutar una función de JavaScript después de una determinada cantidad de tiempo. (El nombre es un poco engañoso. No se detiene el script para un período de tiempo, sino que espera un período de tiempo antes de empezar.)
A diferencia del método setInterval (), sólo se ejecuta una vez.
El formato es:

```
window.setTimeout("functionName()",tiempo);
```

Jquery

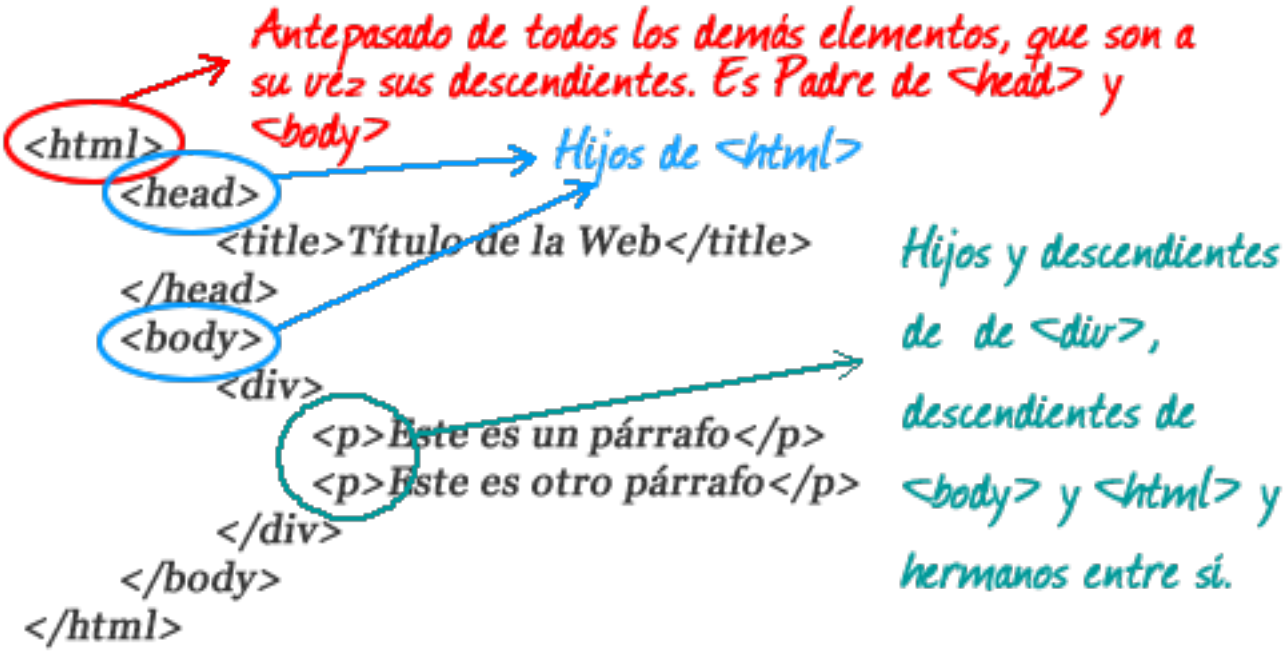
Selectores

Por un *selector* entendemos en jQuery lo mismo que en CSS: una forma de permitirnos elegir un elemento (o varios) entre todos los que tenemos en nuestro documento HTML. **¿Para qué?** Para luego poder aplicar sobre los elementos seleccionados diversas funciones.
Es decir, **jQuery** utiliza el poder de los **selectores** para acceder de una manera rápida y sencilla a un elemento o grupo de elementos del DOM (Document Object Model) **y luego poder aplicar sobre los mismos cualquier tipo de instrucción, evento, animación, etc…**. Por ejemplo, para aplicar la clase “enlace” a todos los elementos “a” que se encuentren dentro de un elemento “p”. Haríamos:

```
$('p a').addClass('enlace');
```

