jQuery

1. Introducción

jQuery es una biblioteca de JavaScript diseñada para simplificar y mejorar la manipulación de elementos HTML, el manejo de eventos y la creación de animaciones en páginas web. Fue creada en 2006 y es ampliamente utilizada en aplicaciones y sitios web de todo tipo.

Una de las principales ventajas de jQuery es que proporciona una sintaxis simplificada y consistente para realizar tareas comunes en JavaScript, lo que permite a los desarrolladores escribir menos código y hacerlo de manera más clara y legible. Además, jQuery incluye una amplia variedad de métodos y funcionalidades que facilitan la manipulación de elementos HTML, el manejo de eventos y la creación de animaciones, lo que permite a los desarrolladores crear efectos interactivos y dinámicos sin tener que escribir código complejo.

Algunas de las cosas que se pueden hacer con jQuery son:

- Seleccionar elementos HTML y manipular sus atributos y estilos.
- Crear y manipular elementos HTML dinámicamente.
- Manejar eventos, como *clic*, *hover*, etc.
- Realizar peticiones HTTP asíncronas y cargar contenido dinámicamente.
- Crear animaciones y transiciones.

jQuery es ampliamente utilizado en aplicaciones y sitios web de todo tipo y es compatible con la mayoría de los navegadores modernos. Es una herramienta muy útil para cualquier desarrollador web que quiera agregar interacción y dinamismo a sus proyectos. No obstante, también tiene algunas desventajas:

- Es ahora menos necesario que cuando surgió debido a la modernización de los estándares.
- Su rendimiento no es óptimo.
- Su sintaxis puede producir código poco legible en proyectos grandes.
- Está perdiendo fuerza ante nuevos frameworks como React, Angular, Vue.
- Bootstrap (en su versión 5) y GitHub han dejado de usarlo.

La **documentación oficial** debe ser la referencia bibliográfica para aprender jQuery. Es muy rica en explicaciones y ejemplos, así que deberías consultarla siempre que necesites resolver una duda.

https://api.jquery.com/

2. Instalación

Tenemos dos opciones para utilizar jQuery: descargarnos el archivo de su página oficial o utilizar un CDN (red de distribución de contenidos).

Para **usar jQuery en local** tenemos que ir a la sección *download* de su web oficial: https://jquery.com/download/.

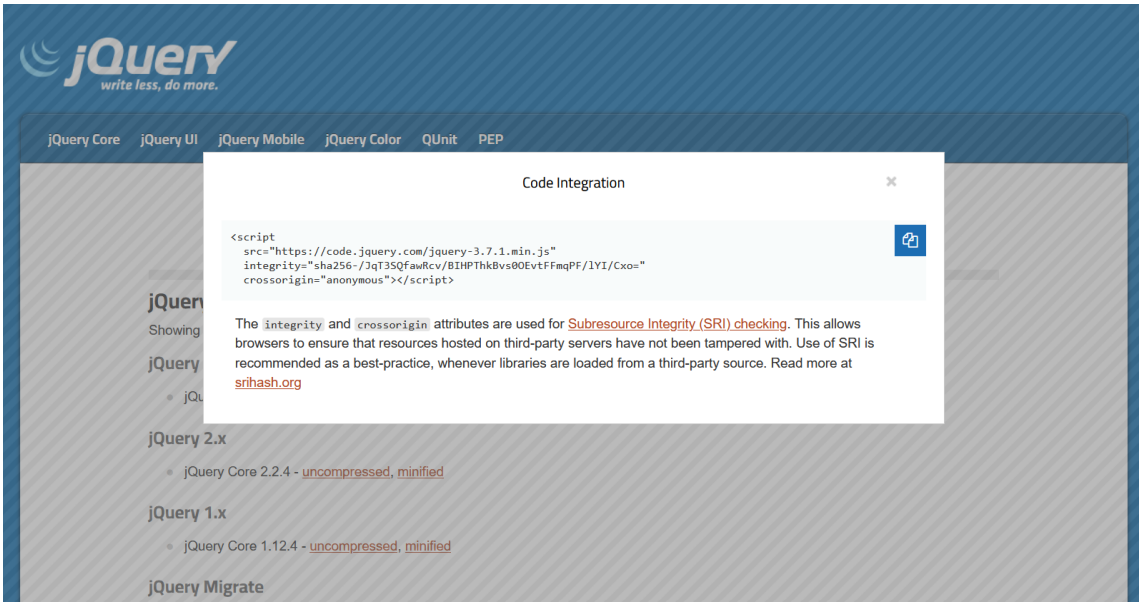
Nos dan a escoger entre descargar el archivo comprimido para producción, el no comprimido para desarrollo y el archivo *map*[1]. Dado que no vamos a revisar el código de jQuery, con el archivo comprimido (o *minificado*) nos vale.

Una vez hemos descargado ese fichero tendremos que añadirlo a nuestra página como cualquier otro archivo JavaScript:

```
<head>
  <script src="jquery-3.7.1.min.js"></script>
</head>
```

Otra opción es **usar jQuery en remoto**, sin tener que descargar nada. Para ello usaremos un CDN (*Content Delivery Network*). En la propia página de descarga de jQuery encontramos esta sección: https://jquery.com/download/#using-jquery-with-a-cdn

Si vamos a https://releases.jquery.com/, podemos escoger la versión que queramos. Por ejemplo, para la última versión minificada:



Copiamos ese link y lo insertamos en nuestra página como cualquier otro js:

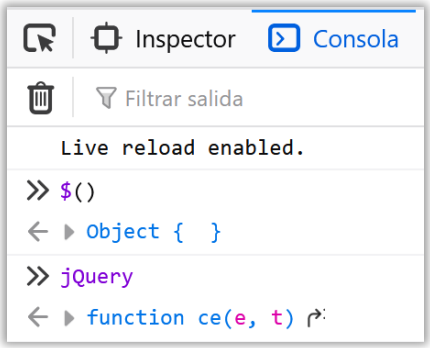
```
<script src="https://code.jquery.com/jquery-3.7.1.min.js" integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
```

[1] Los archivos .map se crean a partir de archivos de JavaScript minificados y se utilizan para proporcionar una correspondencia entre el código minificado y el código fuente original, lo que permite a los desarrolladores depurar y rastrear errores en el código fuente original en lugar de en el código minificado.

Otra opción es usar un CDN externo, como el de Google o el de Microsoft, cuyos enlaces tenemos en https://jquery.com/download/#other-cdns

La ventaja de cargar jQuery, o cualquier otro recurso, a partir de un CDN es que muchos usuarios ya lo habrán descargado al visitar otro sitio web. Como resultado, se cargará desde la caché cuando visiten tu web, lo que conduce a un tiempo de carga más rápido. Además, la mayoría de las CDN se asegurarán de que una vez que un usuario solicite un archivo, este se sirva desde el servidor más cercano, lo que también conduce a un tiempo de carga más rápido.

En cualquiera de los dos casos, podemos comprobar que todo está correcto desde la **consola** del navegador dentro de sus herramientas para desarrolladores. Dentro de esa consola prueba a ejecutar cualquiera de estas dos opciones: `$()` o `jQuery`:



Una vez hemos hecho esto ya podemos crear nuestros scripts usando jQuery. Es recomendable que estos scripts los añadamos al final del `body` y no al principio. Aunque los navegadores ya usan tecnología multihilo para cargar los distintos elementos de la página, puede ocurrir que esos scripts intenten manipular el DOM, pero aún no esté del todo creado, lo que daría lugar a errores.

