UNIDAD 11. Tecnologías actuales: jQuery

Desarrollo Web en Entorno Cliente

2º DAW

Contenidos

1. Introducción. Tecnologías actuales

Durante las unidades anteriores hemos estado trabajando con lo que se conoce como **Vanilla JavaScript** o **JavaScript puro**, es decir, el uso del lenguaje sin el apoyo de alguna librería o *framework*.

JavaScript ha evolucionado muchísimo desde sus inicios y desde su versión ES6 es más fácil que nunca hacer todo lo que se hacía necesariamente a través de librerías, pero prescindiendo de ellas. Además, teniendo una base muy bien asentada, es mucho más fácil resolver los problemas que puedan surgir mientras estamos usando un *framework* o una librería.

Elegir un *framework* o librería de JavaScript

A la hora de programar nos encontramos con miles de recursos que nos permiten desarrollar aplicaciones utilizando herramientas con características similares y que pueden dar pie a confusiones. Por eso, vamos a comenzar distinguiéndolas:

- **Librería** o biblioteca: es una colección de clases o funciones que podemos utilizar desde nuestro propio código con el fin de poder realizar acciones más rápida y fácilmente.
- **API** (*Application Programming Interfaces*, Interfaz de Programación de Aplicaciones): es la interfaz de las funciones o métodos de una librería o biblioteca, su implementación interna está oculta al público.
- ***Framework***: es un entorno de desarrollo completo que suele facilitar herramientas tan indispensables como el compilador, el depurador y el editor de código, además de un gran conjunto de bibliotecas previamente implementadas. Pueden incorporar también un editor visual.
- **SDK** (*Software Development Kit*, Kit de Desarrollo de Software): concepto es similar a *framework*, pero con la diferencia de que se orienta al desarrollo de aplicaciones exclusivas para una plataforma en particular (sistema operativo, navegador, etc.).
- **IDE** (*Integrated Development Environment*, Entorno de Desarrollo Integrado): es un editor de texto mejorado, con soporte adicional para el desarrollo, compilación y depuración de aplicaciones.
- ***Toolkit***: con este término podemos englobar librerías e incluso *frameworks*, ya que suele usarse para aludir a un conjunto de herramientas de desarrollo que facilita una empresa o una comunidad de software libre, y que incluyen formatos prediseñados de botones, tablas, cajas de diálogo, etc.

Ventajas e inconvenientes del uso de *frameworks*

Ventajas:

1. Mejoran la productividad, así como la claridad y mantenibilidad del código, permitiendo el uso de funciones sin necesidad de conocer en profundidad el código (abstracción).
2. Es un código gratuito y abierto, con buena documentación.

3. Reutilización de código creado por otros grandes programadores, posiblemente más efectivo y conciso que el que nosotros seríamos capaces de crear.

Inconvenientes:

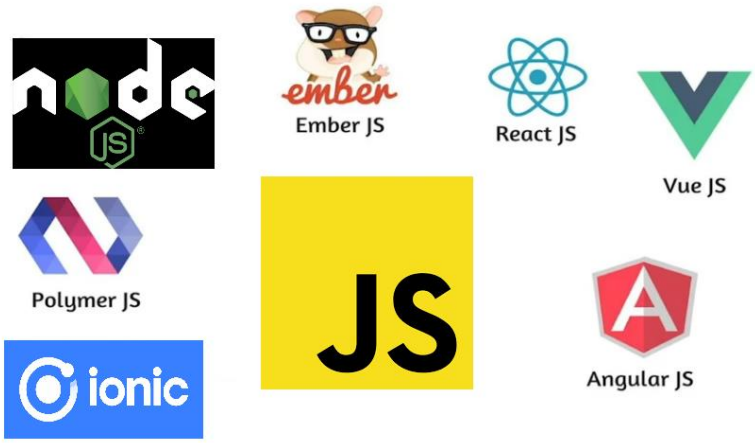
1. Los *frameworks* evolucionan, e incluso en algunos casos dejan de ser mantenidos y, por lo tanto, dejan de usarse.
2. Algunos *frameworks* son incompatibles entre sí, y usar dos de ellos al mismo tiempo puede generar muchos problemas.
3. Aprender a usar un *framework* es algo limitado a algunas tareas determinadas, mientras que aprender un lenguaje de programación es algo universal que tendrá múltiples aplicaciones.
4. Ciertos *frameworks* pueden generar sobrecarga para el navegador.

Ejemplos actuales de *frameworks* y otras utilidades para JavaScript

Muchos de los ejemplos que vamos a ver, están basados en ***WebComponents***, también llamados componentes web, que son una especificación de la W3C para componer la web y reutilizar elementos que usamos en nuestro día a día como formularios, inputs, tablas, encabezados, etc.

Veamos algunos ejemplos actuales:

- **jQuery**: librería que se popularizó en una época dominada por Flash, ya que se presentaba como una alternativa para darle vida a nuestras páginas sin necesidad de un plugin tan pesado. Nos permite manipular de manera sencilla el DOM, crear animaciones, manejar eventos y utilizar AJAX.
- **React**: librería desarrollada por Facebook, que dispone de una gran velocidad de renderizado, ya que emplea un DOM Virtual para ello. Se enfoca en la creación de interfaces de usuario utilizando JavaScript, HTML y CSS, pero todo gestionado desde JavaScript. Está basado en componentes web y enfocado a las SPA (*Single Page Applications*).
- **Vue.js**: *framework* basado en componentes y enfocado a las SPA (al igual que React y Angular). Se autodenomina *framework* progresivo, porque podemos ir utilizando las partes de las librerías que necesitemos. React y Angular cuentan con una organización parecida, lo que diferencia a Vue.js de ellos, es lo bien desacoplados que se encuentran sus componentes, lo que permite que sigan trabajando bien, aunque se incluyan módulos nuevos. Se ha hecho muy popular por su leve curva de aprendizaje.