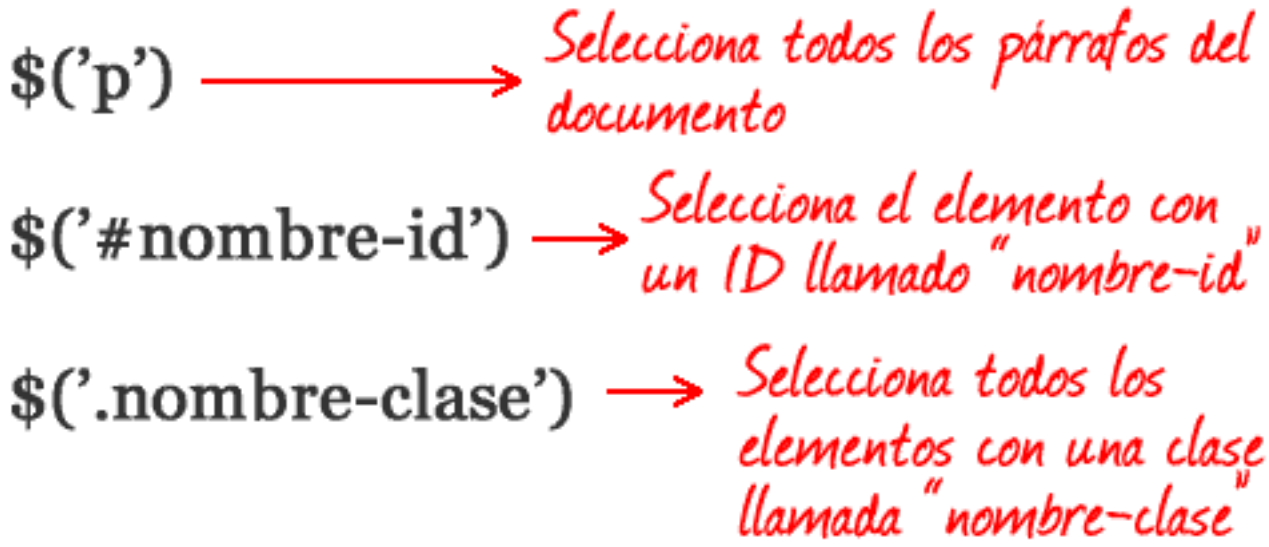
Pero vamos a empezar desde el principio. En este artículo vamos a ver cómo crear los selectores más importantes de jQuery.
Antes de nada es importante saber que el DOM (Document Object Model,

el esqueleto de nuestra página web) utiliza una estructura de árbol para definir las relaciones entre sus elementos, en la que tenemos padres, hijos, etc… Por ejemplo:



$('selector')

No importa qué tipo de selector usemos en jQuery: siempre comenzaremos con ***$()***. Prácticamente todo lo que se pueda usar en CSS se puede también incluir entre esos paréntesis de esta forma ***$('selectores')***. De tal manera que por ejemplo podríamos hacer:



Selectores CSS

jQuery soporta prácticamente todos los **selectores de CSS**, con la ventaja de que podemos utilizar selectores de CSS3 que funcionen con Internet Explorer 6 gracias a jQuery. Lo vemos mejor con estos ejemplos:

$('div') Selecciona todos los DIV del documento

$('a') Selecciona todos los '< a >' del documento

$('p a') Selecciona todos los '<a>' descendientes de un '< p >'

$('p, a') Selecciona todos los '<p>' y todos los '< a >' del documento

$('li.nombreClase') Selecciona todos los '< li >' con clase 'nombreClase'

$('fieldset a') Selecciona todos los '< a >' dentro de '< fieldset >'

$('li>p') Selecciona todos los '< p >' hijos directos de '< li >'

$('h1+p') Selecciona todos los '< p >' inmediatamente precedidos por un '<h1>' hermano

$('div~p') Selecciona todos los 'div' precedidos por un elemento '< p >'

$('p:has(b)') Selecciona todos los '<p>' que contienen un elemento '< b >'

$('div.nombreClase') Selecciona todos los 'div' con la clase 'nombreClase'

$('.nombreClase') Selecciona todos los elementos con la clase 'nombreClase'

$('#nombreID') Selecciona el elemento con un id 'nombreID'

$('img[alt]') Selecciona todos los elementos '< img >' con atributo 'alt'

$('button[id*=boton]') Selecciona todos los botones con atributo 'id' que contenga la palabra boton

$('a[href$=.pdf]') Selecciona todos los '< a >' con atributo 'href' acabado en .pdf

$('a[title^=Ir]') Selecciona todos los '< a >' cuyo atributo 'title' comienza por 'Ir'

Selectores propios de jQuery

A la amplia variedad de selectores propios de CSS jQuery añade sus propios selectores. Como característica que les distingue decir que siempre comienzan por dos puntos (:)
Vamos a distinguir entre ellos:

Selectores Posicionales

Estos selectores están basados en las relaciones posicionales entre elementos (como veíamos antes en ejemplo de la estructura del DOM).
Como antes, los vamos a ver a través de ejemplos:

$('p:first') Selecciona el primer elemento '<p>' de la página

$('img[src$=.png]:first') Selecciona el primer <img> de la página que tiene un atributo src acabado en .png

$('p:last') Selecciona el último '<p>' de la página

$('li:first-child') Selecciona todos los '< li >' que son primeros hijos

$('li:last-child') Selecciona todos los '< li >' que son últmos hijos

$('li:only-child') Selecciona todos los '< li >' que sean hijos únicos

$('li:nth-child (3)') Selecciona todos los '< li >' que sean el tercer elemento de su lista