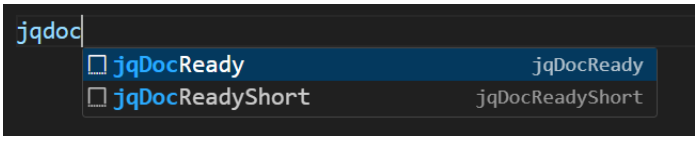
Para hacer nuestros primeros scripts con jQuery podremos optar por una de estas dos opciones, siendo ambas equivalentes:

```
$(document).ready(function () {
  // Codigo jQuery
});
```

O también:

```
$(function () {
  // Codigo jQuery
});
```

En Visual Studio Code, podemos usar alguna extensión de snippets de jQuery, como esta o esta, para escribir esto más rápidamente:



3. La función `jQuery()` o `$()`

La función `jQuery()` es la función principal de jQuery, dado que es la forma de seleccionar elementos del DOM para su posterior tratamiento. La forma `$()` es sólo un atajo, pero es la más utilizada por su sencillez. El uso de `jQuery()` suele quedar restringido a situaciones en las que se usan otras bibliotecas o frameworks JavaScript que también utilizan el carácter '$', como *prototype*, o *MooTools*.

Los tipos de **argumentos de entrada** que se pueden proporcionar a la función `jQuery()` incluyen:

- Selector CSS: Es el argumento más utilizado. Se emplea para seleccionar elementos del DOM mediante un selector CSS. Por ejemplo, para seleccionar todos los párrafos (`<p>`) de nuestra página utilizaríamos `jQuery("p")`. Tienes una relación con todos los selectores jQuery en esta sección de la documentación oficial de jQuery.

- HTML: Se utiliza para crear nuevos elementos HTML y agregarlos al DOM. Por ejemplo, si queremos crear un nuevo elemento `div` con el contenido “Hola Mundo”, podemos utilizar el siguiente código: `jQuery("<div>Hola Mundo</div>")`.

- Elemento DOM: Se utiliza para envolver un elemento DOM existente en un objeto jQuery. Por ejemplo, si queremos seleccionar un `div` con el `id="miDiv"` y envolverlo en un objeto jQuery podemos utilizar el siguiente código: `jQuery(document.getElementById("miDiv"))`, o más fácil, `$("#container")`.

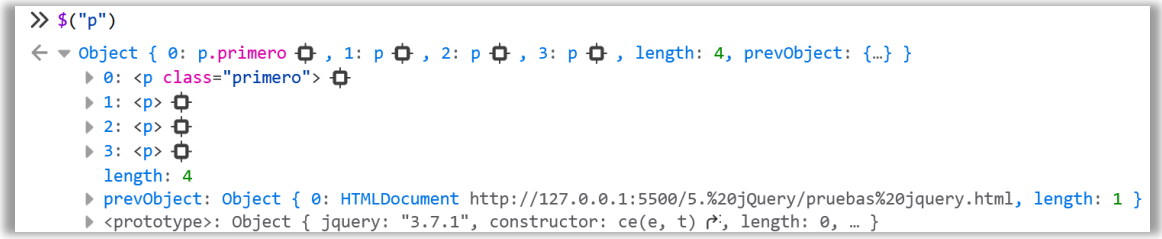
- Objeto jQuery: Se utiliza para crear una copia de un objeto jQuery existente. Por ejemplo, si queremos crear una copia de un objeto jQuery existente llamado “miObjeto”, podemos utilizar el siguiente código: `jQuery(miObjeto)`.

- Función: Se utiliza para ejecutar una función una vez que la página ha terminado de cargar. Por ejemplo, si queremos ejecutar una función llamada “miFuncion” una vez que la página ha terminado de cargar, podemos utilizar el siguiente código: `jQuery(miFuncion)`.

Vamos a ver algunos ejemplos usando este HTML:

```
<div id="container">
  <p class="primero">1</p>
  <p>2</p>
  <p>3</p>
  <p>4</p>
</div>
```

En la consola, si escribimos `$("p")` obtenemos todos los párrafos del documento:



Aunque ya hemos visto que `$()` devuelve un `object`, internamente es una colección de elementos del DOM que coinciden con el selector. Aunque parece un objeto, en realidad es un pseudo-array, ya que sus elementos están indexados de manera similar a un array. El atributo `length` de un objeto jQuery (que se observa en la imagen anterior) devuelve el número de elementos en esa colección.

Si escribimos `$("#container")` deberíamos obtener el único elemento coincidente:

```
>> $("#container")
← ▼ Object { 0: div#container , length: 1 }
    ▶ 0: <div id="container">
      length: 1
    ▶ <prototype>: Object { jquery: "3.7.1", constructor: ce(e, t), length: 0, … }
```

También podríamos haber escrito `$(document.getElementById("container"))` y obtendríamos lo mismo:

```
>> $("#container")
← ▼ Object { 0: div#container , length: 1 }
    ▶ 0: <div id="container">
      length: 1
    ▶ <prototype>: Object { jquery: "3.7.1", constructor: ce(e, t), length: 0, … }
>> $(document.getElementById("container"))
← ▼ Object { 0: div#container , length: 1 }
    ▶ 0: <div id="container">
      length: 1
    ▶ <prototype>: Object { jquery: "3.7.1", constructor: ce(e, t), length: 0, … }
```

Sin embargo, si escribimos `$("#contenedor")` jQuery no debería encontrar nada, por lo que devolvería un objeto vacío:

```
>> $("#contenedor")
← ▼ Object {  }
    ▶ <prototype>: Object { jquery: "3.7.1", constructor: ce(e, t), length: 0, … }
```

Esto es **importante**, cuando jQuery no encuentra coincidencia no devuelve `null`, devuelve un objeto vacío:

```
>> $("contenedor")==null
← false
```

4. Eventos, métodos y *event handlers*

Empecemos por las definiciones más básicas:

- Un **evento**, como ya sabrás de JavaScript, es una acción que sucede en una página web, como un clic del ratón, la pulsación de tecla, etc.

- En jQuery, un ***event handler*** (manejador de eventos) es una función que se ejecuta como respuesta a un evento específico. Se utilizan para manejar interacciones del usuario, como clics, teclas presionadas, etc. Se vinculan a elementos del DOM para responder a eventos y realizar acciones específicas cuando estos eventos suceden.

- Un **método** de jQuery es una función que realiza una determinada acción. Ciertos eventos fueron diseñados para responder a determinados eventos, como `click()`, y otros simplemente realizan acciones concretas, como `addClass()`.

Veamos un ejemplo de todo esto.

Para asignar un evento a un elemento del DOM simplemente tenemos que aplicar el **método** jQuery relacionado con ese evento. Digamos que queremos asignar el evento `click` a un botón con `id="mi-boton"`.

```
<input id="mi-boton" type="button" value="Enviar">
```

Usamos el identificador para seleccionar el elemento y le aplicamos el método relacionado con el evento, que en este caso es el método `click()`. Si revisamos la documentación de este método, vemos que recibe como parámetro un controlador de eventos (*handler*), que es una función que se ejecuta cuando un evento específico se dispara en el elemento. Por ejemplo:

```
$(document).ready(function () {
  $("#mi-boton").click(boton_click);
});
function boton_click() {
  alert("Has pulsado el botón");
}
```

Hemos definido la función y se la hemos pasado como parámetro. Sin embargo, en muchas ocasiones nos encontraremos con que vamos a utilizar esa función sólo una vez en el código. Si es así, podemos omitir la definición y utilizar una **función anónima**, que consiste en definir la función exactamente en el lugar donde va a ser llamada:

```
$(document).ready(function () {
  $("#mi-boton").click(function () {
    alert("Has pulsado el botón");
  });
});
```

Si atendemos a la documentación, el método `click()` aparece como obsoleto (*deprecated*) desde la versión 3.3 de jQuery, que nos insta a utilizar el método `on()`, que veremos a continuación. Esto ocurre con muchos otros métodos de jQuery, que fueron diseñados como métodos para manejar eventos y que han sido sustituidos por `on()`.