- **Angular**: *framework* desarrollado en TypeScript y mantenido por Google. Al igual que React y Vue.js, está basado en componentes y enfocado a las SPA. El principal objetivo de Angular es aumentar las aplicaciones web basadas en el modelo vista controlador (MVC) con el fin de hacer que el desarrollo y las pruebas sean más sencillos.
- **Ember.js**: otro *framework* de la familia MVC, muy similar a Backbone.js. Se caracteriza por ser adaptable y flexible. Aunque está considerado como un *framework* para la web, también posibilita crear aplicaciones de escritorio y móviles.
- **Polymer**: librería de Google para acercar el desarrollo de *WebComponents* a todos los navegadores.
- **Backbone.js**: es un *framework* extremadamente ligero comparado con otros, y permite utilizar el sistema de plantillas que quieras, otras librerías, etc.
- **Node.js**: *framework* para trabajar con JavaScript del lado del servidor. La web es renderizada desde el servidor (bueno para el SEO y la carga de página) y el resto de las interacciones que se realicen serán desde el cliente. Además, podremos crear **aplicaciones isomórficas o universales**, es decir**,** crear componentes que puedan ser reutilizados tanto en el lado del cliente como en el lado del servidor.
- **Ionic**: librería que nos permite crear aplicaciones móviles para Android e iOS programando únicamente en JavaScript. El usuario debe elegir un *framework* base entre Angular, React o Vue.js, para poder usar esta librería. También permite el uso de componentes Ionic sin ningún marco de interfaz de usuario.

Conclusión: elegir un *framework* o librería para desarrollar nuestras aplicaciones, va a depender directamente del tipo de aplicación que estemos buscando desarrollar.

En esta unidad vamos a trabajar con jQuery debido a la gran popularidad que ya tiene desde hace años y la cantidad de aplicaciones que la utilizan actualmente, pero React, Vue.js o Angular están apuntando fuerte para sustituirla en un futuro.

2. jQuery. Introducción

jQuery es una librería de JavaScript que se basa en el lema *escribir menos, hacer más*. Reduce mucho el código JavaScript y simplifica el trabajo con DOM y AJAX.

Añadir jQuery a una página web

Tenemos dos opciones:

1. Descargar la librería desde jquery.com y vincularla a nuestro sitio web. Se pueden descargar en diferentes versiones:
    - Comprimida o de producción (.min.js)
    - No comprimida o de desarrollo (.js)
    - Versión ligera de las dos anteriores, es decir, sin los módulos de AJAX y ni efectos.

    Ejemplo: suponiendo que nuestra descarga se encuentra en el directorio js

    ```
    <script src="js/jquery-3.5.1.min.js"></script>
    ```

2. Vincular la librería desde un **CDN** (*Content Distribution Network*), como Google o Microsoft. La ventaja de utilizar un CDN es que jQuery se puede descargar desde servidores distribuidos por todo el mundo. Además, si el visitante de la página web se ha descargado ya una copia de jQuery desde el mismo CDN, no necesita volver a descargárselo, lo tendrá almacenado en caché, por lo tanto, esta opción, es la recomendada por la comunidad.

    Ejemplos: dos opciones de vinculación desde los CDN de Google y Microsoft

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.5.1.min.js">
</script>
```

3. jQuery. Sintaxis

3.1. Objetos y función jQuery

La librería jQuery representa los objetos DOM como **objetos jQuery**, que permiten procesar el árbol DOM de forma más eficaz, ya que los arrays de objetos jQuery se procesan con un solo método, sin necesidad de bucles.

La **función jQuery**: `jQuery("selectorCSS")` o `$("selectorCSS")` devuelve la colección de objetos jQuery que casan con el selector CSS, si no casa ninguno, devuelve un objeto jQuery vacío. El argumento utiliza la sintaxis CSS para seleccionar objetos DOM.

Ejemplo:

```
document.getElementById("fecha")
//o
document.querySelectorAll("#fecha")
//es equivalente a:
$("#fecha")
```

3.2. Sintaxis básica

La sintaxis básica para trabajar con jQuery, es la siguiente:

```
$("selectorCSS").accion();
```

Donde: `acción()` será un método que indique qué acción queremos que se ejecute sobre el/los elemento/s seleccionados.

Ejemplos:

```
$(this).hide(); //oculta el elemento actual
$("p").hide();  //oculta todos los elementos <p>
$(".test").hide();    //oculta todos los elementos class="test"
$("#test").hide():    //oculta el elemento id="test"
```

3.3. Método ready(): árbol DOM construido

Ejecuta el código de la función que le pasamos como argumento, cuando el árbol DOM está construido, es decir, dicho bloque de código se ejecuta cuando ocurre el evento `DOMContentLoaded` de `body`. Se recomienda ejecutar el código jQuery cuando ha ocurrido este evento, por lo tanto, vamos a utilizar muchísimo esta función. Tenemos tres opciones:

1. Con función anónima:

```
$(document).ready(function() {
    //código jQuery o JavaScript
});
```

2. Con función con nombre:

```
$(document).ready(miFuncion);
function miFuncion(){
    //código jQuery o JavaScript
});
```

3. Con su forma abreviada. Es la que se recomienda utilizar, pero cuando estamos empezando con jQuery, puede dar lugar a confusión, por lo que, en nuestros ejemplos, no vamos a utilizar (de momento) la forma abreviada:

```
$(function() {
    //código jQuery o JavaScript
});
```

4. jQuery. Selectores

jQuery nos permite trabajar con los mismos selectores que CSS3, y además proporciona otros selectores propios. En el siguiente enlace https://www.w3schools.com/jquery/trysel.asp se pueden probar los selectores con un ejemplo muy visual.

4.1. Tipos de selectores

Selectores básicos

Selector	Selecciona
`*`	Selector universal: todos los elementos.
`#id`	Selector de identificador: el elemento con ese identificador.
`.clase`	Selector de clase: todos los elementos con esa clase.
`elemento`	Selector de elemento: todos los elementos que tienen esa etiqueta.
`selector1, selector2...`	Selector múltiple: los elementos seleccionados por el conjunto de selectores.

Ejemplos:

```
$("h1#d83") //devuelve el elemento con h1 e id="d83"
$("#d83") //devuelve el elemento con id="d83"
$("h1.princ")//devuelve array de elementos con h1 y
class="princ"
$(".princ") //devuelve array de elementos con class="princ"
$("h1, h2, p") //devuelve array de elementos h1, h2 y p
$("div").length; //devuelve el número de elementos div
```

Selectores de atributos

Selector	Selecciona
`[atributo]`	Elementos que tienen el atributo especificado, con cualquier valor.
`[atributo="valor"]`	Elementos que tienen ese atributo con exactamente ese valor.
`[atributo!="valor"]`	Elementos que tienen ese atributo con otro valor, o que no tienen el atributo. (Específico de jQuery)
`[atributo^="valor"]`	Elementos que tienen este atributo, y su valor empieza por "valor".
`[atributo|="prefijo"]`	Elementos que tienen ese atributo, y su valor empieza con ese prefijo (separado del resto por un guion (-)).
`[atributo$="valor"]`	Elementos que tienen este atributo, y su valor termina con "valor".
`[atributo*="subcadena"]`	El atributo contiene esa subcadena.
`[atributo~="campo"]`	El atributo puede estar formado por una lista de valores separados por espacios, y uno de esos valores es el especificado.
`[nombre="valor"][nombre`	Elementos que tienen esos atributos con esos valores.

2="valor2"]	

Ejemplos:

```
$("h1[border]") //devuelve array de elementos h1 con atributo border
$("h1[border=yes]") //devuelve array de elementos h1 con atributo border=yes
```

Selectores de jerarquía

Selector	Selecciona
a > b	Selector de hijo: todos los b que son hijos directos de un a.
a b	Selector de descendiente: todos los b que son descendientes de a.
a + b	Selector de siguiente elemento adyacente: todos los b que están a continuación de un elemento a.
a ~ b	Selector de hermanos (a continuación): todos los b que tienen como hermano anterior un a (no necesariamente inmediato).

Ejemplos:

```
$("h1 h2") //devuelve array de elementos con h2 después de h1
$("h1 > h2") //devuelve array de elementos con marca h2 justo después de h1
$("h1 + p") //devuelve array de elementos con marca p adyacente a h1 del mismo nivel
```

Filtros de hijos

Selector	Selecciona
:first-child	Todos los elementos que son el primer hijo de su padre.
:first-of-type	Todos los elementos que son el primero entre los hermanos del mismo tipo de elemento.
:last-child	Todos los elementos que son el último hijo de su padre.
:last-of-type	Todos los elementos que son el último entre los hermanos del mismo tipo de elemento.
:nth-child(n)	Todos los elementos que son el hijo n de su padre.
:nth-last-child(n)	Todos los elementos que son el hijo n de su padre, contando desde atrás.
:nth-of-type(n)	Todos los elementos que son el hijo n de su padre, en relación con los hermanos del mismo tipo de elemento.
:nth-last-of-type(n)	Todos los elementos que son el hijo n de su padre, en relación con los hermanos del mismo tipo de elemento, contando desde el final.
:only-child	Todos los elementos que son hijo único de su padre.
:only-of-type	Todos los elementos que no tienen hermanos del mismo tipo.

Filtros básicos

Selector	Selecciona

:animated	Elementos que están siendo animados. (Específico de jQuery)
:focus	El elemento que tiene el foco.
:header	Elementos cabecera <h1>, <h2>, ... (Específico de jQuery)
:lang(idioma)	Elementos en el idioma especificado.
:not(selector)	Los elementos que no sean seleccionados por el selector dado.
:root	El elemento que es la raíz del documento.
:target	El elemento destino identificado por el identificador en la URI del documento.

Los siguientes selectores de filtro están obsoletos: `:eq()`, `:even()`, `:first()`, `:gt()`, `:last()`, `:lt()`, `:odd()`.

Filtros por contenido

Selector	Selecciona
:contains(text)	Los elementos que contienen el texto.
:empty	Todos los elementos que están vacíos.
:has(selector)	Todos los elementos que tienen un elemento `selector`.
:parent	Los elementos que son padres de algún elemento.

Filtros de elementos de formularios

Selector	Selecciona
:input	Elementos `input`, `textarea`, `select` y `button`. (Específico de jQuery)
:text	Elementos input de tipo `text`. (Específico de jQuery)
:password	Elementos input de tipo `password`. (Específico de jQuery)
:radio	Elementos input de tipo `radio`. (Específico de jQuery)
:checkbox	Elementos input de tipo `checkbox`. (Específico de jQuery)
:submit	Elementos input de tipo `submit`. (Específico de jQuery)
:reset	Elementos input de tipo `reset`. (Específico de jQuery)
:button	Elementos `button` y los elementos `input` con `type=button`. (Específico de jQuery)
:image	Elementos input de tipo `image`. (Específico de jQuery)
:file	Elementos input de tipo `file`. (Específico de jQuery)
:enabled	Elementos que estén `enabled`.
:disabled	Elementos que estén `disabled`.
:selected	Elementos que están seleccionados. (Específico de jQuery)
:checked	Elementos que estén `checked` o `selected`
:focus	Elemento que tiene el foco. (Específico de jQuery)

Filtros por visibilidad (específicos de jQuery)

Selector	Selecciona
:hidden	Elementos que estén ocultos. (Específico de jQuery)
:visible	Elementos que son visibles. (Específico de jQuery)

4.2. Filtrado de selecciones

Métodos que reducen el conjunto de elementos seleccionados.

Selector	Selecciona
.eq(*index*)	Reduce el conjunto de elementos seleccionados al que esté en la posición indicada por `index` (empezando desde 0). Si el índice es un valor negativo, se empieza a contar desde atrás.
.filter(*selector*)	Reduce el conjunto de elementos seleccionados, a los que sean seleccionados por `selector`.
.first()	Reduce el conjunto de elementos seleccionados al primer elemento.
.has(*selector*)	Reduce el conjunto de elementos seleccionados, a los que tienen un descendiente que es seleccionado por `selector`.
.is(*selector*)	Devuelve `true` si al menos uno de los elementos seleccionados, es seleccionado por este `selector`.
.last()	Reduce el conjunto de elementos seleccionados al último elemento.
.map(*función*)	Pasa cada elemento del conjunto de elementos seleccionados a través de una función, produciendo un nuevo objeto jQuery que contiene los valores devueltos.
.not(*argumento*)	Elimina elementos del conjunto de elementos seleccionados. El argumento puede ser un selector, un elemento o un array.
.slice(*inicio* [,*fin*])	Reduce el conjunto de elementos seleccionados a un conjunto especificado por un rango de índices.

Ejemplos:

```
$('div.foo').has('p') //elementos div.foo que contienen <p>
$('h1').not('.bar') //elementos h1 que no poseen la clase 'bar'
$('ul li').filter('.current') //ítems de una lista desordenada que
 poseen la clase 'current'
$('ul li').first() // el primer item de una lista desordenada
$('ul li').eq(5) //el sexto item de una lista desordenada
```

4.3. Encadenamiento de selecciones

Si en una selección se realiza una llamada a un método, y éste devuelve un objeto jQuery, es posible seguir un “encadenado” de métodos en el objeto.

Ejemplo:

```
$('#content').find('h3').eq(2).html('nuevo texto para el tercer
elemento h3');
```

Este código se puede escribir utilizando saltos de línea, haciendo que sea más claro:

```
$('#content')
    .find('h3')
    .eq(2)
    .html('nuevo texto para el tercer elemento h3');
```

Si deseamos volver al conjunto de elementos proporcionado por una selección anterior, tenemos el método `end()`.

Método	**Devuelve**
`.end()`	Finaliza la operación de filtrado más reciente y devuelve el conjunto de elementos coincidentes a su estado anterior.