$('tr:nth-child(odd)') Selecciona todos los '<tr >' que sean impares

$('tr:nth-child(even)') Selecciona todos los '<tr>' que sean pares

$('div:nth-child(3n)') Selecciona cada tercer elemento

$('div:nth-child(3n+1)') Selecciona el elemento tras cada tercer

$('p:odd') Selecciona los '<p>' impares

$('p:even') Selecciona los '<p>' pares

$('p:eq(1)') Selecciona el segundo '<p>' - Comienza a contar desde cero

$('p:gt(1)') Selecciona todos los '<p>' excepto los dos primeros

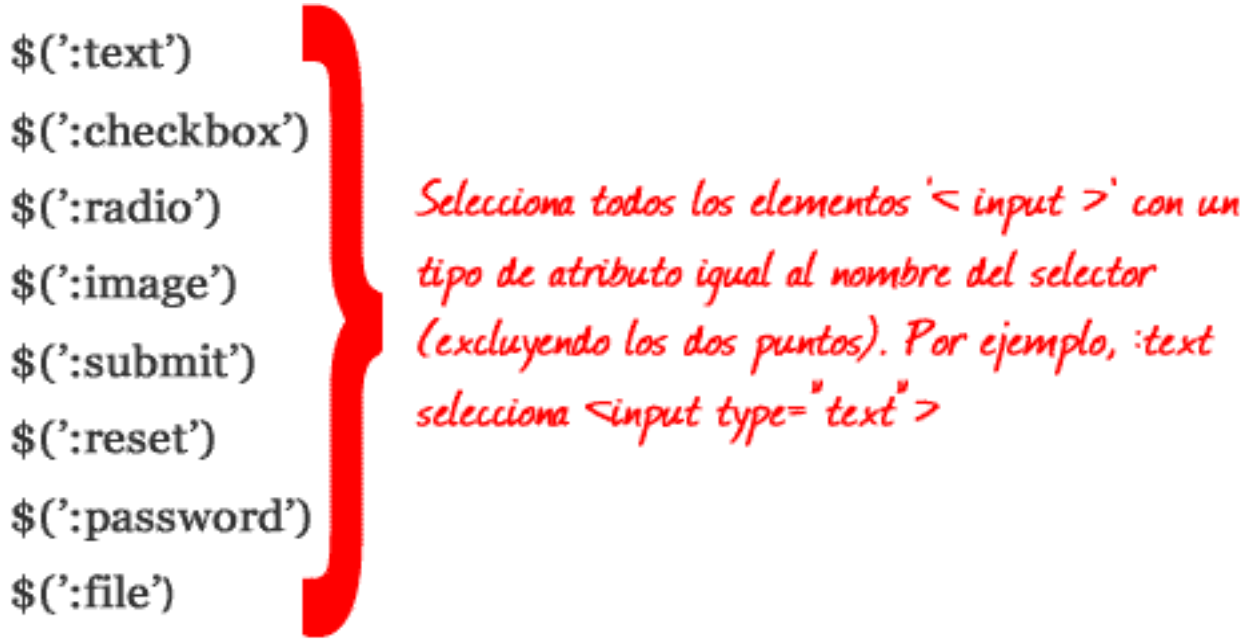
$('p:lt(1)') Selecciona los dos primeros '<p>' - Comienza a contar desde 0

Los selectrores :nth-child comienzan a contar desde uno, mientras que :eq, :gt y :lt comienzan a contar desde cero

Selectores de Formularios

Cuando trabajemos con formularios jQuery nos ofrece una serie de

selectores propios que nos permiten seleccionar de manera sencilla el elemento preciso. Vamos a verlos con ejemplos:



$(':input') Selecciona los elementos input, textarea, select y button.

$(':button') Selecciona los elementos button e input con atributo 'type' igual a 'button'

$(':enabled') Selecciona los elementos del formulario activados

$(':disabled') Selecciona los elementos del formulario desactivados

$(':checked') Selecciona los radio buttons y checkboxes que están pulsados

$(':selected') Elementos de una lista de opciones que estén seleccionados

Estos selectores se pueden combinar, por ejemplo:
$(':radio :checked'), $(':text :disabled'),
$(':button :hidden'), $('select[name=nombre] :selected'),
$('input[name=nombre] :radio :checked')

