UT05.2: jQuery

Contenido

1. Introducción

DOM es muy tedioso de utilizar como hemos podido observar, las sentencias son muy largas y complicadas, además no permite encadenar métodos y las búsquedas de objetos son enrevesadas. La librería jQuery tiene como principal objetivo hacer más cantidad de operaciones sobre los objetos, escribiendo la menor cantidad de código posible.

Esta librería contiene las siguientes características:

- Manipulación DOM.
- Manipulación CSS.
- Manejadores de eventos HTML.
- AJAX
- Otras utilidades.

La mayoría de grandes empresas tecnológicas utilizan jQuery en sus webs, y es la base de otros frameworks populares como puede ser Bootstrap.

El API de la librería la podemos encontrar en https://api.jquery.com/ para que podamos consultar el detalle de cada uno de los objetos con los que vamos a trabajar.

1.1. Añadir jQuery a tu página

Tenemos varias versiones que podemos utilizar:

- **Producción**: Contiene la librería, pero de una forma comprimida.
- **Desarrollo**: La librería está descomprimida, por lo que es fácil seguir el código y editarlo.

Además de estas versiones, podemos encontrar la versión **slim** reducida que excluye las funciones Ajax y los módulos de efectos.

Para integrar jQuery vamos a importarlo directamente desde CDN de la web del proyecto https://code.jquery.com/. Debemos tener en cuenta que la librería debería ser el primer recurso antes de utilizarse.

```
<script src="https://code.jquery.com/jquery-3.7.1.min.js"
    integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
```

1.2. Evento *Document Ready*

Tenemos que asegurarnos de que el documento se haya cargado completamente antes de poder hacer uso de la librería, para ello hacemos uso del evento `ready` del objeto `document`. El carácter `$` hace referencia a que vamos a utilizar las funciones de jQuery. Como vemos en el código utilizamos una función anónima para definir el código que queremos ejecutar cuando se cargue la página.

```
$(document).ready(function(){
  console.log("Documento DOM cargado1");
});
```

Tenemos una abreviatura para esta sintaxis que es la que utilizaremos a lo largo del documento.

```
$(function(){
  console.log("Documento DOM cargado2");
})
```

Podemos averiguar el orden de ejecución de los eventos de la carga de la página. En primer lugar, se ejecutará el evento `DOMContentLoaded`, seguidamente el evento `ready`, por último, tendremos el evento `onload`. El siguiente código configura estos eventos para que apreciemos el orden de ejecución.

```
document.addEventListener("DOMContentLoaded", () => console.log("DOMConte
ntLoaded"));
window.onload = function(){
  console.log("window.onload")
};
```

Nota: el evento `ready` solo nos interesa para asegurarnos que el documento se ha cargado correctamente. Una vez cargado el documento podremos acceder a la librería como necesitemos, por ejemplo, en un manejador de eventos. No todo el código jQuery debe estar embebido en el evento `ready`.

1.3. Uso de la librería

Utilizamos jQuery igual que hacíamos con DOM, seleccionamos uno o varios elementos para manipularlos a través de sus propiedades, atributos y métodos. Como veremos a lo largo del documento, jQuery tiene sus propios métodos sobre los objetos.

Para seleccionar un elemento utilizamos los selectores de CSS. Vamos a crear dos funciones para mostrar y ocultar el elemento con identificador *categories* del documento. Los métodos `show()` y `hide()` ejecutan estás acciones. Las siguientes funciones pueden ser utilizadas como manejadores de evento.

```
function showCategories(){
  $('#categories').show();
}

function hideCategories(){
  $('#categories').hide();
}
```

Como podemos observar en este primer ejemplo, el código es muy más reducido que si lo hubiéramos hecho con DOM. En el siguiente link podemos ver ejemplos de cómo no siempre es necesario utilizar jQuery, pero que si lo utilizamos el código siempre es más reducido.

http://youmightnotneedjquery.com/

Podemos manipular el contenido de un elemento mediante dos métodos. El método `text()` permite cambiar el contenido de un elemento en modo texto, similar a la propiedad `innerText`. El método `html()` es el equivalente a `innerHTML`.

```
$(function(){
  $('#message').text("Texto normal");
  $('#message2').html(`Texto <span class="text-primary">HTML</span>`);
})
```

1.4. Compatibilidad con otras librerías

Otras librerías también hacen uso del carácter $, lo que podría generar problemas de incompatibilidad entre ambas librerías. Podemos hacer que el código jQuery se ejecute de forma local definiendo como parámetro `$`, el cual recibe como argumento la propia librería `jQuery`.

```
(function($){
  console.log("Compartir $ con otras librería.");
  $('#categories').hide();
}(jQuery))
```

Otra solución es utilizar el método `noConflict()` para asignar a una variable para utilizarlo como prefijo.

```
let jq = $.noConflict();
jq(function(){
  console.log("Nuevo prefijo.");
})
```

2. Selección de elementos y manipulación

Vamos a trabajar en la selección de elementos utilizando la librería, también vamos a trabajar en la manipulación de atributos y propiedades de los elementos seleccionados. Como ya hemos comentado, la selección de elementos está basada en los selectores de CSS, aunque ampliado con algunas pseudo-clases que facilitan la búsqueda.

2.1. Manipulación de atributos

Con el método `attr()` podemos acceder y manipular cualquier atributo de un elemento. Vamos a seleccionar la imagen del perfil del usuario y modificarla. Utilizaremos el servicio https://placeholder.com/ que permite generar imágenes de forma dinámica.

El siguiente código muestra cómo podemos accede al atributo `src` de la imagen y mostrarlo en el elemento *message* utilizando el método `attr()`, además modificamos el atributo `src` de la imagen para asígnale una nueva imagen. Como selector hemos utilizado *.image > img* que selecciona la imagen descendiente directa de la clase *.image*.

```
let profile = $('.image img');
$('#message').text(profile.attr('src')); // img/user.jpg
profile.attr('src', 'https://via.placeholder.com/150.jpg?text=User1');
```

Ahora vamos a seleccionar la primera imagen de las categorías, utilizamos el identificador y buscamos los descendientes `img`, por último utilizamos `first()` para obtener la primera ocurrencia. El método `attr()` permite modificar varios atributos simultáneamente utilizando un objeto literal. Vamos a modificar el atributo `src`, el `title` y el `alt` de la imagen.

```
$('#categories img').first().attr({
  src:'https://via.placeholder.com/258x172.jpg?text=Category1',
  title: 'Category1 Title',
  alt: 'Category1 Alt'
});
```

2.2. Objetos jQuery vs objetos DOM

Al trabajar con el API de jQuery, obtenemos objeto del API, sobre los cuales podremos aplicar los métodos del API. Independientemente del número de elementos que seleccionemos, el selector de jQuery siempre devuelve una colección de elementos. La propiedad `length` nos indica el número de elementos de la colección.

Como veremos posteriormente, para el API es independiente que el selector sea una colección, ya que aplicará todas las operaciones invocadas a través de sus métodos a todos los elementos de la colección de forma simultánea.

De un objeto jQuery podemos obtener la equivalencia a su objeto DOM. El método `get()` devuelve un array con todos los objetos DOM que representan los elementos seleccionados por el objeto jQuery. Independientemente del número de elementos, `get()` devuelve un array, por lo que tendríamos que iterar sobre él para obtener el objeto que nos interese.

```
let allElements = $('*');
console.dir(allElements.length);
console.dir(allElements.get());
let button = $('#button');
console.dir(button);
console.dir(button.get());
```

2.3. Selección de elementos por atributos

Podemos seleccionar elementos en función de sus atributos, y en función de los valores de los atributos. Vamos a ver varios ejemplos.

Todos los elementos que tengan atributo `id`.

```
let idElements = $('*[id]');
```

Todos los enlaces que tengan un valor "#" en su atributo `href`.

```
let anchors1 = $('a[href="#"]');
```

El contrario, todos los enlaces que no tengan un valor "#".

```
let anchors2 = $('a[href!="#"]');
```

Todas las imágenes que tengan extensión jpg, es decir, terminen en jpg.

```
let jpg = $('img[src$=".jpg"]');
```

Todas las imágenes cuyo atributo `alt` comience por "Categoría".

```
let alt = $('img[alt^="Categoría"]');
```

Podemos seleccionar por contenido, por ejemplo, las imágenes que en su atributo `alt` contiene el texto 'iles'.

```
let alt2 = $('img[alt*="iles"]');
```

Aunque también podemos hacer que sea la palabra completa.

```
let alt3 = $('img[alt~="iles"]');
```

2.4. Selección de elementos por orden

Podemos seleccionar elementos en función de su orden.