```
$('#content')
    .find('h3')
    .eq(2)
    .html('nuevo texto para el tercer elemento h3')
    .end() //Reestablece la selección a todos los h3 en #content
    .eq(0)
    .html('nuevo texto para el primer elemento h3');
```

5. jQuery. Eventos

La mayoría de los eventos del DOM tienen su método de evento (*Event Method*) equivalente de jQuery.

Ejemplo: el argumento será el manejador del evento (función anónima o con nombre)

```
$("p").click(function() {
    //Acciones a realizar cuando se produzca el evento click
});
```

Eventos del ratón

Evento	Descripción
`.click()`	Al hacer clic sobre un elemento.
`.dblclick()`	Al hacer doble clic sobre un elemento.
`.mousedown()`	Al presionar el botón izquierdo del ratón sobre un elemento.
`.mouseup()`	Al soltar el botón izquierdo del ratón que está presionado sobre un elemento.
`.mouseenter()`	Cuando el puntero del ratón se sitúa sobre un elemento.
`.mouseleave()`	Cuando el puntero del ratón abandona un elemento sobre el que estaba.
`.hover()`	Combinación de `mouseenter()` y `mouseleave()`.

Eventos de teclado

Evento	Descripción
`.keypress()`	Al presionar una tecla.
`.keydown()`	Mientras presionamos una tecla.
`.keyup()`	Al soltar una tecla.

Eventos de formulario

Evento	Descripción
`.submit()`	Al enviar el formulario.
`.change()`	Al cambiar el valor de un campo del formulario.
`.focus()`	Cuando un control de un formulario obtiene el foco.
`.blur()`	Cuando un control de un formulario pierde el foco.

Ejemplo: al hacer clic sobre los enlaces, aparece el mensaje de alerta y continua con su comportamiento por defecto, navegando la URL indicada en el atributo `href` de cada enlace:

```
$(document).ready(function() {
     $("a").click(function(e) {
         alert("¡Gracias por la visita!");
     });
});
```

Se puede impedir el comportamiento por defecto de muchos eventos, con el método `.preventDefault()`.

Ejemplo:

```
$(document).ready(function() {
     $("a").click(function(e) {
         alert("Este enlace no te lleva a ningún sitio." );
         event.preventDefault();
     });
});
```

Métodos avanzados para el trabajo con eventos

Método	Devuelve
.on(evento, [ojeto], function)	Asocia eventos a manejadores de eventos. Se utiliza igual que `.addEventListener()` de JavaScript salvo por el segundo parámetro opcional, que nos permite llamar al manejador con los parámetros indicados en un literal de objetos. También nos permite poner varios eventos asociados al mismo selector.
.one(evento, [ojeto], function)	Igual que `.on()` pero solamente ejecuta el evento UNA VEZ para cada elemento de la selección.
.off()	Elimina los manejadores de eventos asociados a los elementos seleccionados.
.trigger(evento)	Simula la ejecución de un evento.
.triggerHandelr(e)	Lanza el manejador del evento asociado a un evento, pero sin que se produzca este.

Ejemplos:

```
$("p").on("click", mensaje);
function mensaje() {
    alert("Párrafo pulsado");
}
//Llamada a una función con parámetros
$("p").on("click", {nombre: "Ada", apellido: "Lovelace"},
mensajeParametros);
function mensajeParametros(e) {
    alert(e.data.nombre + " " +
    e.data.apellido);
}
```

6. jQuery. Elementos HTML y CSS

6.1. jQuery. Métodos para manipular HTML

Obtener o modificar el contenido HTML o texto de un elemento

Método	Devuelve
`.html()`	Obtiene o modifica el `innerHTML` de los elementos seleccionados.
`.text()`	Obtiene o modifica el `textContent` de los elementos seleccionados.

Ejemplos:

```
$("p").html("Hola"); //Cambia el innerHTML de todos los <p>
let miHTML = $("p").html(); //Obtiene el innerHTML del primer <p>
$( "div" ).html( "<span class='red'>Hola <b>de nuevo</b></span>" );
// Establece contenido html dentro de un <div>
```

Obtener, modificar o eliminar atributos

Método	Devuelve
`.attr(attr [,valor])`	Obtiene o modifica el valor del atributo.
`.removeAttr(attr)`	Elimina el atributo.
`.val()`	Obtiene o modifica el valor del atributo `value`.

Ejemplos:

```
let miAtributo = $('a').attr('href'); //Valor del atributo 'href'
de la primer <a>
$('a').attr('href',  'pagina.html');  //Modifica  el  valor  del
atributo 'href' de todos los <a>
$('a').attr({
    'href':'pagina.html',
    'title':'nueva página'
});
//Modifica el valor del atributo 'href' y 'title' de todos los <a>
```

Funciones *callback*

Los métodos `text`, `html`, `val` y `attr` tienen una función *callback*. La función se ejecuta sobre cada uno de los elementos seleccionados. Tiene dos parámetros: índice, posición del elemento en el conjunto de elementos seleccionados y valor, valor del elemento.

Ejemplo: devuelve el valor que hay que asignar al `html` de ese elemento

```
$("div").html(function(indice, valor) {
    let valorAnt = valor;
    return valorAnt + "Y MAS";
});
```

6.2. jQuery. Métodos para manipular CSS

Métodos para obtener/establecer propiedades CSS de los elementos

Método	Devuelve
`.css(prop [,valor])`	Obtiene o modifica el valor de las propiedades CSS.
`.height([valor])`	Obtiene o establece la altura del o los elementos seleccionados.
`.width([valor])`	Obtiene o establece la altura del o los elementos seleccionados.

Ejemplos:

```
$('h1').css('font-size'); //Tamaño de la fuente del h1
$('h1').css('fontSize'); //Otra opción: formato camelCase
//Modificar propiedades css
$('h1').css('fontSize', '100px');
$('h1').css({ 'fontSize': '100px',
              'color': 'red'});
```

Métodos para manipular clases

Método	Devuelve
`.addClass(clase)`	Añade las clases a cada uno de elementos seleccionados.
`.hasClass(clase)`	Devuelve `true` si los elementos seleccionados tienen esas clases, false en caso contrario.
`.removeClass(clase)`	Elimina las clases de cada uno de los elementos seleccionados.
`.toggleClass(clase)`	Añade clases si no las tienen los elementos seleccionados o se eliminan si las tienen.

En lugar de aplicar estilos con la función `.css()`, es más recomendable crear reglas CSS que se apliquen a elementos de una clase, y con jQuery, asignar o eliminar esas clases.

Ejemplos:

```
let $h1 = $('h1');
$h1.addClass('big');
$h1.removeClass('big');
$h1.toogleClass('big');
if ($h1.hasClass('big')) { ... };
```

7. jQuery. DOM

7.1. Recorrer el DOM: *Traversing*

Métodos para recorrer el DOM: ascendientes

Método	Devuelve
`.parent([selector])`	Obtiene el padre de cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
`.parents([selector])`	Obtiene todos los antecesores de cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
`.parentsUntil([selector][,filtro])`	Obtiene todos los antecesores de cada elemento del conjunto de elementos seleccionados, hasta (pero no incluyendo) el elemento seleccionado por el selector, filtrados opcionalmente por un selector.
`.closest(selector)`	Obtiene el antecesor más cercano con el selector.

Los métodos `parent()` y `parents()` son similares, pero `.parent()` selecciona elementos entre los padres de los elementos seleccionados, y `.parents()` entre todos los ascendientes de los elementos seleccionados.

Ejemplo:

```
<ul class="level-1">
    <li class="item-i">I</li>
    <li class="item-ii">II
      <ul class="level-2">
        <li class="item-a">A</li>
        <li class="item-b">B</li>
        <li class="item-c">C</li>
      </ul>
    </li>
    <li class="item-iii">III</li>
</ul>
...
//Fondo rojo para el padre del item-a: <ul class="level-2">
$("li.item-a").parent().css("background-color", "red");
//Sería equivalente a:
$("li.item-a").closest("ul").css("background-color", "red");
//Fondo rojo para los antecesores del item-a: todo en fondo rojo
$("li.item-a").parents().css("background-color", "red");
```

Métodos para recorrer el DOM: descendientes

Método	Devuelve
.children([selector])	Obtiene los hijos directos de cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
.find(selector)	Obtiene los descendientes de cada elemento del conjunto de elementos seleccionados, filtrados por un selector, un objeto jQuery o un elemento.

Los dos métodos son similares, pero `.children()` selecciona elementos entre los hijos de los elementos seleccionados, y `find()` entre todos los descendientes de los elementos seleccionados.

Ejemplos: partiendo del HTML del ejemplo anterior

```
//Fondo rojo para los descencientes <li> del item-ii
$( "li.item-ii" ).find( "li" ).css( "background-color", "red" );
//Fondo rojo para los hijos del level-2 (mismo resultado en ambos)
$( "ul.level-2" ).children().css( "background-color", "red" );
```

Métodos para recorrer el DOM: hermanos

Método	Devuelve
.next([selector])	Obtiene el hermano inmediatamente posterior a cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
.nextAll([selector])	Obtiene todos los hermanos posteriores a cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
.nextUntil([selector] [,filter])	Obtiene todos los hermanos posteriores de cada elemento del conjunto de elementos seleccionados, hasta (pero no incluyendo) el elemento seleccionado por el selector, filtrados opcionalmente por un selector.
.prev([selector])	Obtiene el hermano inmediatamente anterior a cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
.prevAll([selector])	Obtiene todos los hermanos anteriores a cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.
.prevUntil([selector] [,filter])	Obtiene todos los hermanos anteriores de cada elemento del conjunto de elementos seleccionados, hasta (pero no incluyendo) el elemento seleccionado por el selector, filtrados opcionalmente por un selector.
.siblings([selector])	Obtiene todos los hermanos de cada elemento del conjunto de elementos seleccionados, filtrados opcionalmente por un selector.

Ejemplos:

```
<ul>
    <li class="primer-item">list item 1</li>
    <li>list item 2</li>
    <li class="third-item">list item 3</li>
    <li>list item 4</li>
    <li>list item 5</li>
</ul>
...
//Fondo rojo para el item 4
$( "li.third-item" ).next().css( "background-color", "red" );
//Fondo rojo para los items 4 y 5
$( "li.third-item" ).nextAll().css( "background-color", "red" );
//Fondo rojo para el item 2
$( "li.third-item" ).prev().css( "background-color", "red" );
//Fondo rojo para los items 1 y 2
$( "li.third-item" ).prevAll().css( "background-color", "red" );
//Fondo rojo para los items 1, 2, 3 y 4
$( "li.third-item" ).siblings().css( "background-color", "red" );
```

Otros métodos

Método	Devuelve
.add(selector)	Añade elementos al conjunto de elementos seleccionados. Permite añadir elementos a una selección jQuery, NO crea elementos.
.each(function)	Recorre un objeto jQuery, ejecutando una función para cada uno de los elementos con los que trabaja.
.get([indice])	Devuelve el elemento HTML que está en la posición indicada. Si el índice está fuera de los límites del array (número negativo o superior a la longitud del array), devuelve undefined. Si no se pasa índice, devuelve un array con todos los elementos del objeto jQuery.
.index([selector]|[element])	Devuelve la posición de un elemento de la selección jQuery entre sus hermanos en el DOM. Si no se encuentra el elemento, devuelve -1. Si no se pasa selector, devuelve la posición del primer elemento.

Ejemplos: partiendo del HTML del ejemplo anterior

```
//Fondo rojo para el item 3 y el item 1
$("li.third-item").add("li.primer-item").css("background-color",
"red" );
//Muestra por consola el índice y el texto de cada elemento <li>
$('li').each(function (idx, el) {
    console.log( 'El elemento ' + idx + ': '      + $(el).text() );
});
// Devuelve el elemento li que está en la posición 2
$('li').get(2);
```

```
// Devuelve la posición del <li> que tiene la clase 'current'.
$('li').index('current');
//Sería equivalente a:
$('li.current').index();
```

7.2. Manipulación del DOM

Crear elementos

Para crear elementos html, se puede utilizar la siguiente sintaxis:

```
$(html [,atributos])
```

Ejemplo: crear elementos pasando sólo una cadena HTML o una etiqueta