Vamos a seguir viendo selectores de jQuery. La mayor parte de los selectores que hemos visto hasta ahora nos permitían ir a través y hacia abajo del DOM (Document Object Model) o árbol de nuestro HTML (por ser más gráficos). Hay ocasiones en las que necesitamos acceder de manera más directa a un elemento. En este caso tenemos a nuestra disposición los **métodos transversales**, que nos permiten ir hacia arriba, hacia abajo y por todo el DOM sin problemas. En este artículo vamos a ver uno de ellos, los ***filtros***:

Filtros

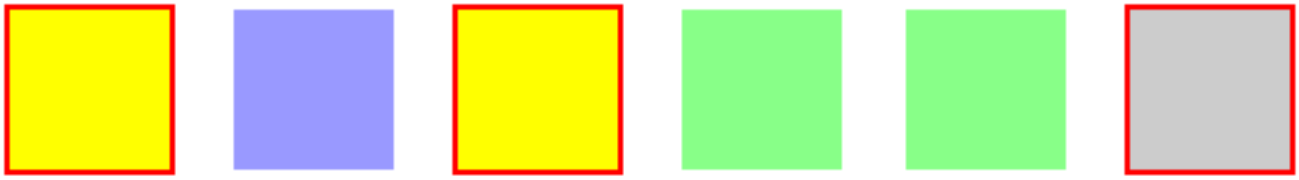
Son aquellos que concretan una selección de elementos ya realizada aplicando una selección adicional que limita la selección anterior (por ejemplo, hemos seleccionado previamente todos los elementos *div* y ahora queremos sólo el cuarto *div* de esa selección). Los filtros se distinguen así de los selectores que encuentran otros elementos que guardan relación con los elementos ya seleccionados (como los que encuentran descendientes, antecesores o hermanos de una selección anterior, por ejemplo, y que veremos en el artículo siguiente). **Vamos a ver los filtros más importantes**:

El filtro .not()

.not() lo utilizaremos siempre que queramos seleccionar aquellos elementos que ***no*** cumplen con un determinado filtro situado dentro de los paréntesis del .not(). Por ejemplo, si queremos añadir un borde a aquellos divs que no tengan un fondo verde o azul (este ejemplo y los siguientes los he tomado de la web oficial de jQuery):

```
$("div").not(".verde, #azul").css("border-color", "red");
```

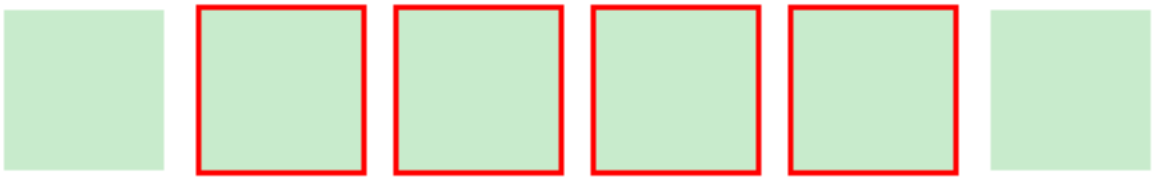
Primero selecciona todos los div del documento y luego selecciona aquellos que no tengan la clase ‘verde’ ni el div ‘azul’ y les aplica el estilo correspondiente de CSS.



El filtro .filter()

Otra manera de filtrar una selección de elementos es reducirla a aquellos que ***sí*** cumplen con un filtro adicional, y lo hacemos a través de*.filter()*
Por ejemplo, si queremos cambiar el color de fondo de todos los divs y después poner un borde alrededor de sólo los que tienen la clase*'centro'*

```
$("div").css("background", "#c8ebcc")
        .filter(".centro")
        .css("border-color", "red");
```



Y viendo un ejemplo algo más complejo, vamos a aplicar la clase 'imagen' a todas las imágenes de un documento, luego vamos a filtrar sólo las que tienen un *alt* que contenga la palabra 'importante' y sólo a esas les vamos a aplicar la clase 'borde' (para poder luego aplicarlas un borde a través de CSS o de jQuery, por ejemplo):

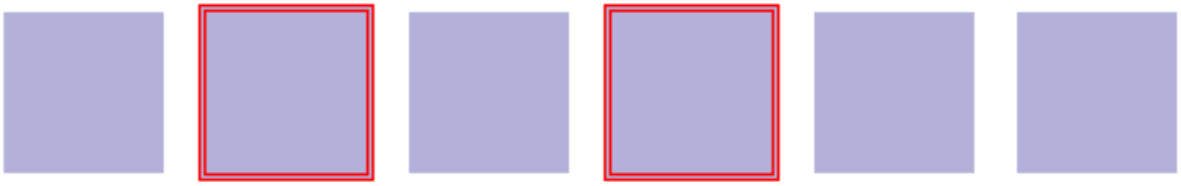
```
$('img').addClass('imagen').filter('[alt*=importante]').addClass('borde');
```