2.4.1. Métodos `first()` y `last()`

Además del método `first()` que selecciona el primer elemento de una colección, el método `last()` selecciona el último elemento. En este ejemplo modificamos la primera y la última imagen de las categorías.

```
$('#categories img').first().attr({
  src:'https://via.placeholder.com/258x172.jpg?text=First',
});
$('#categories img').last().attr({
  src:'https://via.placeholder.com/258x172.jpg?text=Last',
});
```

2.4.2. Pseudo-clases `:first-child` y `:last-child`

Podemos seleccionar el **primer hijo** de un elemento o su **último hijo**. Las pseudo-clases `:first-child` y `:last-child` representan estos elementos. En nuestra página tenemos el elemento *#categories*, el cual es contenedor con un único elemento *.row*. Esta fila está dividida en 4 columnas. El selector *#categories div.row > div:first-child* representa la primera columna, y el selector *#categories div.row > div:last-child* la última. Vamos a añadir un borde a cada una de estas columnas modificando su estilo con el método `css()`.

```
let firstCol = $('#categories div.row > div:first-child');
let lastCol = $('#categories div.row > div:last-child');
firstCol.css('border', '1px solid red');
lastCol.css('border', '1px solid red');
```

2.4.3. Pseudo-clases `:first-of-type` y `:last-of-type`

Al igual que en el ejemplo anterior, podemos seleccionar el primer elemento hijo de un elemento, pero que sea de un tipo concreto de elemento. Vamos a seleccionar todos los elementos `ul` del documento que sean el primer hijo de este tipo, así como todos los elementos `button`.

```
$('ul:first-of-type').css('border', '1px solid red');
$('button:first-of-type').css('background', 'red');
```

2.4.4. Pseudo-clases `:nth-child(n)` y `:nth-last-child(n)`

Estas pseudo-clases nos permite seleccionar un elemento en función de su posición. El selector *footer div.row div:nth-child(2)* selecciona dentro del pie de la página la segunda columna correspondiente a los enlaces de interés. Dentro de los enlaces, vamos a seleccionar el segundo de ellos comenzando por el principio, y el segundo comenzando por el final con `:nth-last-child()`. Los elementos comienzan a enumerarse en "1".

```
$('footer div.row div:nth-child(2)').css('border', '1px solid red');
$('footer div.row div:nth-child(2) li:nth-
child(2)').css('border', '1px solid red');
$('footer div.row div:nth-child(2) li:nth-last-
child(2)').css('border', '1px solid blue');
```

2.4.5. Selección en lista de elementos `:eq(n)`, `lt(n)` y `gt(n)`

Hasta ahora hemos trabajado respecto al orden de los elementos en el documento, pero también podemos hacerlo en función del orden en una selección. La pseudo-clase `:eq(n)` permite seleccionar un elemento en función de una selección previa. El selector `li:eq(2)` selecciona de todos los elementos `li` de la página el que sería el tercero de la selección empezando por cero.

```
$('footer div.row div:nth-child(3) li:eq(2)');
```

Podemos hacer una selección más específica. Vamos a seleccionar de los servicios, los que estén en posiciones menor de 2 y mayor de 2, lo hacemos con las pseudo-clases `lt(n)` y `gt(n)`.

```
$('footer div.row div:nth-
child(3) li:lt(2)').css('border', '1px solid red');
$('footer div.row div:nth-
child(3) li:gt(2)').css('border', '1px solid blue');
```

2.4.6. Métodos even () y odd ()

Los métodos `even()` y `odd()` permiten seleccionar los elementos pares e impares de una colección concreta.

```
$('footer div.row div:nth-
child(2) li a').even().css('border', '1px solid red');
$('footer div.row div:nth-
child(2) li a').odd().css('border', '1px solid blue');
```

2.5. Iteración sobre selección múltiple

Ya hemos visto ejemplos donde hemos hecho selección de varios elementos a la vez, creando una colección de estos elementos.

Podemos iterar sobre la colección de elementos que recibimos utilizando la propiedad `length` o con un bucle `for-of`. Los objetos de la colección son DOM, podemos acceder a ellos a través de la notación corchete, o utilizando el método `get()` y un índice.

Vamos a enumerar los elementos `li` del `footer` de la página, y con un bucle iteramos sobre la colección con la propiedad `length` como condición de salida. Modificamos cada elemento de la colección como objetos DOM.

```
let liElements = $('footer div.row li');
for (let i = 0; i < liElements.length; i++){
 liElements.get(i).innerHTML = "Elemento: " + i + " de " + liElements.len
gth;
}
```

Podemos transformar el objeto DOM y utilizarlo como un objeto jQuery a través de la función `$()`.

```
let liElements = $('footer div.row li');
for (let i = 0; i < liElements.length; i++){
  $(liElements[i]).html("Tradicional:" + i + " de " + liElements.length);
}
```

Podemos realizar el mismo ejemplo anterior, pero utilizando un bucle `for..of`.

```
let liElements = $('footer div.row li');
let i = 0;
for (let element of liElements){
  $(element).html("For-of: " + i++ + " de " + liElements.length);
}
```

Además de los métodos anteriores, tenemos un tercer método para iterar con una colección de elementos jQuery que es utilizando el método propio `each()`, como argumento tiene una función de callback. La función puede recibir como argumento el índice del elemento. La referencia `this` toma el valor del objeto DOM que representa el elemento.

```
let liElements = $('footer div.row li');
liElements.each(function(index){
  $(this).html("Each: " + index + " de " + liElements.length);
});
```

Una de las grandes ventajas que nos da jQuery es poder interactuar con todos los elementos de la colección simultáneamente. Trabajando con DOM, si queremos modificar el estilo tenemos que ir uno por uno modificando el estilo de la colección. El siguiente ejemplo itera sobre todos los de la colección de `li` para modificar su borde.

```
let liElements = $('footer div.row li');
for (let li of liElements){
  li.style.border = "1px solid red";
}
```

El mismo ejemplo lo podemos realizar en un único paso utilizando jQuery ya que el método css() se aplica a todos los elementos de la colección.

```
let liElements = $('footer div.row li');
liElements.css('border', '1px solid blue');
```

La colección de elementos puede ser heterogénea y los métodos siguen teniendo aplicación múltiple. En este ejemplo seleccionamos las imágenes de categorías y de los elementos `li` del `footer`.

```
let liElements = $('footer div.row li, #categories img');
liElements.css('border', '1px solid green');
```

Por último, podemos seleccionar el índice de un elemento en una colección utilizando el método `index()` sobre una colección, e indicando el objeto sobre el que queremos descubrir el índice.

```
let images = $('#categories img');
let img1 = $('#categories img:eq(1)');
console.log(images.index(img1));
```

2.6. Filtrado

En muchas ocasiones tendremos que refinar la primera selección de elementos que hemos hecho. Vamos a trabajar con algunos métodos que nos permiten refinar la primera selección.

2.6.1. Método `filter()`

Este método nos permite seleccionar de la colección una serie de elementos utilizando los siguientes criterios:

- Selectores.
- Una función de callback.
- Uno o más elementos de tipo DOM.
- Una selección jQuery.

Veamos unos ejemplos.

Filtramos de todos los elementos `i`, los que tengan la clase *.fa-angle-right*.

```
$('i').filter('.bi-chevron-right')
  .css('border', '1px solid red');
```

De las columnas que tenemos en el `footer`, vamos a seleccionar las que ocupan 3 espacios.

```
$('footer div.row div').filter('.col-lg-3')
  .css('border', '1px solid red');
```

Si utilizamos una función de callback, en ella debemos elegir con qué elementos nos quedamos y con cuáles no utilizando un valor booleano de retorno. Como argumentos puede recibir el índice de la posición del elemento. Veamos unos ejemplos.

Queremos filtrar la primera y la última imagen que tenemos en categorías, para ello vamos a comparar `this` con los objetos que devuelve los métodos `first()` y `last()` de la colección de imágenes. Estos objetos están en una colección jQuery de un único elemento, por lo que para que sean comparables lo tenemos que acceder a DOM a través de índice 0.

```
let images = $('#categories img');
images.filter(function (index){
  return (this === images.first().get(0) ||
      this === images.last().get(0));
}).attr({src:'https://via.placeholder.com/258x172.jpg?text=Filtrado1'})
```

Repetimos el ejemplo para seleccionar solamente las imágenes que no sean ni la primera ni la última de la colección, utilizamos el parámetro índice para ello.

```
images.filter(function (index){
  return (index > 0 && index < images.length-1);
}).attr({src:'https://via.placeholder.com/258x172.jpg?text=Filtrado2'})
```

Por último, vamos a buscar el contenedor que tiene el elemento con identificador *#button*. El método `contains()` devuelve un booleano si un elemento contiene a otro. Comparamos si la referencia `this` con el elemento del identificador.

```
let containers = $('.container');
let button = $('#button').get(0);
containers.filter(function(){
  return $.contains(this, button);
}).css('border', '1px solid red');
```

2.6.2. Método `not()`

Con este método seleccionaremos lo opuesto al selector pasado como argumento. Los argumentos pueden ser:

- Selectores.
- Una función de callback.
- Una selección jQuery.

Vamos a seleccionar todos los elemento `i` que no tengan asociado la clase *.fa-angle-right*.

```
$('i').not('.bi-chevron-right')
.css('border', '1px solid red');
```

Seleccionamos todos los contenedores que no incluyan el elemento *#button*.

```
let containers = $('.container');
let button = $('#button').get(0);
containers.not(function(){
  return $.contains(this, button);
}).css('border', '1px solid red');
```

2.6.3. Método `has()`

Nos permite seleccionar elementos que tienen como contenido un selector o un elemento DOM determinado.

En el siguiente ejemplo repetimos la búsqueda del contenedor con el elemento *#button*.

```
let containers = $('.container');
let button = $('#button').get(0);
containers.has(button).css('border', '1px solid red');
```

2.6.4. Método `map()`

Por cada elemento de una selección, el método `map()` va a ejecutar una función de callback que permita transformar el elemento de la selección en un nuevo objeto. El método devuelve una colección con todos los nuevos objetos en un objeto jQuery.

Vamos a seleccionar todos los elementos que tengan un atributo identificador, y vamos a mostrar todos los identificadores en la página. El siguiente código mapea la selección previa para obtener un array con los nuevos objetos, en este caso strings, en un objeto jQuery, por lo que para acceder necesitamos el método `get()`.

```
let idElements = $('*[id]');
let map = idElements.map(function (index){
  return this.id;
});
console(map.get().join(' '));
```

2.6.5. Método `slice()`

Reduce una selección al rango de elementos especificados en base al índice de la colección. Como argumento recibe el índice de inicio y el índice final del rango. Los elementos son eliminados de la colección, no del árbol DOM.