4.1. Los métodos `on()` y `one()`

El método `on()` se utiliza para establecer uno o varios *event handlers* para uno o varios elementos específicos del DOM. La sintaxis para utilizarlo es:

```
$(selector).on(event, function);
```

Donde `selector` es el selector jQuery para seleccionar el elemento o elementos para los que se establece el controlador de eventos, `event` es el evento o eventos para los que se establece el controlador de eventos (`'click'`, `'submit'`, etc.) y `function` es la función que se ejecutará cuando el evento ocurra.

Por ejemplo, para establecer un controlador de eventos para un botón con `id="mi-button"` para manejar el evento de clic podrías utilizar el siguiente código:

```
$("#mi-boton").click(function () { });
```

Pero jQuery aconseja usar desde su versión 1.7 el método `on()`:

```
$("#mi-boton").on("click", function () { });
```

El método `on()` es un **método genérico que permite asignar cualquier tipo de evento**, no solo `click`, y esto hace que sea más fácil manejar diferentes tipos de eventos en un mismo elemento o en varios elementos. También permite asignar múltiples controladores de eventos a un mismo elemento o a varios elementos.

Siguiendo el ejemplo del botón, podemos hacer que cuando se haga clic en él, se muestre un mensaje por consola:

```
$("#mi-boton").on("click", function () {
  console.log('Clic en el botón');
});
```

También podemos usar el método `on()` para asignar dos eventos:

```
$("#mi-boton").on("click mouseenter", function () {
  console.log('Clic en el botón');
});
```

Y también hacer que cada evento tenga su propia función:

```
$('#mi-boton').on({
  click: function () {
    console.log('Clic en el botón');
  },
  mouseenter: function () {
    console.log('Mouse entró en el botón');
  }
});
```

Existe otro método muy parecido, `one()`, pero que sólo se ejecuta una vez por cada elemento. Después de esta primera ejecución, el manejador se desvinculará automáticamente del evento.

4.2. Pasando datos al *event handler*

Tal y como indica la documentación de jQuery, a los métodos `.on()` y `.one()` les podemos pasar datos (`data`) como parámetro:

```
.on( events [, selector ] [, data ], handler )
```

Estos datos se pasan al manejador de eventos a través de la propiedad `event.data` cada vez que se dispara un evento. El argumento de datos puede ser de cualquier tipo, pero si se usa una cadena, se debe proporcionar un selector o pasarlo explícitamente como `null` para que los datos no se confundan con un selector. Lo más recomendable es usar un objeto simple para pasar valores como propiedades.

Por ejemplo:

```
$(document).ready(function () {
  $("#mi-boton").on("mouseenter", {
    nombre: "Juan"
  }, saludo);

  $("#mi-boton").on("mouseleave", {
    nombre: "María"
  }, saludo);
});

function saludo(event) {
  console.log("Hola " + event.data.nombre);
}
```

5. Tipos de eventos y métodos

La documentación de jQuery divide los eventos y los métodos en categorías (los métodos llevan paréntesis, los eventos no). Un resumen de ellos es este:

- Event Object: Sección sobre el objeto `event`, que veremos a continuación.
- Browser Events: `resize` y `scroll`.
- Document Loading: `load`, `unload` y `ready()`.
- Event Handler Attachment: `on()`, `one()`, `off()`, `trigger()` y `triggerHandler()`.
- Form Events: `submit`, `change`, `focus`…
- Keyboard Events: `keydown`, `keypress` (obsoleto)[2] y `keyup`.
- Mouse Events: `click`, `dblclick`, `mouseenter`, `mouseleave`…

5.1. El objeto `event`

La documentación explica muy detalladamente este objeto.

Para resumir, el objeto *Event* representa un evento en el navegador y se utiliza para proporcionar información sobre el tipo de evento, la posición del ratón, las teclas presionadas, entre otros detalles. Este objeto se pasa automáticamente como argumento a la función manejadora de eventos cuando ocurre un evento.

Por ejemplo:

```
<input id="mi-boton" type="button" value="Enviar">

<script>
  $(document).ready(function () {
    $('#mi-boton').on('click', function (event) {
      console.log('Tipo de evento:', event.type);
      console.log('Posición del mouse X:', event.pageX);
      console.log('Posición del mouse Y:', event.pageY);
    });
  });
</script>
```

Existe un método muy utilizado sobre objetos *event*, `preventDefault()`.

Este método se utiliza para evitar que se ejecute el comportamiento predeterminado de un evento, permitiéndonos manipular el comportamiento del evento de acuerdo a nuestras necesidades. Por ejemplo:

```
$("a#link").on("click", function (e) {
  e.preventDefault();
});
```

[2] El evento `keypress` ha sido declarado obsoleto y ninguna especificación oficial lo cubre. Aunque jQuery sigue dando soporte al evento, su uso está desaconsejado.

6. Eventos de teclado

Revisa la documentación oficial de jQuery sobre los eventos de teclado.

Los 2 eventos de teclado son bastante autodescriptivos:

- El evento `keydown`: Se activa cuando se pulsa una tecla y antes de que el valor de la tecla se registre en el elemento de entrada.
- El evento `keyup`: Se activa cuando se suelta una tecla y después de que el valor de la tecla se registre en el elemento de entrada.

Existe un tercer evento, `keypress`, que ha sido declarado obsoleto y ninguna especificación oficial lo cubre. Aunque jQuery sigue dando soporte al evento, su uso está desaconsejado.

Fíjate en el siguiente ejemplo:

```
<input type="text" id="textbox">

<script>
  $(document).ready(function () {
    $("#textbox").on("keydown", function () {
      console.log("keydown event");
    });
    $("#textbox").on("keyup", function () {
      console.log("keyup event");
    });
  });
</script>
```

Este código nos muestra por consola cuándo se dispara cada uno de los dos eventos. Es habitual querer conocer qué tecla se ha pulsado en un *input*, para lo que necesitamos hacer uso del objeto `event`, que ya conocemos:

```
$('#textbox').on('keyup', function (event) {
  console.log('Tecla pulsada:', event.key);
});
```

Y si queremos recuperar el texto escrito en el *input*, sería:

```
console.log('Valor del input:', $(this).val());
```

La propiedad `event.key` en JavaScript proporciona una cadena que representa la tecla presionada.

Para determinar si la tecla pulsada es imprimible podemos comprobar si `event.key.length` es igual a 1:

```
if (event.key.length === 1) {
  console.log("Sí es imprimible");
}
```

6.1. Evento `input`

Otro evento de teclado es el **evento input**. El evento `input` se dispara cuando el valor (*value*) de un elemento `<input>`, `<select>`, o `<textarea>` ha sido modificado. Así, podemos comprobar si el usuario ha modificado el contenido de uno de estos elementos de forma muy sencilla.

```
<input type="text" id="textbox">
<script>
  $(document).ready(function () {
    $("#textbox").on("input", function (event) {
      console.log(event.target.value);
      // O también:
      // console.log($(this).val());
    });
  });
</script>
```

Cuando usamos el evento `input`, el objeto *event* no transporta la última tecla pulsada, por lo que deberíamos usar `keyup` o `keydown` si quisiéramos conocerla.