Pero lo que hace a filter() realmente potente es que podemos incluir una función como parámetro del mismo, por ejemplo (obtenido de la web oficial de jQuery):

```
$("div").css("background", "#b4b0da")
        .filter(function (index) {
```

```
            return index == 1 || $(this).attr("id") == "cuatro";
        })
    .css("border", "3px double red");
```

lo que hace es aplicar un color de fondo a todos los div del documento y luego filtra con una función (si ponemos como parámetro de la función*index* tendremos un índice que nos valdrá como contador y que empieza a contar desde cero) el segundo div (el que tiene índice 1) y el div con el id “cuatro” para aplicarles luego un borde rojo de 3px. A través de la función iremos div a div en un bucle devolviendo los que cumplen la condición:



No os preocupéis si no entendéis mucho de lo que se hace, lo importante es ver cómo funciona el selector. Sí destacar que dentro de la función aparece un término nuevo, **$(this)**. Veremos en próximos artículos del curso que *$(this)* se usa dentro de las funciones y hace referencia al elemento exacto al que afecta la función en ese momento, en este caso el div con id=”cuatro” (en vez de escribir *$(div).atrr(‘id’)*).

El filtro .slice(comienzo, final)

Hay ocasiones en las que queremos obtener una porción de una determinada selección de elementos basada en la *posición* de estos elementos dentro de la selección realizada. Entonces utilizamos .slice(), que creará una nueva selección. Admite dos parámetros, un número que indica el inicio (empezando por cero) donde vamos a realizar el corte y otro que indica el final **no incluído**. Este último valor es opcional. Vamos a ver un ejemplo:

Si seleccionamos todos los elementos *p* del documento y luego sólo queremos el tercer elemento haríamos:

```
$('p').slice(2,3);
```

El primer número (donde se empieza a cortar la selección inicial) es un dos porque se empieza a contar desde cero: 0, 1, 2 es decir, empezamos a contar por el tercer elemento. Para el final no incluído pone 3, es decir el cuarto elemento (0,1,2,3) por lo que de todos los *p* del documento solo se selecciona el tercero.

El filtro .eq()

Reduce el grupo de elementos seleccionados a un elemento único. Su argumento (lo que va entre los paréntesis) es la posición del elemento dentro del grupo de elementos previamente seleccionados (que puede ser un número entre 0 y la longitud completa del número de elementos menos uno, al haber empezado en cero).

Por ejemplo (tomado de la web oficial de jQuery), vamos a cambiar el color de fondo del div con índice 2 (es decir, el tercer div) añadiendo la clase apropiada:

```
$("div").eq(2).addClass("azul");
```

Selectores utilizados para encontrar un elemento

Estos selectores encuentran otros elementos que guardan relación con los elementos previamente seleccionados (como los que encuentran descendientes, antecesores o hermanos de una selección anterior, por ejemplo)

.next(), .nextAll(), .prev(), .prevAll()

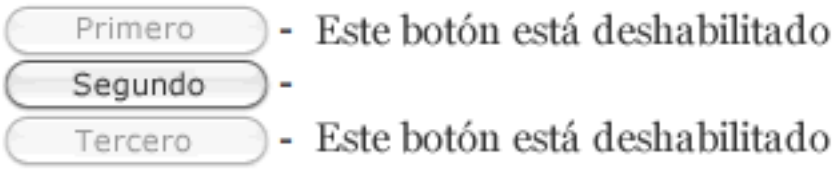
Estos métodos son bastante sencillos. Por ejemplo, si quisiéramos seleccionar sólo el elemento único siguiente hermano de cada elemento previamente seleccionado, utilizaríamos *.next()*.