De las imágenes de categorías, vamos a eliminar la de los extremos para hacer una modificación.

```
let images = $('#categories img');
images = images.slice(1,images.length-1);
images.attr({src:'https://via.placeholder.com/258x172.jpg?text=Slice'});
```

2.7. Relación entre elementos

Al igual que con DOM, podemos seleccionar elementos a través de su relación de “parentesco”.

2.7.1. Ancestros

El método `parent()` nos permite acceder al padre de un elemento. En el siguiente ejemplo ponemos un borde rojo al padre del elemento, y lo hacemos recursivamente hasta tres niveles.

```
let img0 = $('#categories img:eq(0)');
img0.parent().css('border', '1px solid red');
img0.parent().parent().css('border', '1px solid red');
img0.parent().parent().parent().css('border', '1px solid red');
```

El método `parents()` selecciona todos los ancestros del elemento hasta la raíz.

```
let img1 = $('#categories img:eq(1)');
img1.parents().css('border', '1px solid blue');
```

Tenemos el método `parentsUntil()` que selecciona todos los ancestros hasta un determinado elemento especificado por un objeto DOM o jQuery. El elemento no está incluido.

```
let img2 = $('#categories img:eq(2)');
img2.parentsUntil($('#categories')).css('border', '1px solid green');
```

El método `offsetParent()` selecciona el primer ancestro que esté posicionado, es decir, que tenga definida la propiedad CSS `position`. En el ejemplo seleccionamos el ancestro que esté posicionado del elemento *#button*.

```
let button = $('#button');
button.offsetParent().css('border', '10px solid red');
```

Por último, podemos buscar el ancestro de un elemento que se empareja con un criterio. En este ejemplo buscamos el contenedor del elemento *#button*.

```
button.closest('.container').css('border', '10px solid blue');
```

2.7.2. Descendientes

El método `children()` nos da acceso a todos los elementos hijos de un objeto jQuery. Vamos a cambiar los bordes de los elementos `li` de los enlaces de interés.

```
let links = $('#footer div.row div:nth-child(2) ul');
links.children().css('border', '1px solid red');
links.children().first().css('border', '1px solid blue');
links.children().last().css('border', '1px solid green');
```

El método `find()` nos permite seleccionar elementos a partir de un elemento concreto. En este ejemplo buscamos las imágenes a partir del elemento *#categories*.

```
let categories = $('#categories');
categories.find('img').attr('src', 'https://via.placeholder.com/150.jpg?text=Find');
```

El método permite buscar en colecciones. En este ejemplo seleccionamos todos los contenedores, para luego buscar sus elementos *.row*.

```
let containers = $('.container');
containers.find('.row').css('border', '1px solid red');
```

Al igual que que con `find()`, el método `children()` puede recibir un selector como argumento para filtrar los hijos que cumpliesen un determinado patrón.

2.7.3. Hermanos

Vamos a recorrer los hermanos del tercer `li` de los enlaces de servicios. El método `siblings()` selecciona todos los hermanos del elemento.

```
let li = $('#footer div.row div:nth-child(3) ul li:eq(2)');
li.siblings().css('border', '1px solid red');
```

Con los métodos `nextAll()` y `prevAll()` seleccionamos todos los elementos siguientes y previos respectivamente.

```
li.nextAll().css('background', 'red');
li.prevAll().css('background', 'blue');
```

Si queremos ir de uno en uno lo hacemos con `next()` y con `prev()`.

```
li.next().css('background', 'green');
li.prev().css('background', 'green');
```

Por último, podemos recorrer los hermanos hasta uno determinado con `nextUntil()` y `prevUntil()`.

```
li.nextUntil('*:nth-child(5)').css('text-decoration', 'underline');
li.prevUntil('*:nth-child(1)').css('text-decoration', 'underline');
```

3. Manipulación de elementos en la página

En este punto vamos a trabajar cómo crear nuevos elementos y añadirlos en la página, y cómo podemos borrarlos igual que hicimos con DOM.

3.1. Fijar contenido

Ya hemos trabajado con los métodos `text()` y `html()`, los cuales nos permiten recuperar el contenido de un elemento en formato texto o en formato HTML. También hemos visto cómo modificar el contenido de un elemento utilizando un string como argumento.

Además del uso de string, podemos utilizar una función de callback como argumento. La función puede recibir dos parámetros, el índice con la posición del elemento en la colección, utilizado básicamente para asignación múltiple, y el antiguo texto del elemento. El retorno será el contenido a mostrar por el elemento.

```
let ul = $('#footer div.row div:nth-child(3) ul');
ul.find('a').text(function (index, oldText){
  return oldText + " MiTienda " + (index + 1);
});
```

El método `val()` nos permite trabajar con los valores de los `INPUT` en un formulario. Si no tiene argumento devuelve el valor del `INPUT`. Si le pasamos un argumento puede ser un string o una función de callback para asignar el valor al `INPUT`.

```
let email = $('input[type=email');
email.val('usuario@dominio.es');
```

El método `attr()` admite como argumentos, un string para asignarlo como valor, un objeto literal, que nos permite hacer varias modificaciones simultáneas, y un función de callback como en los casos anteriores.

```
let logo = $('#logo');
logo.attr({
  title: 'Enlace logo',
  href: 'https://api.jquery.com/',
  alt: 'Logo de la página'
})
```

3.2. Cambiar estilos

Los hacemos mediante el objeto `style` o mediante clases.

3.2.1. Utilizando el objeto `style`

Hemos trabajado con el método `css()` pasando como argumento el nombre de la propiedad y el valor. El valor puede ser un string, como hasta ahora hemos hecho, utilizando una función de callback, o un objeto literal con las propiedades a modificar.

```
let ul = $('#footer div.row div:nth-child(3) ul');
ul.children().css({
  border: '1px solid red',
  textDecoration: 'underline',
});
```

3.2.2. Utilizando clases

Podemos añadir, eliminar o activar y desactivar clases utilizando los métodos `addClass()`, `removeClass()` y `toggleClass()` respectivamente. Las clases las pasamos como argumentos en forma de string, para clase individual, array de string, para un conjunto de clases, o una función de callback cuyos argumentos son el índice y las clases aplicadas en ese momento.

El siguiente código recoge primero de los enlaces de servicios los impares, para en el elemento `i` borrar las clases y añadir un globo como icono. El segundo paso es hacerlo para los pares, pero a través de una función de callback, donde en función de la posición mostramos un insecto con o sin relleno.

```
const ul = $('#footer div.row div:nth-child(3) ul');
const oddI = ul.children().odd().find('i');
oddI.removeClass(['bi', 'bi-chevron-right']);
oddI.addClass(['bi', 'bi-balloon-fill']);
const evenI = ul.children().even().find('i');
evenI.removeClass(['bi', 'bi-chevron-right']);
evenI.addClass((index, classes) => {
  console.log(index);
  if (index % 2 === 0) { return ['bi', 'bi-bug']; }
  return ['bi', 'bi-bug-fill'];
});
```

El método `toggleClass()` lo veremos más adelante cuando apliquemos interacción al contenido.

Por último, disponemos del método `hasClass()` que nos indica si un elemento incluye una clase o no.

3.3. Añadir contenido

Tenemos diversos mecanismos de añadir contenido al árbol DOM de la página.

3.3.1. Insertar elementos

Tenemos cuatro métodos para añadir

- `append()`: Inserta contenido al final del elemento como hijo.
- `prepend()`: Inserta contenido como primer hijo.

- `after()`: Inserta contenido como siguiente hermano del elemento.
- `before()`: Inserta contenido como hermano previo del elemento.

Como argumento, estos métodos aceptan string con HTML, un objeto DOM, un objeto jQuery, o un array con varios objetos. Además, admiten varios argumentos para hacer varias inserciones simultaneas, lo que nos ofrece una gran versatilidad.

Antes de utilizar los métodos, vamos a ver cómo podemos crear un elemento utilizando jQuery. Utilizamos como constructor la función `$` y le pasamos el HTML en forma de string. Como vemos en el ejemplo, hemos creado 4 elementos `DIV`, y hemos utilizado el encadenamiento de métodos para poder asignarles texto y estilo. También hemos creado una lista de ítems.

```
let div1 = $('<div></div>').text('Append').css('border','1px solid red');
let div2 = $('<div></div>').text('Prepend').css('border','1px solid red')
;
let div3 = $('<div></div>').text('After').css('border','1px solid red');
let div4 = $('<div></div>').text('Before').css('border','1px solid red');
let ul = $('<ul><li>item1</li><li>item2</li><li>item3</li></ul>')
```

Creamos unos objetos DOM para añadir como contenido.

```
let dom1 = document.createElement('div');
dom1.appendChild(document.createTextNode('DOM Append'));
dom1.style.border = '1px solid blue';
let dom2 = document.createElement('div');
dom2.appendChild(document.createTextNode('DOM Prepend'));
dom2.style.border = '1px solid blue';
let dom3 = document.createElement('div');
dom3.appendChild(document.createTextNode('DOM After'));
dom3.style.border = '1px solid blue';
let dom4 = document.createElement('div');
dom4.appendChild(document.createTextNode('DOM Before'));
dom4.style.border = '1px solid blue';
```

Por último, invocamos los métodos de inserción utilizando argumentos de objetos y arrays sobre el elemento *#categories*.

```
let categories = $('#categories');
categories.css('border', '5px solid green');
categories.append(div1, [dom1, 'Texto en append']);
categories.prepend(div2, [dom2, 'Texto en prepend']);
categories.after(div3, [dom3, 'Texto en after']);
categories.before(div4, [dom4, 'Texto en before']);
```

3.3.2. Insertar y mover elementos respecto un elemento

Un objeto jQuery puede ser insertado en el árbol DOM utilizando los métodos `appendTo()` y `preprendTo()`. Como argumentos toman el target donde queremos insertar el objeto. La ventaja de estos métodos es que nos permiten hacer varias inserciones simultáneamente si el target es una colección.