Para elementos `<input>` con `type=checkbox` o `type=radio`, el evento `input` debería dispararse cuando el usuario alterna el control, según la especificación HTML5. Sin embargo, históricamente no siempre ha sido así. Es necesario revisar la compatibilidad o usar el evento `change` en su lugar para estos tipos.

6.2. Evento `change`

Dependiendo del tipo de elemento que cambia y la forma en que el usuario interactúa con el elemento, el evento `change` se dispara en un momento diferente:

- Cuando se modifica el estado `:checked` (ya sea dando `click` o usando el teclado) para elementos `<input type="radio">` y `<input type="checkbox">`.
- Cuando el usuario confirma un cambio explícitamente (por ejemplo, al seleccionar un valor de un `<select>`, al seleccionar una fecha en un `<input type="date">`, al seleccionar un archivo en un `<input type="file">`…
- Cuando un elemento pierde el foco después de que su valor haya cambiado (ej., después de editar el valor de un `<textarea>` o `<input type="text">`).

Evento	¿Qué hace?	Elementos típicos	Observaciones
input	Captura cambios en tiempo real mientras el usuario escribe o edita.	`<input>` (text, number…) `<textarea>`	Se activa cada vez que el valor cambia (tecleando, pegando texto, eliminando caracteres). No se activa en elementos como `<select>` o `<input type="checkbox">`.
change	Captura el valor final después de que el usuario confirme un cambio.	`<select>` *`checkboxes radiobuttons`*	Para `<select>`, se activa al cambiar la opción seleccionada. Para `<checkbox>` y `<radio>`, se activa al marcar/desmarcar. En campos de texto, solo se activa al perder el foco después de modificar el valor.

7. Métodos para manejar atributos

Revisa la documentación oficial de jQuery sobre el manejo de atributos.

Método	Explicación
`.addClass()`	Agrega la clase especificada a cada elemento del conjunto de elementos coincidentes.
`.attr()`	Obtiene el valor de un atributo del primer elemento del conjunto de elementos coincidentes o establece uno o varios atributos para cada elemento coincidente.
`.hasClass()`	Determina si alguno de los elementos coincidentes tiene asignada la clase dada.
`.html()`	Obtiene el contenido HTML del primer elemento del conjunto de elementos coincidentes o establece el contenido HTML de cada elemento coincidente.
`.prop()`	Obtiene el valor de una propiedad del primer elemento del conjunto de elementos coincidentes o establece una o varias propiedades para cada elemento coincidente.
`.removeAttr()`	Elimina un atributo de cada elemento del conjunto de elementos coincidentes.
`.removeClass()`	Elimina una clase, varias clases o todas las clases de cada elemento del conjunto de elementos coincidentes.
`.removeProp()`	Elimina una propiedad para el conjunto de elementos coincidentes.
`.toggleClass()`	Agrega o elimina una o varias clases de cada elemento del conjunto de elementos coincidentes, dependiendo de la presencia de la clase o del valor del argumento de estado.
`.val()`	Obtiene el valor actual del primer elemento del conjunto de elementos coincidentes o establece el valor de cada elemento coincidente.

Vamos a realizar algunos ejercicios para practicar. Partimos del siguiente HTML:

```
<style>
  .highlight {
    background-color: yellow;
  }

  .hidden {
    display: none;
  }
</style>

<h1 id="main-title">Título principal</h1>
<p id="description" class="hidden">Este es un párrafo con una clase
oculta.</p>
<button id="toggle-class">Cambiar clase</button>
<input id="input-field" type="text" value="Texto inicial">
<a id="link" href="https://example.com">Ir a Example</a>
<button id="submit-button" disabled>Enviar</button>
```

1. Usa `addClass()` para añadir la clase `highlight` al elemento `#main-title`.
2. Usa `attr()` para cambiar el atributo `href` del enlace `#link`.
3. Usa `hasClass()` para comprobar si el párrafo `#description` tiene la clase `hidden`. Muestra el resultado en la consola.

4. Usa `html()` para cambiar el contenido del título con ID main-title a *Título actualizado con jQuery*.
5. Usa `prop()` para habilitar el botón con ID `submit-button`.
6. Usa `removeAttr()` para eliminar el atributo `href` del enlace con ID link.
7. Usa `removeClass()` para quitar la clase `hidden` del párrafo `#description`.
8. Usa `removeProp()` para eliminar la propiedad `disabled` del botón `#submit-button`.
9. Usa `toggleClass()` para alternar la clase `highlight` en el botón #`toggle-class` cada vez que se haga clic en él.
10. Usa `val()` para cambiar el valor del campo de entrada con ID `#input-field`.

Las soluciones son:

```
// 1. addClass()
$("#main-title").addClass("highlight");

// 2. attr()
$("#link").attr("href", "https://google.com");

// 3. hasClass()
if ($("#description").hasClass("hidden")) {
  console.log("El párrafo tiene la clase 'hidden'.");
} else {
  console.log("El párrafo no tiene la clase 'hidden'.");
}

// 4. html()
$("#main-title").html("¡Título actualizado con jQuery!");

// 5. prop()
$("#submit-button").prop("disabled", false);

// 6. removeAttr()
$("#link").removeAttr("href");

// 7. removeClass()
$("#description").removeClass("hidden");

// 8. removeProp()
$("#submit-button")[0].removeAttribute("disabled");

// 9. toggleClass()
$("#toggle-class").click(function () {
  $(this).toggleClass("highlight");
});

// 10. val()
$("#input-field").val("Nuevo texto");
```

8. Métodos para manipular propiedades CSS

Revisa la documentación oficial de jQuery sobre manipular propiedades CSS.

Método	Descripción
`.css()`	Obtiene el valor de una propiedad para el primer elemento en el conjunto de elementos coincidentes o establece una o más propiedades CSS para cada elemento coincidente.
`.height()`	Obtiene la altura actual calculada para el primer elemento en el conjunto de elementos coincidentes o establece la altura de cada elemento coincidente.
`.innerHeight()`	Obtiene la altura interna actual calculada (incluyendo `padding`, pero no `border`) para el primer elemento en el conjunto de elementos coincidentes o establece la altura interna de cada elemento coincidente.
`.outerHeight()`	Obtiene la altura externa actual calculada (incluyendo `padding`, el `border` y `margin`) del primer elemento del conjunto de elementos coincidentes o establece la altura externa de cada elemento coincidente.
`.width()`	Obtiene el ancho calculado actual para el primer elemento en el conjunto de elementos coincidentes o establece el ancho de cada elemento coincidente.
`.innerWidth()`	Obtiene el ancho interno actual calculado (incluyendo el `padding`, pero no `border`) para el primer elemento en el conjunto de elementos coincidentes o establece el ancho interno de cada elemento coincidente.
`.outerWidth()`	Obtiene el ancho externo actual calculado (incluyendo el `padding`, el `border` y `margin`) del primer elemento del conjunto de elementos coincidentes o establece el ancho externo de cada elemento coincidente.
`.scrollTop()`	Obtiene la posición vertical actual de la barra de desplazamiento del primer elemento del conjunto de elementos coincidentes o establece la posición vertical de la barra de desplazamiento de cada elemento coincidente

9. Métodos para realizar efectos

Revisa la documentación oficial de jQuery sobre efectos.