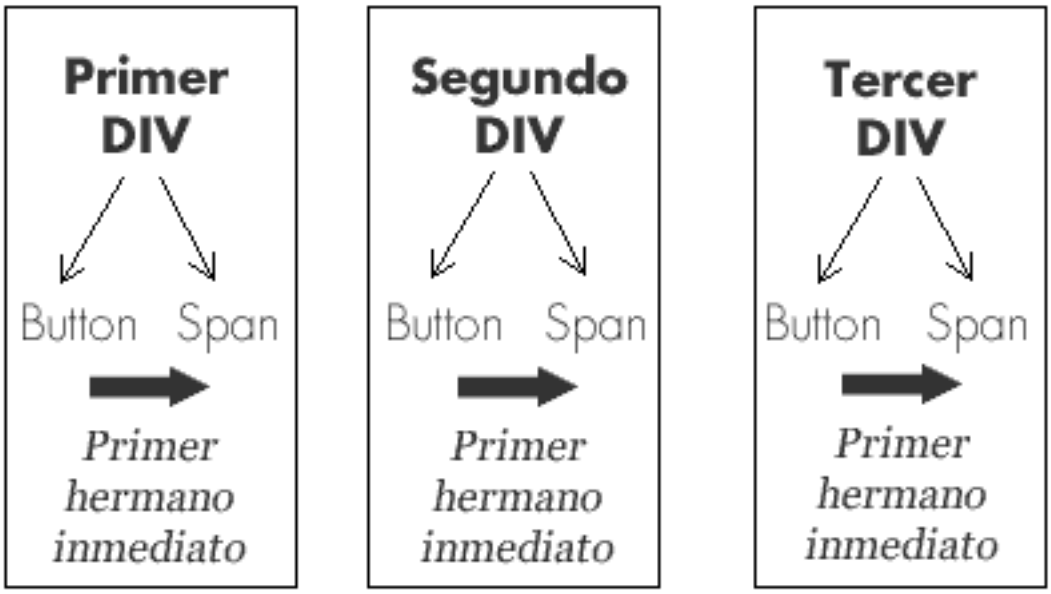
Vamos a ver un ejemplo (tomado de la web oficial de jQuery):

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

```
  $("button[disabled]").next().text("Este botón está desactivado");
 });
 </script>
 <style>
 span { color:blue; font-weight:bold; }
 button { width:100px; }
 </style>
</head>
<body>
 <div><button disabled="disabled">Primero</button> - <span></
span></div>
 <div><button>Segundo</button> - <span></span></div>
 <div><button disabled="disabled">Tercero</button> - <span></
span></div>
</body>
</html>
```

Lo que hace es encontrar el hermano inmediatamente posterior de cada *button* con atributo *disabled* (el *span*) y cambiar su texto por “Este botón está desactivado”:



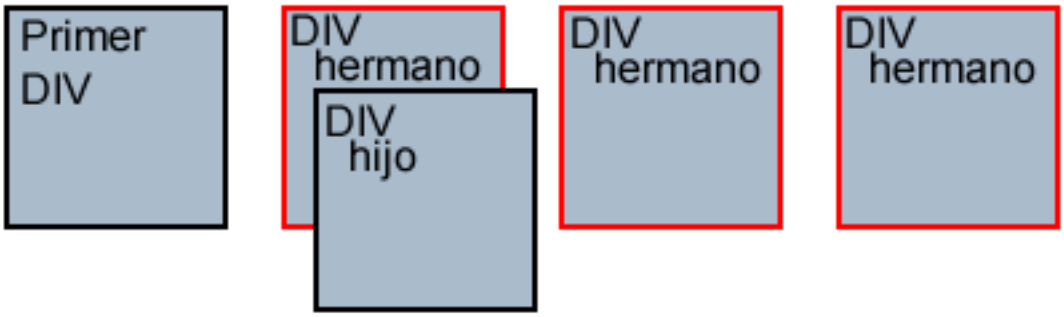


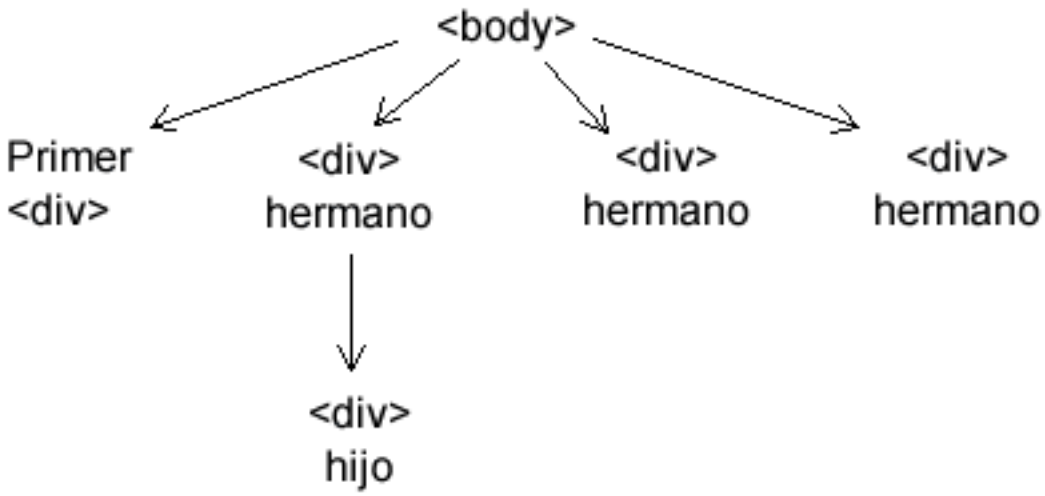
Si quisiéramos seleccionar todos los hermanos que siguen al elemento seleccionado utilizaríamos *.nextAll()*. Por ejemplo, localizar todos los divs que siguen al primero y aplicarles una clase que llamaremos “rojo” y que implica un borde rojo alrededor del DIV (fuente: web oficial de jQuery):