En este ejemplo creamos dos objetos `DIV` y los añadimos en cada columna de elemento *#categories*.

```
let div1 = $('<div></div>').text('appendTo').css('border','1px solid red'
);
let div2 = $('<div></div>').text('prependTo').css('border','1px solid red
');
div1.appendTo('#categories div.row > div');
div2.prependTo('#categories div.row > div');
```

Estos métodos también nos permiten mover un objeto dentro del árbol. Vamos a intercambiar la primera columna por la última. Primero hacemos la selección de ambos objetos, y posteriormente invocamos los métodos para moverlo al target elegido, en este caso la fila en *#categories*.

```
let firstColumn = $('#categories div.row > div').first();
let lastColumn = $('#categories div.row > div').last();
firstColumn.appendTo('#categories div.row ');
lastColumn.prependTo('#categories div.row ');
```

Podemos hacer inserciones de elementos en lugar de como hijos como hermanos. Los métodos `insertAfter()` e `insertBefore()` realizan está acción, para ello debemos indicar el target donde realizaremos la inserción.

Vamos a copiar una de las columnas de *#categories* y la vamos a insertar en el centro. Primero vamos a copiar la primera columna. El método `clone()` hace la copia de cualquier elemento. Este método admite dos argumentos boolenos, ambos false por defecto, para indicar si el clonado incluye los eventos asignados al elemento y los eventos de sus descendientes.

```
let category = $('#categories div.row > div').first();
let newCategory = category.clone(true, true);
```

Hacemos unas modificaciones en el nuevo elemento y lo insertamos después de la segunda columna.

```
newCategory.find('img').attr('src', 'https://via.placeholder.com/258x172.
jpg?text=Categoría nueva');
newCategory.find('h3').text('Nueva categoría');
newCategory.insertAfter(category.next());
```

Los métodos `insertAfter()` e `insertBefore()` también permiten mover elementos en el árbol DOM. Repetimos el ejemplo de mover las categorías.

```
let firstColumn = $('#categories div.row > div').first();
let lastColumn = $('#categories div.row > div').last();
firstColumn.insertAfter('#categories div.row > div:last-child');
lastColumn.insertBefore('#categories div.row > div:first-child ');
```

3.3.3. Reemplazar elementos

Podemos reemplazar un elemento por otro utilizando el método `replaceWith()`. Dada una serie de objetos seleccionados, el método los sustituye por el nuevo objeto pasado como argumento.

El método `replaceAll()` es el inverso. Dado un objeto, le pasamos un target que queremos reemplazar.

```
let newAnchor = $('<a></a>').text('Nuevo enlace');
newAnchor.attr('href','#');
```

```
let links1 = $('#footer div.row div:nth-child(2) ul li a');
let links2 = $('#footer div.row div:nth-child(3) ul li a');
links1.replaceWith(newAnchor);
newAnchor.replaceAll(links2);
```

3.3.4. Envolver elementos

Podemos crear un elemento que nos sirva *wrapper* sobre un cojuntos de elementos dentro del árbol DOM, es decir, este nuevo elemento será insertado como el padre de los elementos seleccionados.

Vamos a crear un elemento que nos sirva de *wrapper* para las selecciones que consideremos.

```
let div = $('<div></div>').css({
  border: '5px solid red',
  margin: '10px'
})
```

Tenemos cuatro opciones.

El método `wrap()` añade el elemento como padre de los objetos seleccionados. Añadimos a los `ul` del `footer` un nuevo `div`.

```
$('footer ul').wrap(div);
```

El método `wrapInner()` añade el nuevo elemento como padre del contenido de la selección. Añadimos el nuevo elemento como padre del contenido de cada columna de *#categories*.

```
$('#categories div.row > div').wrapInner(div);
```

Podemos hacer que todos los elementos de una selección tengan el mismo padre con `wrapAll()`. Seleccionamos el `header` y el primer `div` para que tengan el mismo padre.

```
$('body > header, body > div').wrapAll(div);
```

Por último, podemos eliminar todos los padres de un elemento con `unWrap()`. Eliminamos los padres de los contenedores de `footer`.

```
$('footer div.container').unwrap();
```

3.3.5. Eliminar elementos

Tenemos tres métodos que nos permiten eliminar elementos de árbol DOM. El método `remove()` eliminar la selección de objetos que sobre la que estamos ejecutando. Opcionalmente recibe un parámetro con un selector que nos permita hacer un filtrado más fino, por lo que solamente borrará de la selección los que cumplan el selector. En este ejemplo seleccionamos todos los elementos `li` del `footer`, pero al no seleccionar los que están en primera posición, estos se mantienen.

```
$('footer ul li').remove('li:not(:first-child)');
```

Podemos eliminar el contenido de un elemento determinado, dejándolo vacío utilizando el método `empty()`. Eliminamos el contenido de `div.banner`.

```
$('div.banner').empty();
```

Por último, el método `detach()` es igual que `remove()`, pero mantiene la referencia del objeto jQuery, por lo que podemos añadirlo al árbol posteriormente. En este caso eliminado la primera categoría y la añadimos al final del contenedor.

```
let firstColumn = $('#categories div.row > div').first();
firstColumn = firstColumn.detach();
$('#categories div.row').append(firstColumn);
```

3.3.6. Dimensiones

Tenemos métodos que nos permiten acceder a las dimensiones de un elemento. Algunos de estos métodos son getter y otros pueden funcionar como getter/setter.

- `height()`: Permite obtener la altura del contenido del elemento. Además, podemos fijarla si la pasamos como argumento.
- `width()`: Igual que la anterior pero referente al ancho del contenido del elemento.
- `innerHeight()`: Da la altura de un elemento incluido el padding y el borde.
- `innerWidth()`: Da el ancho de un elemento incluido el padding y el borde.
- `outerHeight()`: Incluye el margen.
- `outerWidth()`: Incluye el margen.
- `offset()`: Devuelve un objeto con las propiedades top y left del elemento relativas al documento.
- `position()`: Es igual que la anterior pero las propiedades top y left están referidas al elemento padre.
- `scrollLeft()`: Permite obtener o fijar el scroll horizontal en una determinada posición.
- `scrollTop()`: Permite obtener o fijar el scroll vertical en una determinada posición.

4. Eventos

La librería jQuery dispone de su propio objeto `Event` para trabajar con los eventos. Este objeto copia todas las propiedades del objeto `Event` del API de DOM.

El funcionamiento es básicamente igual que con DOM, pero tenemos un método específico por cada tipo de evento, al cual pasaremos como argumento la función que nos servirá de manejador de eventos.

Las categorías de eventos que tenemos son:

- Eventos del navegador.
- De carga de documento.
- Acoplamiento de manejadores.
- Eventos de formulario.
- Eventos de teclado.
- Eventos de ratón.

Al igual que en los eventos DOM, un evento jQuery puede recibir como argumento el evento que ha sido capturado por el manejador. La referencia `this` apunta al objeto donde se captura el evento, es decir, el elemento donde hemos definido el manejador de eventos.

Vamos a realizar los mismos ejemplos que hicimos en DOM para trabajar con la librería jQuery.

4.1. Manejadores de eventos

Vamos a empezar a ver cómo podemos añadir manejadores de eventos a un elemento con jQuery. Utilizamos el método `click()` para asociar el manejador de eventos.

4.1.1. Ejemplo 1: Enlaces de interés

Queremos añadir un elemento `input` y un `button` en los enlaces de interés. El botón debe recoger el texto del input, validar que tenga al menos cinco caracteres y añadirlo como enlace al listado.

La primera parte creamos el `input` y el `button` y los añadimos como hermanos del elemento `ul` con los enlaces.

```
let links = $('#footer div.row div:nth-child(2) ul');
let input = $('<input>').attr({
  type: 'text',
  id: 'new-link',
  placeholder: 'Nuevo enlace'
});
let button = $('<button></button>').text('Añadir');
links.after(button);
links.after(input);
```

Construimos el manejador dentro del método `click()`.

```
$('button').click(function(){
  let input = $('#new-link');
  if (input.val().length > 4){
    let links = $('#footer div.row div:nth-child(2) ul');
    let li = $('<li></li>')
    let anchor = $('<a></a>').attr({
      href: '#'
    }).text(input.val());
    let i = $('<i></i>').addClass('bi bi-chevron-rightt');
    li.append(i, anchor);
    links.append(li);
  } else {
    alert ('La longitud debe ser de al menos 5 caracteres.');
  }
});
```

4.1.2. Ejemplo 2: Imágenes a tamaño completo

Queremos añadir un manejador de eventos para cada una de la sección de categorías, para que al cliquear en ellas se abra la imagen a tamaño de pantalla completo. Podemos utilizar el método `requestFullscreen()` para hacerlo.