Método	Descripción
`.animate()`	Realiza una animación personalizada de un conjunto de propiedades CSS.
`.fadeOut()`	Oculta los elementos coincidentes desvaneciéndolos a transparente.
`.fadeIn()`	Muestra los elementos coincidentes desvaneciéndolos a opaco.
`.fadeToggle()`	Muestra u oculta los elementos coincidentes animando su opacidad.
`.hide()`	Oculta los elementos coincidentes.
`.show()`	Muestra los elementos coincidentes.
`.toggle()`	Muestra u oculta los elementos coincidentes.
`.slideDown()`	Muestra los elementos coincidentes con un movimiento de deslizamiento.
`.slideUp()`	Oculta los elementos coincidentes con un movimiento de deslizamiento.
`.slideToggle()`	Muestra o oculta los elementos coincidentes con un movimiento de deslizamiento.

Vamos a ver sólo un ejemplo para explicar algún parámetro de entrada interesante, pero puedes ver ejemplos de funcionamiento de cada uno de los métodos en la documentación de jQuery.

El ejemplo lo veremos con la función `animate()` de jQuery se utiliza para crear animaciones. Aunque esto ya se puede hacer con la propiedad de CSS3 `transition`, la ventaja de usar jQuery es poder disparar el evento en función de nuestras necesidades.

Observa el siguiente ejemplo:

```
<button id="boton">Dale</button>
<div id="saludo" style="background-color: #bca; width: 100px">Hola Mundo</div>

<script>
  $(document).ready(function () {
    $("#boton").on("click", function () {
      $("#saludo").animate({
        width: "70%",
        opacity: 0.5,
        marginLeft: "1em",
        fontSize: "3em",
        borderWidth: "15px"
      }, 1500, "swing", function () {
        $(this).html("Animación acabada");
      });
    });
  });
</script>
```

Fíjate en el segundo, tercer y cuarto parámetro de la función, todos ellos opcionales: el valor 1500, el string `"swing"` y la función anónima:

- El primero es la duración de la animación, dada en milisegundos. También se aceptan los strings `"slow"` y `"fast"`, equivalentes a 200 y 600 milisegundos.
- El segundo es un *easing*, una cadena de texto que indica qué función de aceleración se usará para la transición. Las únicas implementaciones de aceleración en jQuery son la predeterminada, `swing`, y una que progresa a un ritmo constante, `lineal`. Hay más funciones disponibles con el uso de complementos, sobre todo la suite jQuery UI.
- El tercero es una función que se ejecutará cuando la animación haya terminado.

10. Métodos y eventos de formulario

Revisa la documentación oficial de jQuery sobre manejar formularios.

Método/Evento	Explicación
`blur`	Se dispara cuando un elemento pierde el foco. No se propaga.
`focusout`	Se dispara cuando un elemento o uno de sus descendientes **pierde el foco**.
`focus`	Se dispara cuando un elemento **recibe el foco**. No se propaga.
`focusin`	Se dispara cuando un elemento o uno de sus descendientes **recibe el foco**.
`change`	Se dispara cuando el valor de un elemento de formulario cambia. Por ejemplo, después de seleccionar una opción diferente en un menú desplegable.
`select`	Se dispara cuando se selecciona texto en un `<input>` o en un `<textarea>`.
`.serialize()`	Codifica un conjunto de elementos de formulario como una cadena para su envío.
`.serializeArray()`	Codifica un conjunto de elementos de formulario como una matriz de nombres y valores.
`submit`	Se dispara cuando se envía un formulario o activa ese evento en un formulario.
`.val()`	Obtiene el valor actual del primer elemento en el conjunto de elementos coincidentes o establece el valor de cada elemento coincidente.

10.1. *Event bubbling*

Hay dos pares de eventos que parecen hacer lo mismo: `blur` y `focusout` por un lado y `focus` y `focusin` por otro. Para explicar la diferencia tenemos que introducir el concepto de ***event bubbling***.

Event bubbling es un concepto en la propagación de eventos en el DOM en JavaScript. Cuando ocurre un evento en un elemento del DOM, este evento no afecta sólo a ese elemento, se propaga. Por ejemplo, si tienes un elemento `div` dentro de un elemento `form` y el `div` tiene un evento de `click` asociado a él, cuando se hace `click` en el `div`, se activará el evento asociado al `div` y también el evento asociado al `form`.

Se puede detener la propagación de un evento utilizando el método `stopPropagation()`. Por ejemplo:

```
$("p").on("click", function (event) {
  event.stopPropagation();
  // resto del código
});
```

Aquí hay un ejemplo de cómo utilizar estos eventos en jQuery:

```
<div id="padre" style="border: 1px solid black; padding: 5px">
  <input class="verde" type="text" value="Verde"><br>
  <input class="azul" type="text" value="Azul"><br>
</div>

<input class="verde" type="text" value="Verde">

<script>
  $(document).ready(function () {
    // Reciben el foco
    $(".verde").on("focus", function () {
      console.log("Evento focus en input.verde");
    });
    $("#padre").on("focusin", function () {
      console.log("Evento focusin en #padre");
    });

    // Pierden el foco
    $(".verde").on("blur", function () {
      console.log("Evento blur en input.verde");
    });
    $("#padre").on("focusout", function () {
      console.log("Evento focusout en #padre");
    });
  });
</script>
```

10.2. Comprobación de *checkboxes*

Para comprobar si un *checkbox* o un *radiobutton* está seleccionado tenemos un dos buenas alternativas:

10.2.1. Usando el método `is()`

El método `is()` permite comprobar si un elemento cumple con una determinada condición. La condición puede especificarse mediante una cadena de selectores, un objeto jQuery, un elemento DOM o una función de prueba.

La sintaxis básica del método es:

```
$(element).is(selector)
```

Donde *element* es el elemento o conjunto de elementos que queremos evaluar y *selector* es la condición que queremos comprobar.

Este método devuelve un valor booleano, es decir, *true* si el elemento o alguno de los elementos del conjunto cumplen con la condición y *false* en caso contrario. En nuestro caso, sería:

```
$('#cb').is(":checked") // Devuelve TRUE o FALSE
```

10.2.2. Usando el método `prop()`

El método *prop()* se puede usar para obtener o establecer el valor de una propiedad. En nuestro caso, sería:

```
$('#cb').prop("checked") // Devuelve TRUE o FALSE
```

Puedes ver un ejemplo de ambas formas a continuación:

```
<input type="checkbox" name="cb" id="cb">

<script>
  $(document).ready(function () {
    $('#cb').change(function () {
      console.log($('#cb').is(":checked"));
      console.log($('#cb').prop("checked"));
    });
  });
</script>
```

Un **error muy común** suele ser **utilizar** `attr()` para comprobar el estado de un *checkbox*. Un *checkbox* es un `input type="checkbox"`, que puede contener un atributo *inline* para determinar si aparecerá marcado o no al cargar la página.

Podemos acceder a los atributos usando jQuery con `attr()`. Pero el problema es que **el atributo** `checked` **no refleja el estado del *checkbox*, sino el valor predeterminado al cargar la página**.

En el siguiente ejemplo puedes probar esto. Al modificar cualquiera de los checkboxes en la consola siempre aparecerá la situación inicial.

```
<label for="cb1">cb1</label>
<input type="checkbox" name="cb1" id="cb1">

<label for="cb2">cb2</label>
<input type="checkbox" name="cb2" id="cb2">

<script>
  $(document).ready(function () {
    $("#cb1").on("change", function () {
      console.log("$('#cb1').attr('checked'): " + $('#cb1').attr('checked'));
    });

    $("#cb2").on("change", function () {
      console.log("$('#cb2').prop('checked'): " + $('#cb2').prop('checked'));
    });
  });
</script>
```

10.3. Comprobación de *radiobuttons*

Para comprobar cuál de los *radiobuttons* de un grupo es el que está seleccionado usamos el selector de atributo con el *name* que tenga el grupo y la función `is()`, que ya vimos en el apartado anterior.