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//
EN"   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<script type="text/javascript" src="http://code.jquery.com/jquery-
latest.js"></script>
  <style>
  div { width: 80px; height: 80px; background: #abc;
        border: 2px solid black; margin: 10px; float: left; }
  div.rojo { border-color: red; }
  </style>
<script>
  $(document).ready(function(){
    $("div:first").nextAll().addClass("rojo");
  });
  </script>
</head>
<body>
```

```
  <div>Primer DIV</div>
  <div>DIV hermano<div>DIV hijo</div></div>
  <div>DIV hermano</div>
  <div>DIV hermano</div>
</body>
</html>
```

El resultado es:

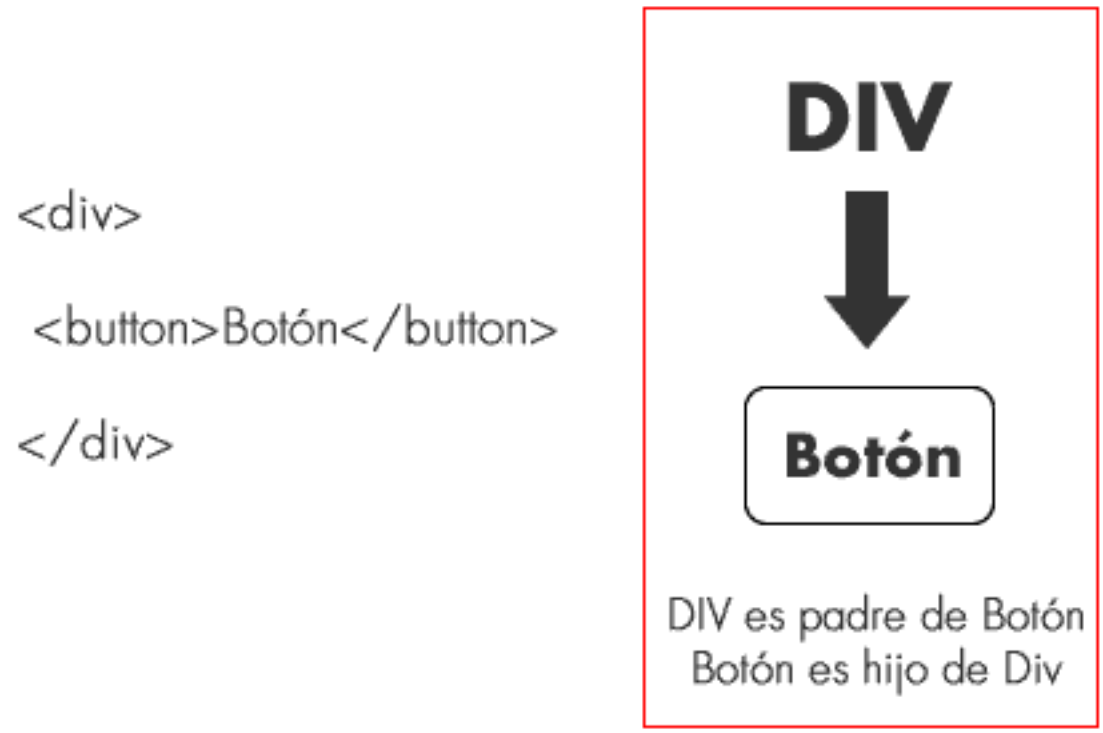




Como vemos, en esta ocasión se ha cargado jQuery desde su web oficial. Esto nos permite (como vimos en la primera parte del curso) cargar siempre la última versión disponible. ¿Es esto recomendable? Depende, ya que hay veces que algún plugin o código que tengamos de jQuery en nuestra web no soporte la última versión de jQuery.
Sus contrarios son *.prev()* y *.prevAll()*, que nos sirven para seleccionar a los anteriores en vez de a los siguientes.
.parent(), .parents(), .children(), .siblings()
.parent() obtiene el padre directo de un elemento. Si la selección es de un

grupo de elementos, obtiene un grupo de sus padres directos únicos.
Por ejemplo, supongamos el ejemplo anterior cuando veíamos los botones dentro de un Div. En este caso, seleccionaríamos el Div padre de cada botón y le aplicamos la clase rojo (que tendríamos definida en nuestra hoja de estilos como borde rojo) :