```
$("<p id='test'>Hola</p>")
```

Al pasar una cadena como parámetro a `$()`, jQuery examina la cadena para ver si es HTML (por ejemplo, si empieza con `<etiqueta …>`. En otro caso, interpreta que la cadena es un selector, como hemos visto hasta ahora.

Pero si la cadena parece ser HTML, jQuery intenta crear los elementos del DOM que indica la cadena, utilizando la función de JavaScript `createElement()`. Después se creará un objeto jQuery que contenga esos elementos.

Se puede pasar como **segundo parámetro** un **objeto de atributos** y algunos métodos de jQuery: `val`, `css`, `html`, `text`, `data`, `width`, `height` y `offset`. En este caso, el primer argumento debería ser un elemento simple, sin atributos.

Ejemplo: crear elementos pasando un objeto de atributos

```
//Crea un párrafo con un atributo class, un texto y un manejador
//del evento clic
let nuevaP = $("<p/>", {
    "class": "test",
    text: "Mi párrafo",
    click: function() {
        console.log($(this).text());
    }
})
```

Insertar elementos como hermanos de elementos existentes

Método	Devuelve
`.after(content[, content])`	Inserta el contenido especificado por el parámetro, detrás de cada elemento del conjunto de elementos con el que se está trabajando. `content`: string html, elemento, texto, array o jQuery
`.before(content[,content])`	Inserta el contenido especificado por el parámetro, delante de cada elemento del conjunto de elementos con el que se

	está trabajando. `content`: string html, elemento, texto, array o jQuery
`.insertAfter(target)`	Inserta todos los elementos del conjunto de elementos con el que se está trabajando, detrás del target. `target`: selector, string html, elemento, array o jQuery
`.insertBefore(target)`	Inserta todos los elementos del conjunto de elementos con el que se está trabajando, delante del target. `target`: selector, string html, elemento, array o jQuery

Ejemplos:

```
<div class="container">
    <h2>Saludos</h2>
    <div class="inner">Hola</div>
    <div class="inner">Adiós</div>
</div>
```

```
//Inserta los párrafos después de cada elemento inner
$("<p>Test</p>").insertAfter(".inner");
//Sería equivalente a:
$(".inner").after("<p>Test</p>");
//Inserta el título h2 después del elemento container
$("h2").insertAfter($(".container"));
//Sería equivalente a:
$(".container").after($("h2"));
```

```
//Inserta los párrafos antes de cada elemento inner
$("<p>Test</p>").insertBefore(".inner");
//Sería equivalente a:
$(".inner").before("<p>Test</p>);
//Inserta el título h2 antes del elemento container
$("h2").insertBefore($(".container"));
//Sería equivalente a:
$(".container").before($("h2"));
```

Insertar elementos como hijos de elementos existentes

Método	Devuelve
`.append(content[,content])`	Inserta el contenido (`content`) como último hijo de cada elemento del objeto jQuery. `content`: string html, elemento, texto, array o jQuery
`.appendTo(target)`	Inserta todos los elementos del conjunto de elementos con el que se está trabajando, como último hijo del target. `target`: selector, string html, elemento, array o jQuery
`.prepend(content[,content])`	Inserta el contenido (content) como primer hijo de cada elemento del objeto jQuery.

	`content`: string html, elemento, texto, array o jQuery
`.prependTo(target)`	Inserta todos los elementos del conjunto de elementos con el que se está trabajando, como primer hijo del target. `target`: selector, string html, elemento, array o jQuery

Ejemplos: partiendo del HTML del ejemplo anterior

```
//Inserta los párrafos como último hijo de cada elemento inner
$("<p>Test</p>").appendTo(".inner");
//Sería equivalente a:
$(".inner").append("<p>Test</p>");
//Inserta el título como último hijo del elemento container
$("h2").appendTo( $(".container"));
//Sería equivalente a:
$(".container").append($("h2"));
```

```
//Inserta los párrafos como primer hijo de cada elemento inner
$( "<p>Test</p>" ).prependTo(".inner" );
//Sería equivalente a:
$( ".inner" ).prepend( "<p>Test</p>" );
//Inserta el título como primer hijo del elemento container
$("h2").prependTo( $(".container") );
//Sería equivalente a:
$( ".container" ).prepend( $("h2") );
```

Clonar elementos

Método	Devuelve
`.clone([eventos])`	Duplica el conjunto de elementos del objeto jQuery, junto con su contenido (hijos). `eventos`: `true` si queremos que se copien también los manejadores de eventos, por defecto `false` (no se copian).

Ejemplo: partiendo del HTML del ejemplo anterior

```
//Duplica el div como último hijo del body
$("div.container").clone().appendTo($("body"));
```

Eliminar elementos del DOM

Método	Devuelve
`.empty ()`	Elimina todos los nodos hijos de los elementos seleccionados.
`.remove([selector])`	Elimina todos los elementos seleccionados, opcionalmente filtrados por el selector.

Ejemplo: partiendo del HTML del ejemplo anterior

```
//Elimina los elementos de clase inner
```

```
$(".inner").remove();
//Sería equivalente a
$("div").remove(".inner" );
//Elimina los hijos de container
$(".container").empty();
//Elimina el div container
$(".container").remove();
```

8. Métodos del núcleo del objeto jQuery

Métodos que se aplican directamente al núcleo de jQuery, no a una selección. Por ejemplo, el método `.each()` se puede llamar con `jQuery.each()` o `$.each()`.
Ojo: hay métodos, como `each()` que se pueden aplicar al núcleo de jQuery o a una selección jQuery, funcionando de forma un poco diferente en cada caso.

8.1. Utilidades para arrays

Método	Devuelve
`$.each(array|objeto, function)`	Función para realizar iteraciones sobre arrays u objetos. Devuelve el array original. • Con arrays: **`$.each(array, function(índice, valor) {})`** • Con objetos: **`$.each(objeto, function(prop, valor) {})`**
`$.map(array|objeto, function)`	Función para realizar iteraciones sobre arrays u objetos. Se utiliza igual que el método anterior, pero en este caso, devuelve un array nuevo basado en el original con los cambios.
`$.inArray(valor, array)`	Busca el valor en el array y devuelve su índice (o -1 si no lo encuentra).
`$.merge(array1, array2)`	Mezcla el contenido de dos arrays dentro del primer array.
`$.grep(array, function)`	Encuentra los elementos de un array (u objeto similar a un array) que satisfacen una función filtro. El array original no cambia. Devuelve un array formado por los elementos que satisfacen la función **`function(elemento, índice).`**
`$.makeArray(array_jQuery)`	Convierte un array jQuery en un array en un array JavaScript. Devuelve el array creado.

Ejemplo:

```
let array = [1, 3, 5, 7];
$.each(array, function (indice, valor) {
    alert(indice + " : " + valor);
});
```

8.2. Otros métodos

Métodos de tipos

Método	Devuelve
`$.isFunction(elem);`	Devuelven `true` si `elem` es del tipo indicado en la función, `false` en caso contrario.
`$.isArray(elem)`	
`$.isNumeric(elem)`	
`$.type(elem)`	Devuelve el tipo de datos de `elem`.

Método de fechas

Método	Devuelve
$.now()	Devuelve el número de milisegundos desde 1/1/1970.

Método de cadenas

Método	Devuelve
$.trim()	Elimina los espacios en blanco del principio y final de una cadena.

Métodos para añadir/eliminar datos

Estos métodos son muy útiles cuando necesitamos añadir información sobre los elementos DOM, pero no queremos modificar el DOM. Es mucho más eficiente.

Método	Devuelve
$.data(nombre, valor)	Añade un dato denominado `nombre` con el `valor` indicado. Este método añade información a un elemento, pero no es información DOM, esta información puede ser una cadena o un objeto.
$.removeData(nombre)	Eliminamos el dato denominado `nombre`.

Ejemplo:

```
$("#nombre").click(function () {
    $("div").data("nombre", "Ada Lovelace");
});
$("#nombre2").click(function () {
    alert($("div").data("nombre"));
});
$("#nombre3").click(function () {
    $("div").removeData("nombre");
});
```

9. jQuery. Efectos y animaciones

Si algo ha hecho popular a jQuery es la facilidad con la que nos permite aplicar efectos y animaciones sobre nos elementos del DOM para hacer mucho más atractivas nuestras páginas web.

Sintaxis de efectos y animaciones:

```
$(selector).efecto(velocidad)
```

El argumento **`velocidad`** puede tomar tres valore posibles: `slow`, `fast` o el `número de milisegundos`.

jQuery posee un objeto **`jQuery.fx.speeds`** que contiene la velocidad predeterminada para la duración de un efecto, así como también los valores para las definiciones `slow` y `fast`. Por lo tanto, es posible sobrescribir o añadir nuevos valores al objeto.

```
speeds: {
    slow: 600,
    fast: 200,
    //Velocidad predeterminada
    _default: 400
}
```

Ejemplo: añadir dos velocidades personalizadas

```
jQuery.fx.speeds.muyRapido = 100;
jQuery.fx.speeds.muyLento = 2000;
```

9.1. Efectos sobre la visibilidad: *hide* y *show*

Estos tres métodos las propiedades **`heigth`**, **`width`** y **`opacity`** de los elementos seleccionados. Su duración por defecto es 0. Además cambian el valor de la propiedad `display`.

Tienen dos argumentos opcionales: la **velocidad** y una **función** callback que se ejecutará una vez que se haya aplicado el efecto.

Método	Devuelve
`.hide([vel], [function])`	Oculta el elemento sobre el que se aplica el método. Anima las propiedades `heigth`, `width` y `opacity`. Cuando alcanzan el valor 0, establece la propiedad `display` a `none`.
`.show([vel], [function])`	Muestra el elemento sobre el que se aplica el método. Restaura el valor de la propiedad `display`. Anima las propiedades `heigth`, `width` y `opacity` (hasta su valor inicial).
`.toggle([vel], [function])`	Muestra elementos que están ocultos, y oculta los elementos que se están mostrando.

Ejemplos:

```
//Oculta los párrafos tras hacer clic en un botón
$("button").click(function() {
    $("p").hide(1000);
});
//Al hacer clic en los enlaces, se ocultan
$("a").click(function(e) {
     event.preventDefault();
     $(this).hide("slow");
});
```

9.2. Efectos sobre la opacidad: *fade*

Estos efectos animan la **opacidad** (**`opacity`**) de los elementos seleccionados. La duración por defecto es 400. Además cambian el valor de la propiedad **`display`**.

Tienen dos argumentos opcionales: la **velocidad** y una **función** callback que se ejecutará una vez que se haya aplicado el efecto. El método `.fadeTo()` tiene un argumento más, la opacidad, con un valor entre 0 y 1. En este caso tanto la velocidad como la opacidad, son obligatorios.

Método	Devuelve
`.fadeIn([vel], [function])`	Muestra el elemento (display deja de ser none). Después, anima la propiedad `opacity` (hasta el valor 1). Es similar a `.fadeTo()`, pero ese método no cambia la propiedad `display` y puede especificar el nivel de opacidad.
`.fadeOut([vel], [function])`	Anima la propiedad `opacity`, hasta que alcance el valor 0. Después, establece la propiedad `display` a `none`.
`.fadeToggle([vel], [function])`	Hace lo mismo que `.fadeIn()`, `.fadeOut()`, dependiendo de la situación inicial de la imagen.
`.fadeTo(vel, opacidad,[function])`	Cambia el valor de la propiedad `opacity` hasta el valor indicado. No cambia el valor de `display`. Es similar a `.fadeIn()`, pero ese método cambia la propiedad `display` y le aplica una opacidad del 100%.

Ejemplos:

```
$("button").click(function(){
    $("#div1").fadeOut();
    $("#div2").fadeOut("slow");
    $("#div3").fadeOut(3000);
});
```

```
$("button").click(function(){
    $("#div1").fadeTo("slow", 0.15);
```

```
    $("#div2").fadeTo("slow", 0.4);
    $("#div3").fadeTo("slow", 0.7);
});
```

9.3. Efectos sobre la altura: *slide*

Estos efectos animan la **altura** (`height`) de los elementos seleccionados. También cambian el valor de la propiedad `display`. La duración por defecto es 400.

Tienen dos argumentos opcionales: la **velocidad** y una **función** callback que se ejecutará una vez que se haya aplicado el efecto.

Método	Devuelve
`.slideDown([vel], [function])`	Muestra el elemento (`display` deja de ser `none`). Después, anima la propiedad `heigth` (hasta el valor del elemento).
`.slideUp([vel], [function])`	Después, anima la propiedad heigth (hasta el valor 0 o min-heigth). Después, oculta el elemento (display deja de ser none).
`.slideToggle([vel], [function])`	Si el elemento está oculto, aplica slideDown(); y si se ve, aplica slideUp().

Ejemplos:

```
$("#flip").click(function(){
    $("#panel").slideDown("slow");
});
```

```
$("#flip").click(function(){
    $("#panel").slideToggle("slow");
});
```

9.4. Efectos personalizados: animate

Método	Devuelve
`.animate({params},[vel],function])`	Crea animaciones personalizadas sobre distintas propiedades CSS. Tiene un primer argumento obligatorio **`params`**, que son los pares propiedad-valor de CSS que se quieren cambiar. Los otros dos argumentos son opcionales e idénticos a los de los métodos anteriores.
`.delay(ms)`	Crea un retardo de los milisegundos indicados.
`.stop([stopAll], [goToEnd])`	Para una animación. Tiene dos parámetros opcionales: • **`stopAll:`** `true` si queremos borrar también la cola de animación o no. El valor predeterminado es `false`, lo que significa que solo se detendrá la animación activa. • **`goToEnd:`** `true` si se completa o no la animación actual

	inmediatamente, el valor predeterminado es `false`.

Ejemplo: para que las propiedades que indican posición, como `left`, funcionen, el posicionamiento (`position`) del div debe ser `relative`, `absolute` o `fixed`.

```
$("div").animate({
    left: '250px',
    opacity: '0.5',
    height: '150px',
    width: '150px'
});
```

Qué propiedades se pueden animar
Se pueden animar casi todas las propiedades que tengan un valor numérico. Por ejemplo `width` y `left` se pueden animar, pero `background-color`, no.

Las propiedades CSS *shorthand* (ej. `font`, `border`), no están completamente soportadas. Por eso, es preferible utilizar `fontSize` o `font-size` en lugar de `font`.

Valores que pueden tomar las propiedades
Los valores de las propiedades se tratan como un número de píxeles a menos que se especifique otra unidad. Las unidades % y `em` se pueden especificar donde son aplicables.

Además de los valores numéricos, cada propiedad puede tomar los valores `show`, `hide` y `toggle`.

Asignar valores relativos mediante += y -=
Si los valores de las propiedades empiezan por `+=` o `-=`, esa cantidad de sumará o restará al valor que tenga la propiedad.

Ejemplo:

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

Asignar a las propiedades valores predefinidos *show*, *hide* o *toggle*
Algunas propiedades pueden tomar como valor *show*, *hide* o *toggle*.

Ejemplo:

```
$("#book").animate({
    opacity: 0.25,
    left: "+=50",
```

```
    height: "toggle"
}, 5000);
```

Animaciones y funcionalidad de cola

Por defecto, jQuery crea una cola de efectos para cada elemento. Esto quiere decir que, hasta que no se termine un efecto sobre un elemento, no se ejecutará el siguiente.

10. jQuery. AJAX

Otra de las partes del trabajo con JavaScript que nos facilita mucho jQuery, es el trabajo con AJAX.

10.1. Método `$.ajax()`

Nos permite realizar peticiones AJAX al servidor.

Sintaxis:

```
$.ajax(argumento)
```

Su único argumento es un objeto con una serie de opciones de configuración para personalizar la solicitud AJAX (similar al objeto de opciones que le podíamos pasar al método `fetch()`). Algunas de esas opciones:

- `url`: URL de la petición.
- `data`: datos que se quieran añadir a la solicitud, en formato de cadena o de objeto.
- `success`: función que se ejecuta si la petición ha resultado satisfactoria. Es la función de respuesta.
- `error`: función que se ejecuta si la petición ha devuelto un error. Tiene dos parámetros opcionales `xhr` y `status`.
- `complete`: función que se ejecuta siempre, independientemente de que se haya ejecutado la de `success` o la de `error`. También tiene los dos parámetros opcionales `xhr` y `status`.

Ejemplo:

```
$.ajax({
    url: "saludo.php",
    data: datos,
    success: function (respuesta) {
        $("#mostrar").text(respuesta);
    },
    error: function (xhr, status) {
        alert("Ha ocurrido un error");
    },
    complete: function (xhr, status) {
        alert("Petición realizada");
    }
});
```

10.2. Otros métodos para realizar peticiones

Basados en el método `$.ajax()` pero ya tienen algunos valores del objeto argumentos, establecidos por defecto.

Método `$.get()`
Recibe los datos de un servidor utilizando una petición AJAX GET.

Sintaxis:

```
$.get(url, [argumentos], [función])
```

Los parámetros son:

- `url`: obligatorio, URL del archivo solicitado.
- `argumentos`: opcional, es un objeto con datos que se añaden a a la solicitud.
- `función`: opcional, función que se ejecuta si la petición ha resultado satisfactoria, es la función de respuesta.

Ejemplo:

```
$("#enviarGet").click(function () {
    $.get("saludo.php", {
        "nombre": "Ada",
        "apellido": "Lovelace"
    }, function (respuesta) {
        $("#mostrar").text(respuesta);
    });
});
```

Método `$.post()`

Recibe los datos de un servidor utilizando una petición AJAX POST.

Sintaxis:

```
$.post(url, [argumentos], [función])
```

Los parámetros son los mismos que para el método `$.get()`.

Métodos que se usan al final de una petición `$.get()` o `$.post()`

Método	Devuelve
`.done()`	Se ejecuta si la petición es satisfactoria, después de haya ejecutado la función de respuesta.
`.fail()`	Se ejecuta si se produce una error en la petición.
`.always()`	Se ejecuta siempre.

Ejemplo:

```
$.get("saludo.php", function () {
    alert("Exito");
}).done(function () {
    alert("Exito 2");
}).fail(function () {
    alert("Error");
}).always(function () {
    alert("Siempre");
});
```

Otros métodos para realizar peticiones AJAX

Método	Devuelve
$.getScript(url, [argumentos], [función])	Carga un JavaScript y ejecuta una función de ese script. Los parámetros son los mismos que para los métodos `$.get()` y `$.post()`.
$.getJSON(url, [argumentos], [función])	Obtiene un JSON desde el servidor. Los parámetros son los mismos que para los métodos `$.get()` y `$.post()`.
$.load(url, [argumentos], [función])	Carga datos del servidor e introduce el HTML en un elemento. Los parámetros son los mismos que para los métodos `$.get()` y `$.post()`.

Ejemplo:

```
$("#getScript").click(function () {
    $.getScript("script.js", function () {
        dentroScript();
    })
});
```