Un ejemplo sencillo con solo 2 *radiobuttons* sería:

```
<label for="guapo">Guapo</label>
<input type="radio" id="guapo" name="opcion" value="guapo">
<label for="feo">Feo</label>
<input type="radio" id="feo" name="opcion" value="feo">
<p id="mensaje"></p>

<script>
  $(document).ready(function () {
    $('input[name="opcion"]').on("change", function () {
      if ($('#guapo').is(':checked')) {
        $('#mensaje').html('Eres guapo');
      } else if ($('#feo').is(':checked')) {
        $('#mensaje').html('Eres feo');
      }
    });
  });
</script>
```

Si lo que necesitamos es conocer el **valor** del *radiobutton* que está seleccionado, simplemente podemos usar el método `val()` para extraer su *value*.

Una variante del ejemplo anterior sería:

```
$('input[name="opcion"]').on("change", function () {
  $('#mensaje').text('Eres ' + $(this).val());
});
```

11. Modificar contenido dinámicamente

Aunque la documentación de jQuery no recoge una sección concreta sobre los métodos que se utilizan para añadir o eliminar elementos del DOM, los métodos de la siguiente tabla aparecen en la sección de manipulación del DOM.

Método	Explicación
`.before()`	Inserta el contenido especificado por el parámetro antes de cada elemento en el conjunto de elementos coincidentes.
`.insertBefore()`	Inserta cada elemento en el conjunto de elementos coincidentes antes del destino.
`.after()`	Inserta el contenido especificado por el parámetro después de cada elemento en el conjunto de elementos coincidentes.
`.insertAfter()`	Inserta cada elemento en el conjunto de elementos coincidentes después del destino.
`.prepend()`	Inserta contenido especificado por el parámetro al comienzo de cada elemento en el conjunto de elementos coincidentes.
`.prependTo()`	Inserta cada elemento en el conjunto de elementos coincidentes al principio del objetivo.
`.append()`	Inserta el contenido especificado por el parámetro al final de cada elemento en el conjunto de elementos coincidentes.
`.appendTo()`	Inserta cada elemento en el conjunto de elementos coincidentes al final del destino.
`.clone()`	Crea una copia profunda del conjunto de elementos coincidentes.
`.detach()`	Elimina el conjunto de elementos coincidentes del DOM, pero mantiene todos sus datos y eventos asociados.
`.remove()`	Elimina completamente el conjunto de elementos coincidentes del DOM.
`.empty()`	Elimina todos los nodos hijos del conjunto de elementos coincidentes del DOM.

Hay algunos pares de métodos que hacen lo mismo y solo se diferencia en la sintaxis. En la mayoría de los casos, usar una u otra función es equivalente. Son:

- `before()` e `insertBefore()`. Insertan contenido antes del elemento o elementos definidos por el selector.
- `after()` e `insertAfter()`. Insertan contenido después del elementos o elementos definidos por el selector.
- `prepend()` y `prependTo()`. Insertan contenido dentro del elemento, antes del contenido que ya tuviera el elemento o elementos definidos por el selector.
- `append()` y `appendTo()`. Insertan contenido dentro del elemento, después del contenido que ya tuviera el elemento o elementos definidos por el selector.

Un ejemplo del comportamiento de los métodos que insertan contenido antes lo tienes a continuación. Observa dónde se insertan los `<h3>` en el DOM en cada caso.

```
<style>
  div {
    border: 1px solid black;
    background-color: #ccc;
    margin-block: 0.5em;
  }
</style>

<input type="button" value="before" id="btn-before">
<input type="button" value="insertBefore" id="btn-insertBefore">
<input type="button" value="prepend" id="btn-prepend">
<input type="button" value="prependTo" id="btn-prependTo">

<div>Div 1</div>
<div>Div 2</div>
<div>Div 3</div>

<script>
  $(document).ready(function () {
    $("#btn-before").on("click", function () {
      $("div").before("<h3>before</h3>");
    });

    $("#btn-insertBefore").on("click", function () {
      $("<h3>insertBefore</h3>").insertBefore("div");
    });

    $("#btn-prepend").on("click", function () {
      $("div").prepend("<h3>prepend</h3>");
    });

    $("#btn-prependTo").on("click", function () {
      $("<h3>prependTo</h3>").prependTo("div");;
    });
  });
</script>
```

12. AJAX

AJAX (*Asynchronous JavaScript And XML*) es una técnica de desarrollo web que permite a las páginas web comunicarse con el servidor en segundo plano, sin necesidad de recargar la página. Esto permite una experiencia de usuario más fluida y rápida.

Todos los métodos que implementa jQuery sobre esta técnica están en la documentación. En esta sección veremos dos, `load()` y el más importante, `jQuery.ajax()` (o `$.ajax()`).

Algunos de los parámetros más importantes que se utilizan con `$.ajax()` son:

Parámetro	Descripción
`url`	Especifica la URL a la cual se realizará la solicitud. Es **obligatorio**.
`method`	Especifica el tipo de petición HTTP a realizar. Los valores posibles son "GET", "POST", "PUT" y "DELETE". El valor predeterminado es "GET".
`data`	Especifica los datos a enviar en la solicitud. Puede ser un objeto, una cadena de texto o un array.
`dataType`	Especifica el tipo de datos esperados en la respuesta. Los valores más comunes son "json", "xml", "html" y "text", que es el predeterminado.
`success`	Especifica la función a ejecutar en caso de que la solicitud sea exitosa. La función recibirá los datos de respuesta como argumento.
`error`	Especifica la función a ejecutar en caso de que la solicitud falle. La función recibirá el objeto `jqXHR` como argumento, que contiene información sobre el estado de la solicitud y el error ocurrido. `jqXHR` hace referencia al objeto `XMLHttpRequest` (ver en MDN).
`complete`	Especifica la función a ejecutar cuando la solicitud se haya completado, independientemente de si fue exitosa o falló. La función recibirá el objeto `jqXHR` y el estado de la solicitud como argumentos. Sería el equivalente a `finally` en un bloque `try…catch`.

12.1. Petición GET sin parámetros

En el siguiente ejemplo solicitamos la *url* externa *mi-otra-pagina.html*. Si no se produce error, guardamos la respuesta de la petición en un *div*.

```
<input type="button" id="boton" value="Cargar HTML externo">
<div id="cargaexterna" style="border: 1px solid black">Aquí se cargará el HTML externo</div>

<script>
  $(document).ready(function () {
    $("#boton").click(function () {
      $.ajax({
        method: "GET",
        url: "mi-otra-pagina.html",
        success: function (data, textStatus, jqXHR) {
          $("#cargaexterna").html(data);
        },
      });
    });
  });
</script>
```

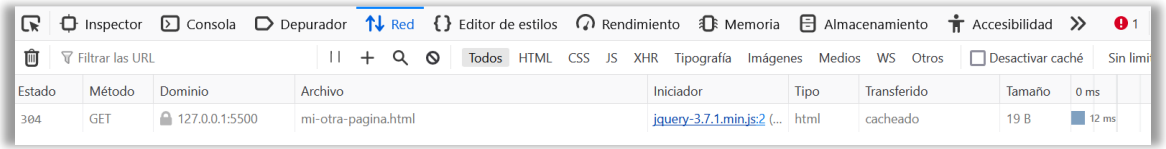
Si *mi-otra-pagina.html* es:

```
<h1>Hola Mundo</h1>
```

Entones el resultado de hacer clic en el botón es:

Con las *devtools* puedes ver que se ha insertado el contenido de *mi-otra-pagina.html* en el *div* con **id="cargaexterna"**:

También con las *devtools* puedes ver la petición en la pestaña *Red*.