Seleccionamos los elementos con los que vamos a trabajar.

```
let categories = $('#categories');
let images = categories.find('img');
```

Creamos una función como manejador. En este caso, al querer añadir la misma función a varios elementos es mejor no utilizar una función anónima, ya que de esa forma se estaría creando funciones diferentes por cada imagen. Con una función nominal podemos asociar el mismo manejador a todas las imágenes.

```
function imageFullScreen(){
  if (this.requestFullscreen) {
    this.requestFullscreen();
  }
}
```

Por último, añadimos el manejador a todas las imágenes.

```
images.click(imageFullScreen);
```

4.2. Acoplamiento de manejadores

Tenemos métodos que nos permiten asociar más de un manejador para la misma colección de objetos.

4.2.1. Asociar varios eventos a un objeto

El método `on()` nos permite asociar varios manejadores a un objeto. El método es muy versátil, vamos primero a redefinir el ejemplo anterior, como argumento indicamos el tipo de evento, y el manejador que queremos pasar. Con on() podemos definir varios manejadores al mismo tipo de evento.

```
images.on('click', imageFullScreen);
images.on('click', function(){
  $(this).css('border', '5px solid blue');
});
```

Existe una notación para definir varios manejadores simultáneos para eventos diferentes utilizando una notación de objeto literal. En este caso no podemos repetir el mismo tipo de evento.

```
images.on({
  mouseenter: function(){
    $(this).css('border', '5px solid red');
  },
  mouseleave: function(){
    $(this).css('border', 'none');
  }
});
```

El método `off()` nos permite desactivar manejadores. Podemos pasar como argumento el tipo de evento, o hacerlo más específico con el tipo de evento y el manejador en concreto.

```
images.off('click', imageFullScreen);
```

En ambos métodos podemos hacer un filtrado más fino de los objetos seleccionados, pasando una selección de estos. Además, también podemos pasar objetos con información para ser utilizados en el manejador como veremos posteriormente.

4.2.2. Ejecutar el manejador una sola vez

El método `one()` nos permite que en lugar de ejecutar un manejador n veces, solamente lo hagamos una vez. La siguiente vez que se dispare el evento, el manejador no lo capturará. En base al ejemplo anterior sustituimos `on()` por `one()`.

```
images.one('click', imageFullScreen);
images.one('click', function(){
  $(this).css('border', '5px solid blue');
});
```

4.2.3. Pasar datos en el evento

El objeto evento tiene una propiedad `data` que nos sirve para pasar información al manejador de eventos de forma personalizada. Al declarar el manejador, el método acepta un argumento opcional que puede ser de cualquier tipo, que es asociado con la propiedad `data`. Este objeto puede ser un array para pasar varios objetos de forma simultánea.

Como ejemplo, queremos que al cliquear en la imagen de categorías se muestre la información de qué categoría se trata, y su posición.

Primero vamos a seleccionar los objetos que necesitamos, las categorías, cada una con su `div` respectivo, las imágenes dentro de las categorías, y la capa con el título de cada categoría.

```
let categories = $('#categories > div.row > div');
let images = categories.find('img');
let info = categories.find('div.cat-list-text')
```

Definimos un manejador de eventos para mostrar la información. En él accedemos a la propiedad `data` que está pensada para recibir dos objetos, un objeto DOM y un objeto literal.

```
function showInfo(event){
  let name = $(event.data[0]).find('h3').text();
  alert(`Categoría: ${name} Posición: ${event.data[1].index}`);
  event.stopPropagation();
}
```

Por último, iteramos sobre cada una de las imágenes para asociar el manejador. En este caso debemos iterar porque el objeto será diferente para cada manejador. El objeto para la propiedad `data` es el primer argumento que pasamos a `click()`, y contiene un array con el objeto DOM y el literal con la posición.

```
for (let i = 0; i < images.length; i++){
  $(images[i]).click([info[i], {index: i}], showInfo);
}
```

4.3. Eventos de ratón

Al igual que en DOM, la información más importante que aporta este tipo de eventos es la posición exacta donde se ha disparado el evento. Las propiedades para dar información están en base a coordenadas X e Y utilizando varios puntos de referencia.

- `offsetX` y `offsetY`: Fija el punto de referencia en la esquina superior izquierda del elemento donde se disparó el evento, es decir, el objeto referenciado por `target`. No incluimos el borde del elemento.
- `clientX` y `clientY`: El punto de referencia es la esquina superior izquierda del área del usuario.

- `pageX` y `pageY`: El documento es el punto de referencia, por lo que tenemos también en cuenta el posible scroll que hayamos generado en la página.
- `screenX` y `screenY`: El último punto de referencia es la esquina superior izquierda de la pantalla.

4.3.1. Evento `click`

En el siguiente código añadimos un elemento `DIV` al *banner* de la página, y le asociamos un manejador de eventos para el evento `click` que nos indique las coordenadas para los cuatro tipos de puntos de referencia.

```
let banner = $('.banner');
let div = $('<div></div>').attr('id','references').css('border','5px soli
d red');
banner.after(div);
banner.click(function(event){
  let str = "";
  str = "offsetX: " + event.offsetX + " offsetY: " + event.offsetY + "<br
>";
  str += "clientX: " + event.clientX + " clientY: " + event.clientY + "<b
r>";
  str += "pageX: " + event.pageX + " pageY: " + event.pageY + "<br>";
  str += "screenX: " + event.screenX + " screenY: " + event.screenY + "<b
r>";
  div.html(str);
});
```

4.3.2. Eventos `mouseenter`, `mouseleave` y `mousemove`

Para probar los eventos, vamos a mostrar una capa flotante al situarnos en las imágenes de las categorías de la página. En esta capa mostraremos las dimensiones de alto y ancho que ocupan las imágenes exactamente en la página.

El primer paso es seleccionar los elementos donde vamos a añadir los manejadores de eventos.

```
let categories = $('#categories');
let divImages = categories.find(".cat-list-image");
```

El manejador que nos servirá para mostrar la capa es el siguiente. Debemos notar que la referencia `this` es un objeto DOM, por lo que si queremos trabajar con jQuery lo tenemos que transformar en un objeto jQuery con la función `$`. Para evitar que el ratón se mueve por encima de la capa que hemos creado, a la posición del ratón respecto al padre, le sumaremos 5 píxeles.

```
function showDimensions(event){
  let dimensionsDiv = $('<div></div>')
    .addClass('border border-primary p-2')
    .css({
      background: '#f5f5f5',
      width: '150px',
      position: 'absolute',
      zIndex:'1',
      top: event.offsetY + 5 + 'px',
      left: event.offsetX + 5 + 'px',
```

```
    });

  let str = "offsetWidth: " + $(this).innerWidth() + "<br>" + "offsetHeig
ht: " + $(this).innerHeight();
  dimensionsDiv.html(str);
  $(this).append(dimensionsDiv);
}
```

La segunda función la utilizaremos para eliminar el elemento. De los hijos, borramos el segundo elemento que corresponde a la capa filtrándolo en el método. El primero es la imagen.

```
function hideDimensions(event){
  $(this).children().remove(':nth-child(2)');
}
```

El tercer manejador nos servirá para mover la capa con el ratón. Nuevamente, para evitar que el ratón toque la capa y nos altere las posiciones, la desplazamos 5 píxeles.

```
function moveDimensions(event){
  event.stopPropagation();
  event.preventDefault();
  $(this).children(':nth-child(2)').css({
    top: event.offsetY + 5 + 'px',
    left: event.offsetX + 5 + 'px'
  });
}
```

Por último, indicamos la posición relativa para el `div` contenedor de las imágenes, y le asignamos los manejadores a los eventos respectivos.

```
divImages.css('position', 'relative');
divImages.on({
  mouseenter: showDimensions,
  mouseleave: hideDimensions,
  mousemove: moveDimensions
})
```

4.3.3. Eventos `mouseover` y `mouseout`

Para todos los elementos de la página, queremos mostrar un borde cuando el ratón pase por encima de ellos, y lo elimine cuando salga. Añadimos los eventos `mouseover` y `mouseout` al elemento `BODY`, y añadimos las clases con los bordes al `target` del evento, con la precaución de transformarlo en un objeto jQuery previamente.

```
$('body').on({
  mouseover: function(event){
    $(event.target).addClass('border border-danger');
  },
  mouseout: function(event){
    $(event.target).removeClass('border border-danger');
  }
});
```

4.3.4. Eventos de foco

En el formulario de envío de correo electrónico para la suscripción, vamos a mostrar un mensaje de texto al situarnos en el `INPUT` que recoge el email, y eliminar el mensaje al cambiar de elemento por perder el foco.

```
let input = $('input[name = "email"]');
input.on({
  focus: function(){
    let div = $('<div></div>');
    $(this).parent().append(div);
    div.text ('Introduce un correo electrónico.');
  },
  blur: function(){
    $(this).next().next().remove();
  }
});
```

4.3.5. Evento `hover`

Es una combinación de `mouseenter` y `mouseleave`. Vamos a poner un borde a las imágenes de las categorías utilizando clases para verlo como ejemplo. El método `hover()` admite dos argumentos como manejador, el primero funcionaría como `mouseenter`, y el segundo como `mouseleave`.

```
let categories = $('#categories');
let divImages = categories.find(".cat-list-image");

$( divImages ).hover(
  function() {
    $(this).addClass('border border-danger');
  }, function() {
    $(this).removeClass('border border-danger');
  }
);
```

Si el método recibe solamente un argumento con un manejador, este manejador será invocado tanto al entrar como al salir, por lo que el manejador debe estar preparado para ambos casos.

```
let categories = $('#categories');
let divImages = categories.find(".cat-list-image");

$( divImages ).hover(
function() {
  $(this).toggleClass('border border-success');
});
```

4.4. Eventos de teclado

Los tipos de evento de teclado son al pulsar una tecla `keydown` y al liberarla `keyup`.