```
$("button").parent().addClass("rojo");
```



.parents() obtiene un grupo de elementos que contienen los antecedentes únicos de el grupo de elementos previamente seleccionados (excepto el elemento raíz).
.children() Obtiene un grupo de elementos que contienen todos los hijos inmediatos únicos de cada grupo de elementos reviamente seleccionados. Mientras .parents() devuelve todos los antecesores, .children() sólo considera los elementos hijo inmediatos.
.siblings() obtiene un grupo de elementos que contienen todos los

hermanos únicos del grupo de elementos seleccionados previamente.
Por ejemplo, supongamos que queremos aplicar un fondo blanco a todos los elementos hermanos de *“p”* que tengan la clase *“nombredeclase”*:

```
$("p").siblings(".nombredeclase").css("background", "white");
```

.contents()
Encuentra todos los nodos hijos de los elementos previamente seleccionados (incluyendo textos), o el contenido del documento si el elemento es un iframe.
Por ejemplo, si queremos seleccionar el *texto* que se encuentra en el interior de los elementos *“p”* (por ejemplo, para añadirle posteriormente más texto, etc..):

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

.add()
Añade más elementos -que cumplan con la selección que se encuentre dentro de los paréntesis de .add()- a la selección de elementos previamente seleccionados.
Por ejemplo, si habíamos seleccionado ya todos los elementos “p” de un documento y queremos añadir también los elementos “span”:

```
$("p").add("span")
```

.closest()
Este selector se ha introducido por primera vez con jQuery 1.3 Obtiene un grupo de elementos que contenga el elemento padre más cercano que cumpla con el selector especificado, incluyendo al elemento inicial. Comienza comprobando primero si el elemento actual cumple con la selección especificada. Si es así devuelve ese elemento. En caso contrario continúa buscando en el elemento, padre a padre, hasta que encuentra un elemento que cumpla la condición dada. Si no lo encuentra, no devuelve nada. Podéis ver un ejemplo muy interesante en la web oficial de jQuery.
.find()

Devuelve una nueva selección que contiene aquellos elementos descendientes de la selección previa que cumplen la condición dada dentro de los paréntesis de *find()*. Por ejemplo, si queremos seleccionar todos los *span* descendientes de elementos *p* haríamos:

```
$('p').find('span')
```

Encadenamientos

.not()

Es posible con jQuery seleccionar múltiples grupos de elementos y hacer múltiples cosas con ellos, todo en una única línea de código. Hemos visto que podemos partir de una selección inicial e ir reduciéndola o ampliándola. La nueva selección será la que estará activa para realizar sobre ella cualquier acción. Pero ¿qué pasa con la selección inicial? Podemos volver a ella en cualquier momento gracias al selector ***.end()*** Vamos a ver un ejemplo. Supongamos que tenemos:

```
$('div').add('a').css('background','black');
```

Hemos seleccionado todos los *div* del documento. Luego hemos creado un nuevo grupo que también contiene los elementos *a*. Cuando apliquemos el método css() lo haremos sobre el nuevo grupo creado. Pero ¿qué pasa si hacemos esto?:

```
$('div').add('a').css('background','black').end().css('color','red')
```

Después de aplicar el primer método *.css()* que crea un fondo negro hemos aplicado el método *.end()* devolviéndonos a la primera selección de los *div* sobre los que se aplicará el segundo método *.css()* que pone el color en rojo.

.andSelf()

Otro método útil para afectar las cadenas de jQuery es el método *.andSelf()*, que crea un nuevo grupo de elementos a partir de los dos últimos grupos de elementos seleccionados. Por ejemplo:

```
$('div').css('background,'black').children('img').css('border', '5px solid red').andSelf().css('padding','10px');
```

Hemos seleccionado todos los elementos *div* y les hemos dado un fondo negro. Después hemos seleccionado todos los elementos *img* hijos de los *div* anteriores y a estos les hemos aplicado un borde rojo de 5 píxeles. Y a continuación hemos unido las dos selecciones gracias a*.andSelf()* y les hemos aplicado a ambas un *padding* de 10px.

Accediendo directamente a los elementos del DOM: el método get()

Hay ocasiones en las que necesitamos acceder directamente a un elemento del DOM. Para estas ocasiones jQuery ofrece el método *.get()* Por ejemplo, para seleccionar todos los elementos *div*:

```
$('div').get()
```

O para obtener el primer div(empezamos a contar desde cero):
```
$('div').get(0)
```

Hasta aquí hemos visto los selectores de jQuery. Ahora la pregunta es: Vale, ya tengo unos elementos de mi web seleccionados. Ahora ¿Qué puedo hacer con ellos? Hemos visto pequeños avances en métodos como *.css()* para aplicar estilos CSS de manera directa o *.addClass()* para añadir el nombre de una clase, pero queda mucho por ver.