El código de estado HTTP 304 significa “No modificado”. Esto indica que el recurso solicitado no ha sido modificado desde la última vez que fue solicitado por el cliente, por lo que el servidor no envía el contenido del recurso de nuevo. En su lugar, el servidor responde con un encabezado *304 Not Modified*, indicando que el cliente puede utilizar la copia en caché del recurso que ya tiene. Si modificas *mi-otra-pagina.html* y vuelves a pulsar el botón verás que el código recibido ahora es 200. Si el contenido se cachea o no se puede ver en la columna “Transferido”.

También podemos controlar qué ocurre si se produce un error en la llamada Ajax. Si cambiamos el nombre del recurso externo que vamos a cargar por uno que no exista, como *mi-otra-paginas.html*, entonces se producirá un error.

```
$.ajax({
  method: "GET",
  url: "mi-otra-paginas.html",
  success: function (data, textStatus, jqXHR) {
    $("#cargaexterna").html(data);
  },
  error: function (jqXHR, textStatus, errorThrown) {
    var errorMessage = "Error: " + jqXHR.status + ". " + errorThrown;
    $("#cargaexterna").html(errorMessage);
  }
});
```

12.2. Petición GET con parámetros

Vamos a ver cómo enviar parámetros y mostrar el resultado de la petición AJAX con método GET. Imagina el ejemplo anterior, pero ahora modificando la *url* a un fichero php y enviando datos de un supuesto formulario.

```
<input type="button" id="boton" value="Cargar HTML externo">
<div id="cargaexterna">Aquí se cargará el HTML externo</div>

<script>
  $(document).ready(function () {
    $("#boton").on("click", function () {
      $.ajax({
        method: "GET",
        url: "mi-otra-pagina.php",
        success: function (data, textStatus, jqXHR) {
          $("#cargaexterna").html(data);
        },
        data: { coche: "Ford", modelo: "Focus", color: "rojo" },
      });
    });
  });
</script>
```

Si el *php* fuera:

```
<?php
$coche = $_GET['coche'];
$modelo = $_GET['modelo'];
$color = $_GET['color'];
echo "Has escogido un $coche, modelo $modelo y color $color";
```

Obtendríamos “Has escogido un Ford, modelo Focus y color rojo”. Para probarlo es necesario utilizar un servidor local.

12.3. Recibiendo datos del servidor en JSON: `JSON.parse`

Ya sabemos mandar datos al servidor en una petición asíncrona, en esta sección veremos cómo tratar los datos que recibamos. Habitualmente los recibiremos en formato XML o, más comúnmente, JSON.

Para leer una respuesta recibida en formato JSON usaremos la función `JSON.parse`, que es un método de JavaScript que se utiliza para parsear (analizar) una cadena de texto en formato JSON y convertirla en un objeto JavaScript. Es una forma de convertir los datos en formato JSON recibidos por una llamada Ajax en un objeto manipulable en JavaScript.

Si la respuesta fuera en XML podemos parsearla mediante la función jQuery `jQuery.parseXML()`.

Supongamos que hemos realizado una llamada AJAX a un servidor y hemos recibido una respuesta en JSON con formato texto:

```
data = '{"nombre": "Juan", "apellidos": "Pérez", "edad": 32}'
```

Fíjate que lo que recibimos es un JSON con formato *string*. Si hubiéramos recibido el *string* "55" podríamos decir que hemos recibido un número en formato *string*, y para convertirlo a número deberíamos usar `parseInt()` o `parseFloat()`, dependiendo del tipo del número.

Para convertir este *string* en un objeto manipulable usaremos `JSON.parse()`:

```
var objeto = JSON.parse(data);
console.log(objeto.nombre); // imprimirá "Juan" en la consola
```

Después de parsear el texto JSON a un objeto, podemos acceder a sus propiedades mediante notación de punto, como se ve en el ejemplo anterior.

Veamos un **ejemplo más avanzado**. Tenemos un formulario de registro cuyos datos se deberán enviar con AJAX a la página *registro.php*, que devolverá un JSON con dos parámetros: "error" y "error_msg". Si "error" = 0, mostraremos un mensaje con el texto "Registro completado". Si error es distinto de 0, el mensaje será "Error XXX. YYY", donde XXX es el valor del parámetro "error" y YYY es el valor del parámetro "error_msg".

El código del formulario sería:

```
<form id="formulario-registro">
  <div>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre">
  </div>

  <div>
    <label for="apellidos">Apellidos:</label>
    <input type="text" id="apellidos" name="apellidos">
  </div>

  <div>
    <label for="fecha-nacimiento">Fecha de nacimiento:</label>
    <input type="date" id="fecha-nacimiento" name="fecha-nacimiento">
  </div>

  <div>
    <label for="sexo">Sexo:</label>
    <input type="radio" id="sexo-hombre" name="sexo" value="hombre">
    <label for="sexo-hombre">Hombre</label>
    <input type="radio" id="sexo-mujer" name="sexo" value="mujer">
    <label for="sexo-mujer">Mujer</label>
  </div>
  <input type="submit" value="Registrarse">

</form>
<div id="mensaje-registro"></div>
```

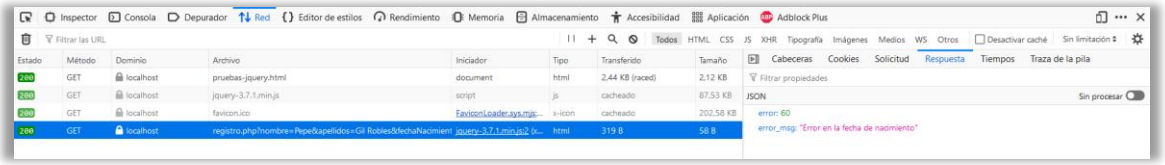
Y el jQuery:

```
$(document).ready(function () {
    $("#formulario-registro").on("submit", function (event) {
    event.preventDefault();

    var nombre = $("#nombre").val();
    var apellidos = $("#apellidos").val();
    var fechaNacimiento = $("#fecha-nacimiento").val();
    var sexo = $("input[name='sexo']:checked").val();

    $.ajax({
      url: "registro.php",
      type: "GET",
      data: {
        nombre: nombre,
        apellidos: apellidos,
        fechaNacimiento: fechaNacimiento,
        sexo: sexo
      },
      success: function (data) {
        var respuesta = JSON.parse(data);
        if (respuesta.error == 0) {
          $("#mensaje-registro").html("Registro completado");
        } else {
          $("#mensaje-registro").html("Error " + respuesta.error + ": " +
respuesta.error_msg);
        }
      }
    });
  });
});
```

En el servidor recibiríamos la petición AJAX, la procesaríamos y generaríamos una respuesta. Por ejemplo:

```
<?php
// Procesamiento de la petición...

// Variables que queremos empaquetar en JSON
$error = 60;
$error_msg = "Error en la fecha de nacimiento";

// Array asociativo con las variables
$datos = array(
  "error" => $error,
  "error_msg" => $error_msg
);

// Convertimos el array a JSON
$json_resultado = json_encode($datos);

// Devolver el resultado
echo $json_resultado;
```

Usando un servidor local obtendremos nuestro JSON de respuesta:



Para que el ejemplo funcione sin necesidad de levantar un servidor local, podemos modificar la página de destino de nuestra petición por *registro.txt* y hacer que ese fichero almacene un JSON con el formato previsto:

```
{ "error": 60, "error_msg": "Error en la fecha de nacimiento" }
```

Simplemente modificando este fichero JSON podremos observar cómo se gestiona la respuesta en jQuery.