Las dos propiedades que vamos a utilizar son `key` con la tecla que ha sido pulsada, y `code` con el código asociado a la tecla.

4.4.1. Evento keydown

Vamos a reproducir cada carácter y la tecla pulsada al escribir en el elemento `input` para enviar el correo de suscripción. Seleccionamos el elemento, y creamos un `DIV` para mostrar el resultado.

```
let input = $('input[name = "email"]');
let div = $('<div></div>');
input.parent().append(div);
input.focus();
input.keydown(function(event){
  let div = $(this).next().next();
  div.text(div.text() + `${event.key}(${event.code}) `);
});
```

4.4.2. Evento keyup

Vamos a mostrar la capan con las dimensiones de las imágenes de las categorías y a ocultarlas. En función de la tecla numérica pulsada, y en combinación con la tecla ALT, mostraremos las dimensiones en base a su posición.

Seleccionamos las imágenes con las que vamos a trabajar como hicimos en el ejercicio anterior.

```
let categories = $('#categories');
let divImages = categories.find('.cat-list-image');
```

Asociamos los eventos en el objeto `document`. Al recuperar un objeto de una colección jQuery, lo que obtenemos es un objeto DOM. Si queremos seguir trabajando con jQuery hay que transformarlo a jQuery.

```
$(document).on({
  keydown: function (event) {
    if (event.altKey) {
      if (event.code.indexOf("Numpad") > -1 ||
        event.code.indexOf("Digit") > -1) {
        let number = (event.code.length === 7) ?
          event.code.substring(6) :
          event.code.substring(5);
        number = +number;

        let divImage = $(divImages[number]);
        if (number < divImages.length && divImage.children().length < 2)
{
          let dimensionsDiv = $('<div></div>')
            .addClass('border border-primary p-2')
            .css({
              background: '#f5f5f5',
              width: '150px',
              position: 'absolute',
              top: '0px',
              left: '0px'
            });
```

```
          let str = "offsetWidth: " + divImage.innerWidth() + "<br>" + "o
ffsetHeight: " + divImage.innerHeight();
          dimensionsDiv.html(str);
          divImage.css ('position', 'relative');
          divImage.append(dimensionsDiv);
        }
      }
    }
  },
  keyup: function (event) {
    if (event.altKey) {
      if (event.code.indexOf("Numpad") > -1 ||
        event.code.indexOf("Digit") > -1) {
        let number = (event.code.length === 7) ?
          event.code.substring(6) :
          event.code.substring(5);
        number = +number;
        if (number < divImages.length) {
          $(divImages[number]).children(':nth-child(2)').remove();
        }
      }
    }
  }
});
```

5. Ejercicios

5.1. El gordo de la Navidad

Genera un número aleatorio correspondiente al número de la lotería de Navidad. A través de `input` de tipo `number`, intentamos adivinar el número. Utilizamos un código de color para saber si el número es el correcto.

- Color rojo, el valor está por debajo del número.
- Color azul, el valor está por encima del número.
- Color verde, el valor es el correcto.

Solución

Creamos una estructura HTML con los elementos que necesitamos para interactuar con el usuario. Básicamente tenemos dos botones uno de comienzo y otro para saber si hemos acertado el número. Necesitamos 5 elementos `input` para calcular el número.

```
<!-- Lotería -->
<div id="loto">
    <div class="jumbotron jumbotron-fluid d-flex">
        <div
            class="text-dark container d-flex flex-column justify-
content-center align-items-sm-stretch align-items-md-center">
```

```
            <h1 id="message" class="col-sm-
12">Adivina el gordo de Navidad</h1>
            <button type="button" id="b-start" class="btn btn-
primary">Comenar</button>
            <div class="p-
3" style="display: none" id="output">Resultado</div>
            <div id="game" class="container" style="display:none">
                <div id="ciphers" class="row justify-content-center m-3">
                    <div class="col-1 m-
2"><input type="number" id="num1" min="0" max="9" value="0"></div>
                    <div class="col-1 m-
2"><input type="number" id="num2" min="0" max="9" value="0"></div>
                    <div class="col-1 m-
2"><input type="number" id="num3" min="0" max="9" value="0"></div>
                    <div class="col-1 m-
2"><input type="number" id="num4" min="0" max="9" value="0"></div>
                    <div class="col-1 m-
2"><input type="number" id="num5" min="0" max="9" value="0"></div>
                </div>
                <div class="row justify-content-center m-3">
                    <button id="b-guess" type="button" class="btn btn-
primary">Adivinar</button>
                </div>
                <p>Reglas</p>
                <ul>
                    <li>Color rojo, el valor está por debajo del número.<
/li>
                    <li>Color azul, el valor está por encima del número.<
/li>
                    <li>Color verde, el valor es el correcto.</li>
                </ul>
            </div>
        </div>
    </div>
</div>
```

Creamos una estructura Singleton para generar el número aleatorio, mantenerlo y recuperarlo en forma de array para que podamos compararlo con los `input` de la página.

```
let Prize = (function () {
  let instantiated;

  function init() {
    let number = null;
    return { // Devuelve el objeto que será único.
      setNumber: function () {
        number = Math.floor(Math.random() * 100000);
      },
      getNumber: function (){
```

```
        let ciphers = [];
        let tmpNumber = number;
        for (let i = 4; i >= 0; i--){
          ciphers[i] = tmpNumber % 10;
          tmpNumber = Math.floor(tmpNumber/10);
        }
        return ciphers;
      }
    };
  }
  return {
    getInstance: function () {
      if (!instantiated) {
        instantiated = init();
      }
      return instantiated;
    }
  };
})();
```

Definimos el manejador del botón de inicio, en él generamos el número ganador y damos acceso al interfaz para que el usuario pueda interactuar.

```
$('#b-start').click(function () {
  Prize.getInstance().setNumber();
  $('#game').show();
  $(this).attr('disabled', 'true');
  let ciphers = $('#ciphers').find('input');
  $(ciphers).css('background', 'white');
  $('#ciphers').find('input').get(0).focus();
  $('#b-guess').removeAttr('disabled');
  alert(Prize.getInstance().getNumber());
});
```

Vamos a utilizar una función auxiliar que compare números recogidos de la página, con las cifras del número. La función devuelve el color de cada uno de los `input`.

```
function checkNumber(cipher, num){
  //alert(num + " " + cipher);
  if (cipher < num) return 'red';
  if (cipher > num) return 'blue';
  if (cipher === num) return 'green';
}
```

Por último, creamos un manejador para el botón de adivinar para saber si los `input` corresponden con las cifras del número. Tenemos una variable booleana que nos permite saber si todos los números son correctos.

```
$('#b-guess').click(function(){
  let winning = true;
  let ciphers = $('#ciphers').find('input');
  let color;
```

```
  let numbers = Prize.getInstance().getNumber();
  for (let i = 0; i < 5; i++){
    console.log(numbers);
    color = checkNumber(Number.parseInt($(ciphers[i]).val()), numbers[i])
;
    if (color !== 'green') winning = false;
    $(ciphers[i]).css('background', color);
  }
  if (winning) {
    $('#output').text('El número ganador es: ' + numbers.join('')).show()
;
    $(this).attr('disabled', 'true');
    $('#b-start').removeAttr('disabled');
  } else {
    $('#ciphers').find('input').get(0).focus();
  }
});
```

5.2. Buscador

Creamos una tabla con datos tabulares inventados en HTML. Los datos pueden ser sobre música, deportes, economía, cine, etc. La tabla debe tener entre 3 y 4 columnas. Con una caja de texto, tenemos que hacer un buscador que filtre las filas que no se adapten al patrón introducido en la caja de texto. Como evento, debemos capturar `keyup` para mejorar la usabilidad de la página.

Solución

Generamos la tabla en HTML.

```
<!-- Buscador -->
<div id="search" style="display: block">
    <div class="jumbotron jumbotron-fluid d-flex">
        <div
            class="text-dark container justify-content-center">
            <h2>Máximos anotadores NBA</h2>
            <div class="row justify-content-end">
                <input id="b-search" type="text" placeholder="Buscar">
            </div>
            <table class="table table-striped">
                <thead>
                    <tr>
                        <th>Jugador</th>
                        <th>Equipo</th>
                        <th>Posición</th>
                        <th>Media</th>
                    </tr>
                </thead>
                <tbody id="players">
                    <tr>
                        <td>Donovan Mitchell</td>
```

```
                    <td>UTAH</td>
                    <td>Escolta</td>
                    <td>33.85</td>
                </tr>
                <tr>
                    <td>Luka Doncic</td>
                    <td>Dallas</td>
                    <td>Base</td>
                    <td>26.98</td>
                </tr>
                <tr>
                    <td>James Harden</td>
                    <td>Houston</td>
                    <td>Alero</td>
                    <td>26.76</td>
                </tr>
                <tr>
                    <td>Joel Embiid</td>
                    <td>Philadelphia</td>
                    <td>Pivot</td>
                    <td>25.72</td>
                </tr>
                <tr>
                    <td>Kawhi Leonard</td>
                    <td>Clippers</td>
                    <td>Alero</td>
                    <td>25.70</td>
                </tr>
            </tbody>
        </table>
    </div>
</div>
</div>
```

Asignamos un manejador al `input` para introducir la cadena de búsqueda. Por cada pulsación de tecla tenemos que filtrar la filas de `tr` para mostrarlas u ocultarlas. Utilizamos una función que nos permita decidir en base al texto que contenido en la fila.

Como ampliación, comprobamos las filas que están oculta en base a la propiedad `display` que haya sido añadida. Si todas las filas están ocultas, mostramos una capa con el texto indicando que no hay emparejamiento con la cadena de búsqueda.

```
$('#b-search').on('keyup', function() {
  let value = $(this).val().toLowerCase();
  let rows = $('#players tr');
  rows.filter(function() {
    $(this).toggle($(this).text().toLowerCase().indexOf(value) > -1);
  });

  let hideRows = $('#players tr[style = "display: none;"]');
```

```
  if (hideRows.length === rows.length){
    if ($('#players').parent().next().length === 0)
      $('#players').parent().after($('<div>No hay registros emparejados a
l criterio de búsqueda.</div>'));
  } else {
    $('#players').parent().next().remove();
  }
});
```

6. Efectos y animaciones

Con jQuery podemos aplicar varias técnicas de animación en nuestras páginas. Tenemos disponibles de las más sencillas, con movimientos prediseñados por la librería, hasta las más sofisticadas, donde podemos diseñar el tipo de animación que queremos. Una de las ventajas que tendremos es que podemos encadenar todos los tipos de animaciones, para que se ejecuten de forma secuencial sobre el conjunto de objetos que estamos aplicando.

6.1. Efectos básicos

Ya hemos mencionado estos efectos. Son los métodos que nos permiten mostrar un elemento, `show()`, y ocultarlo, `hide()`. Ambos elementos trabajan en base a la propiedad `display` de CSS. Como complemento a ambos métodos tenemos `toggle()`, que altera el comportamiento en función del estado actual del elemento, o lo muestra o lo oculta.

Como veremos a lo largo de este punto, podemos pasarle al efecto una serie de parámetros de forma opcional. El primer parámetro es el tiempo o la velocidad a la que se ejecutará el efecto. Es más útil trabajar por tiempo, por lo que le pasaremos el número de milisegundos que deben transcurrir en efectuarse el efecto.

Veamos un ejemplo. Vamos a añadir tres botones a cada una de las categorías para mostrar u ocultarlas. Primero Creamos los botones.

```
let categories = $('#categories > div.row > div');
let bShow = $('<button class="btn btn-primary m-
1"></button>').text('Show');
let bHide = $('<button class="btn btn-primary m-
1"></button>').text('Hide');
let bToggle = $('<button class="btn btn-primary m-
1"></button>').text('Toggle');
```

Asociamos al evento `click` a cada botón, y buscamos el *div.cat-list-image* para mostrarlo u ocultarlo. A cada efecto le hemos indicado el tiempo que debe transcurrir para que se lleve a cabo el efecto.

```
bShow.click(function (){
  let img = $(this).parent().find('div.cat-list-image').first();
  img.show(1000);
});

bHide.click(function (){
  let img = $(this).parent().find('div.cat-list-image').first();
  img.hide(2000);
});
```

```
bToggle.click(function (){
  let img = $(this).parent().find('div.cat-list-image').first();
  img.toggle(3000);
});
```

Por último, añadimos los botones a cada una de las categorías.

```
categories.prepend(bShow, bHide, bToggle);
```

6.2. Funciones de callback

Al invocar uno de los efectos, el intérprete no se queda bloqueado hasta la finalización del efecto para que pueda continuar con la ejecución del código de manera secuencial. Para evitar esta circunstancia, el intérprete trabaja en base de hilos, generador un hilo de ejecución para el evento, y otro hilo de ejecución para el efecto o las posibles animaciones.

Podemos añadir una función a la invocación del efecto para que sea llamada justo cuando el efecto ha finalizado. Al tener dos hilos de ejecución, la única forma de garantizar que se ejecute la función al finalizar el efecto es pasarla como argumento.

Vamos a modificar el código anterior para habilitar o deshabilitar los botones, en función del estado en el que se encuentra el elemento a mostrar u ocultar. Primero vamos a deshabilitar el botón de *mostrar*, ya que por defecto el elemento está mostrado.

```
bShow.attr('disabled', true);
```

Modificamos el efecto `show()` en el evento `click` del botón de *mostrar* para que se deshabilite el botón y habilitar el de *ocultar* cuando finalice el efecto.

```
img.show(1000, function(){
  button.attr('disabled', true);
  button.next().removeAttr('disabled');
});
```

Repetimos la funcionalidad inversa para el botón de *ocultar*.

```
img.show(1000, function(){
  button.attr('disabled', true);
  button.next().removeAttr('disabled');
});
```

Por último, en el botón de *toggle* vamos habilitamos y deshabilitamos los botones en base del estado actual.

```
img.toggle(3000, function(){
  if(button.prev().attr('disabled')){
    button.prev().removeAttr('disabled');
    button.prev().prev().attr('disabled',true);
  } else {
    button.prev().attr('disabled', true);
    button.prev().prev().removeAttr('disabled');
  }
});
```

6.3. Efectos `slide`

Los efectos `slide` muestran u ocultan los elementos de arriba hacia abajo y viceversa en función de las alturas de los elementos. Modificamos los efectos para mostrar el elemento de arriba hacia abajo utilizando el método `slideDown()`.

```
img.slideDown(1000, function(){
  button.attr('disabled', true);
  button.next().removeAttr('disabled');
});
```

Para ocultar el elemento lo haremos de abajo hacía arriba utilizando el método `slideUp()`.

```
img.slideUp(2000, function(){
  button.attr('disabled', true);
  button.prev().removeAttr('disabled');
});
```

Como en el caso anterior, tenemos el efecto que nos muestra o nos oculta el elemento en función del estado actual del elemento con `slideToggle()`.

```
img.slideToggle(3000, function(){
  if(button.prev().attr('disabled')){
    button.prev().removeAttr('disabled');
    button.prev().prev().attr('disabled',true);
  } else {
    button.prev().attr('disabled', true);
    button.prev().prev().removeAttr('disabled');
  }
});
```

6.4. Efectos `fade`

Este tipo de efectos muestran u ocultan los elementos en función de visibilidad. El efecto `fadeIn()` muestra el elemento desde dentro hacía afuera.

```
img.fadeIn(1000, function(){
  button.attr('disabled', true);
  button.next().removeAttr('disabled');
});
```

El efecto contrario es `fadeOut()` que oculta el elemento desde fuera hacía adentro.

```
img.fadeOut(2000, function(){
  button.attr('disabled', true);
  button.prev().removeAttr('disabled');
});
```

Como con los efectos anteriores, tenemos el método `fadeToggle()` que muestra u oculta el elemento en función de su estado.

```
img.fadeToggle(3000, function(){
  if(button.prev().attr('disabled')){
    button.prev().removeAttr('disabled');
    button.prev().prev().attr('disabled',true);
  } else {
```

```
    button.prev().attr('disabled', true);
    button.prev().prev().removeAttr('disabled');
  }
});
```

Por último, este tipo de efectos tienen la posibilidad de llevar la opacidad del elemento a un valor determinado entre 0 y 1. El método `fadeTo()` realiza esta operación recibiendo el argumento del valor de la opacidad que queremos aplicar al elemento. A través de los eventos `mouseenter` y `mouseleave` vamos a modificar la opacidad del elemento.

```
categories.find('div.cat-list-image').mouseenter(function (){
  $(this).fadeTo(1000, 0.5);
});

categories.find('div.cat-list-image').mouseleave(function (){
  $(this).fadeTo(1000, 1);
});
```

6.5. Animaciones personalizadas

Hasta ahora hemos trabajado con efectos prediseñados. Podemos crear nuestros propios efectos a través de la animación de elementos utilizando el método `animate()`. Las animaciones funcionan en base a la modificación de las propiedades CSS del elemento. En las animaciones podemos modificar básicamente las propiedades numéricas, por ejemplo, podemos modificar el tamaño de la fuente de un elemento, sus dimensiones, su posición en la página, tamaño de bordes, etc, pero no podemos cambiar propiedades como los colores de fuente, fondo o de los bordes.

6.5.1. Animaciones encadenadas

Como ejemplo vamos a mover la primera categoría a la última posición del listado de la página. En primer lugar, vamos a seleccionar los `DIV` con las categorías y al trabajar con posicionamiento, tomaremos como referencia el `DIV` contenedor de las categorías, por lo que aplicamos una posición relativa en ese elemento, además creamos un botón que nos permita disparar la animación a nuestro antojo.

```
let container = $('#categories');
let categories = $('#categories > div.row > div');
container.css('position', 'relative');
let button1 = $('<button class="btn btn-primary m-
1"></button>').text('Animate');
container.prepend(button1);
```

Vamos a calcular las ubicaciones exactas en base al contenedor de la primera y de la última categoría. En base a estas referencias, podremos hacer los movimientos.

```
let first = categories.first();
let last = categories.last();

let coordinates = {
  first: {
    left: first.get(0).offsetLeft,
    top: first.get(0).offsetTop
  },
  last: {
    left: last.get(0).offsetLeft,
    top: last.get(0).offsetTop
  }
}
```

Pasamos a asignar el evento `click` al botón. Primero vamos a modificar la posición de la primera categoría para que se absoluta, lo que nos permite que podamos ubicar su posición en base al contenedor con posición relativa.

Al método `animate()` le pasamos como argumento un objeto literal con las modificaciones de las propiedades CSS que queremos hacer, en este caso posiciones `left` y `top` en base a las calculadas anteriormente, y opcionalmente el tiempo para ejecutar la animación. Existen otra sintaxis, pero nos vamos a centrar en el objeto literal. Como podemos observar, no es necesario indicar la medida en pixeles, como si ocurre en CSS, basta con el valor absoluto que queremos asignar.

```
button1.click(function (){
  first.css({position: 'absolute', zIndex: '1', border: '10px solid red'}
);
  first
    .animate({left: coordinates.last.left, top: coordinates.last.top});
})
```

Como comentamos al comienzo del punto, podemos encadenar animaciones y efectos sobre un elemento. Vamos a ejecutar el movimiento de la primera categoría a la última en tres pasos:

- Primer paso: Situamos la posición de la categoría de forma relativa a su posición actual utilizando los operadores += y −=. Modificamos también la opacidad.
- Segundo paso: movemos en el eje X la categoría para acercarla en el destino.
- Tercer paso: ubicamos la categoría en la posición final y restauramos la opacidad.

```
button1.click(function (){
  first.css({position: 'absolute', zIndex: '1', border: '10px solid red'}
);
  first
    .animate({left: '+=200', top: '-=200', opacity: '0.5',}, 1000)
    .animate({left: coordinates.last.left - 200}, 1000)
    .animate({left: coordinates.last.left, top: coordinates.last.top, opa
city: '1',}, 1000);
})
```

6.5.2. Detener animaciones

Las animaciones anidadas se gestionan a través de una cola con las animaciones, sobre la cual tenemos cierto control. El método `stop()` sobre el elemento que ejecutamos la animación detiene la ejecución de la animación, aunque existen otros métodos para realizarlo.

Creamos un botón para detener la animación y lo añadimos a la página.

```
let button2 = $('<button class="btn btn-primary m-
1"></button>').text('Stop');
container.prepend(button1, button2);
```

En el manejador del evento `click` del botón, detenemos la animación. Al método `stop()` podemos pasarle un booleano. Por defecto el valor es `false`, lo que implica que la animación actual se detiene, pero el resto de las animaciones de la cola se ejecutarán. Si le aplicamos el valor `true`, la animación se detiene y se cancelan el resto de las animaciones puestas en la cola.

```
button2.click(function (){
  first.stop(true);
});
```

6.5.3. Opciones de la animación

Para implementar el dinamismo en las animaciones, la ejecución de cada animación se hace en pequeños pasos en función del tiempo asociado a la animación. Podemos tener control de cada uno de estos pasos, así como a la configuración de cada animación utilizando un objeto literal. Este objeto admite una gran cantidad de propiedades, pero vamos a ver algunas de las más importantes aplicadas a las animaciones del ejemplo.

- `duration`: Permite configurar el tiempo de duración de la animación. Lo hacemos a partir de una variable definida previamente.
- `start`: es una función invocada antes del comienzo de la animación. Aquí vamos a modificar las propiedades CSS de contenedor de la categoría para ponerle una posición absoluta, el apilamiento lo ponemos en primer plano, y le añadimos un borde.
- `done`: se trata de una función que será invocada una vez que la animación haya finalizado. Añadimos esta propiedad como opción en las dos primeras animaciones para cambiar el color del borde de la categoría.

```
button1.click(function (){
  let duration = 1000;
  first
    .animate({left: '+=200', top: '-=200', opacity: '0.5',},{
      duration: duration,
      start: function(){
        $(this).css({position: 'absolute', zIndex: '1', border: '10px sol
id red'});
      },
      done: function(){
        first.css({border: '10px solid blue'});
      }
    })
    .animate({left: coordinates.last.left - 200},{
      duration: duration,
      done: function(){
        first.css({border: '10px solid green'});
      }
    })
    .animate({left: coordinates.last.left, top: coordinates.last.top, opa
city: '1',},{
      duration: duration,
    });
})
```

Vamos a ampliar la funcionalidad añadiendo una barra de progreso. La barra consta de un contenedor y una capa que iremos modificando el tamaño para mostrar la evolución de la animación. Al comienzo del evento creamos los elementos de la barra de progreso.

```
let progress = $('#progress');
if (progress.length === 0){
    let progressBar = $('<div class="row border border-primary mt-
2"><div id="progress"></div></div>');
    container.append(progressBar);
    progress = $('#progress');
    progress.css({
        height: '40px',
        color: '#ffffff',
        fontSize: '2em',
        textAlign: 'center',
    });
}
```

En la propiedad `start` modificamos el estilo de la barra de progreso. Le asociamos un tamaño de 0 y un color de fondo rojo.

```
progress.css({
    background: 'red',
    width: '0%'
});
```

Siguiendo en la misma función, asignamos una animación a la barra de progreso que lleve su tamaño hasta el 100%. En esta animación le asignamos como opciones varias propiedades.

- `duration`: Es la misma que la duración de la animación principal completa.
- `easing`: por defecto esta propiedad tiene el valor de *swing*, lo que implica que la animación se ejecuta en fases, comienza despacio, se va acelerando en el medio, y finaliza de forma suave. Le vamos a asociar un valor *linear* para que su evolución sea constante.
- `step`: Cada animación es dividida en una serie de pasos, en cada uno de los cuales se irán alterando las propiedades asignadas en la animación para crear el efecto dinámico. Si asignamos una función en esta propiedad, será ejecutada con cada paso y propiedad de la animación. La función recibe un argumento con el valor actual de la propiedad, y el segundo es un objeto que nos permite identificar la propiedad. Vamos a modificar el texto de la barra de progreso a cada paso, en función del tamaño actual de la barra.

```
progress.animate({width: '100%'}, {
    duration: duration * 3,
    easing: 'linear',
    step: function(width, fx){
        $(this).text(width.toFixed(2) + "%");
    },
});
```

También cambiamos el color de la barra de progreso en la ejecución de la función `done()`. En la primera animación añadimos:

```
progress.css({background: 'blue'});
```

En la segunda función:

```
progress.css({background: 'green'});
```

Por último, debemos detener las dos animaciones.

```
button2.click(function (){
    first.stop(true);
    $('#progress').stop();
});
```

6.5.4. Ejercicio animaciones

Transforma el ejemplo de anterior para que independientemente de la categoría que tengamos en la primera posición, al ejecutar la animación la primera categoría pase a la última posición, y la segunda, tercera y cuarta categorías, también a través de una animación, se mueven una posición a la izquierda. Si volvemos a ejecutar la animación, la categoría mostrada en primera posición deberá pasar a la última, moviendo una posición a la izquierda el resto de las categorías, pudiendo de esta manera poder ejecutar un bucle infinito.

Solución

Cambiamos la posición relativa al primer `DIV.row` que contiene las categorías. También le asignamos la propiedad `height` de CSS el valor actual de este `DIV`, para dejarlo como tamaño absoluto fijo. Esto lo hacemos porque cada `DIV` de las categorías pasarán a tener posicionamiento absoluto, y debemos mantener el espacio.

```
container.children().first().css('position', 'relative');
container.children().first().height(container.height());
```

Podemos eliminar las variables *first* y *last*, porque las pasaremos al manejador de eventos del botón.

Ahora necesitamos las coordenadas de las 4 categorías, por lo que utilizaremos un array para contenerlas. Además, una vez calculadas las posiciones, las fijamos a los elementos mediante propiedades CSS y le asignamos un posicionamiento absoluto.

```
let coordinates = [];
for (let img of categories){
    coordinates.push({
        left: img.offsetLeft,
        top: img.offsetTop
    });
}
for (let i = 0; i < categories.length; i++){
    $(categories[i]).css({
        position: 'absolute',
        left: coordinates[i].left,
        top: coordinates[i].top
    });
}
```

La primera acción del manejador guarda referencias de la primera y la última categoría, ya que irán cambiando en función de las diversas ejecuciones de la animación. También queremos deshabilitar el propio botón mientras se ejecuta la animación, y guardamos una referencia para que podamos acceder a él más adelante. Tenemos que notar que recalculamos en cada animación la colección de categorías, esto es debido a que la colección se modifica en cada ejecución.

```
let categories = $('#categories > div.row > div');
let first = categories.first();
let last = categories.last();
let button = $(this);
button.attr('disabled', true);
```

En la función `start()` la modificamos para añadir las animaciones a cada una de las categorías que no forman parte de la animación principal, para que se muevan un espacio en base a las coordenadas calculadas en el inicio.

```
let siblings = $(this).siblings();
for (let i = 0; i < siblings.length; i++){
    $(siblings[i]).animate({
        left: coordinates[i].left
    }, duration);
}
```

Modificamos la segunda animación para que la posiciones se calcule en base al array de coordenadas.

```
.animate({left: coordinates[coordinates.length-1].left - 200},{
  duration: duration,
  done: function(){
      first.css({border: '10px solid green'});
      progress.css({background: 'green'});
  }
})
```

En la tercera animación, asignamos la posición en base al array de coordenadas. Además, eliminamos el borde al asignarle un tamaño de 0px. Por último, en la función `done()`, habilitamos el botón borrando el atributo `disabled`, restauramos la propiedad zIndex para que esté alineada con el resto de categorías, y por último y más importante, movemos el DIV contenedor de la categoría en el árbol DOM para que sea el último, de esta forma habilitamos que la posición visual de los elementos coincida con la del árbol DOM, facilitando la siguiente ejecución de la animación.

```
.animate({
    left: coordinates[coordinates.length-1].left,
    top: coordinates[coordinates.length-1].top,
    borderWidth: '0px',
    opacity: '1',},{
  duration: duration,
  done: function(){
    button.removeAttr('disabled');
    $(this).parent().append($(this));
    $(this).css({zIndex: '0'});
  }
});
```

Por último, en el botón de parado, detenemos todas las animaciones de la colección de categorías de forma simultánea, y volvemos a activar el botón de la animación.

```
button2.click(function (){
  $('#progress').stop(true);
  categories.stop(true);
  $(this).prev().removeAttr('disabled');
});
```