12.4. Funciones *callback* `error` y `complete`

Mediante las funciones *callback error* y *complete* podemos controlar si ha habido errores o hacer algo cuando acabe la petición, haya ido bien o no. Para ver qué ocurre cuando hay un error puedes modificar la *url* para apuntar a un fichero inexistente, lo que nos devolverá un error 404. Veamos el ejemplo:

```
<input type="button" id="boton" value="Cargar HTML externo">
<div id="cargaexterna">Aquí se cargará el HTML externo</div>

<script>
  $(document).ready(function () {
    $("#boton").click(function () {
      $.ajax({
        method: "GET",
        url: "mi-otra-pagina.html",
        success: function (data, textStatus, jqXHR) {
          console.log("Todo ha ido bien");
          console.log(data);
          $("#cargaexterna").html(data);
        },
        error: function (jqXHR, textStatus, errorThrown) {
          console.log("Error!!!");
          console.log(jqXHR);
          console.log(textStatus);
          console.log(errorThrown);
        },
        complete: function (jqXHR, textStatus) {
          console.log("Esto se ejecutará siempre, haya error o no");
          console.log(jqXHR);
          console.log(textStatus);
        },
      });
    });
  });
</script>
```

12.5. La función `load()`

La función `load()` es un método que permite cargar y recuperar contenido HTML de una URL e insertarlo en el elemento seleccionado. Es una forma fácil y rápida de cargar contenido dinámico en una página sin tener que utilizar la función `$.ajax`.

```
<p>Queremos insertar contenido en el párrafo siguiente:</p>
<input type="button" value="Traer contenido" id="btn">
<p id="result"></p>

<script>
  $(document).ready(function () {
    $('#btn').on("click", function () {
      $("#result").load("mi-otra-pagina.html", function () {
        alert("Contenido cargado!");
      });
    });
  });
</script>
```

La función anónima que aparece como segundo parámetro es un *callback* que se ejecuta cuando la llamada asíncrona ha tenido éxito.

12.6. Diferencias entre `load()` y `$.ajax()`

Los métodos `load()` y `$.ajax()` permiten realizar solicitudes HTTP asíncronas, pero tienen diferencias en su uso y funcionalidad.

- `load()`:
  - Simplifica la realización de solicitudes HTTP para cargar contenido de manera asíncrona.
  - Se utiliza principalmente cuando se desea cargar contenido HTML de forma simple y directa en un elemento específico de la página.
  - Es más fácil de usar en comparación con `$.ajax()` cuando se trata de cargar contenido de una URL específica en un elemento del DOM.
- `$.ajax()`:
  - Es más flexible y poderoso que `load()` debido a su capacidad para manejar una variedad de opciones y configuraciones.
  - Se utiliza cuando se necesita control total sobre la solicitud y la respuesta.

En resumen, usar `load()` es buena idea cuando sólo se necesita cargar contenido HTML de una URL específica en un elemento del DOM de manera simple y directa. Por otro lado, es mejor usar `$.ajax()` cuando se necesita un mayor control sobre la solicitud y la respuesta HTTP, como manejar errores específicos, configurar encabezados personalizados o utilizar otros métodos HTTP además de GET.

13. Varios

13.1. La función `each()` y el objeto `$(this)`

La función `each` recibe como parámetro una función que se ejecutará una vez para cada uno de los elementos que nos devuelva el selector. Por ejemplo:

```
<div class="container">
  <p class="item">1</p>
  <p class="item">2</p>
  <p class="item">3</p>
  <p class="item">4</p>
</div>

<script>
  $(document).ready(function () {
    $(".item").each(function () {
      $(this).css("background-color", "lightgreen");
    });
  });
</script>
```

Este script pone como color de fondo `lightgreen` cada elemento con clase `item`. En este caso concreto el uso de `each()` es innecesario, ya que se podría conseguir el mismo objetivo con sólo una línea de código:

```
$(".item").css("background-color", "lightgreen");
```

Cada vez que se ejecuta una función, el objeto `$(this)` hace referencia al objeto que se está tratando actualmente de entre los devueltos por la selección. Un ejemplo lo encontramos en el código anterior, donde `$(this)` apunta a cada uno de los elementos devueltos por la selección `$(".item")`.

13.2. El objeto `this` con *arrow functions*

Fíjate en el siguiente código:

```
<input type="button" id="mi-boton-1" value="Botón 1">
<input type="button" id="mi-boton-2" value="Botón 2">

<script>
  $(document).ready(function () {
    $("#mi-boton-1").on("click", function () {
      $(this).css("background-color", "lightgreen");
    });
    $("#mi-boton-2").on("click", () => {
      $(this).css("background-color", "lightgreen");
    });
  });
</script>
```

Tenemos dos botones y dos manejadores del evento clic para cada uno de los dos botones en los que simplemente cambiamos el color de fondo del botón. El primer botón funciona, pero el segundo no.

El motivo es el comportamiento del objeto `this` con las funciones flecha. Fíjate si mostramos por consola el objeto `this` en cada caso:

```
$("#mi-boton-1").on("click", function () {
  console.log("Botón 1");
  console.log(this);
  $(this).css("background-color", "lightgreen");
});
$("#mi-boton-2").on("click", () => {
  console.log("Botón 2");
  console.log(this);
  $(this).css("background-color", "lightgreen");
});
```

En el primer caso, `this` hace referencia al contexto en el que se llama la función (en este caso, el contexto es el elemento que dispara el evento). En el segundo caso, sin embargo, `this` hace referencia al contexto en el que fue definida la función, en este caso, el objeto `html`, que es el objeto de nivel superior y al que hace referencia `$(document).ready()`.

Una explicación más completa aquí.

13.3. Diferencias entre `$(window)` y `$(document)`

En jQuery, `$(window)` y `$(document)` representan objetos diferentes en el DOM y están asociados con diferentes partes de la estructura de una página.

- `$(window)` selecciona el objeto que representa la ventana del navegador, el *viewport*. Se usar para manejar eventos relacionados con la ventana del navegador, como el redimensionamiento y desplazamiento de la ventana. Por ejemplo:

```
$(window).on('resize', function () {
  console.log('La ventana del navegador ha sido redimensionada.');
});
```

- `$(document)` selecciona el objeto que representa el documento HTML. Se utiliza para manejar eventos relacionados con el documento, como clics, teclas presionadas y otros eventos de usuario.

Por ejemplo:

```
$(window).on('resize', function () {
  console.log('La ventana del navegador ha sido redimensionada.');
});

$(document).on('click', function () {
  console.log('Se hizo clic en algún lugar del documento.');
});
```

13.4. *Event handlers* directos y delegados

Como ya hemos visto, un *event handler* (o manejador de eventos) es una función que se ejecuta como respuesta a un evento específico. Con jQuery podemos distinguir entre manejadores de eventos directos y delegados.

El método de jQuery que se encarga de asignar estos manejadores de eventos es `on()`. Si nos fijamos en la documentación del método: veremos que tiene esta pinta:

```
.on( events [, selector ] [, data ], handler )
```

Veamos qué es ese *selector* que vemos como parámetro opcional. La documentación del método nos dice:

If selector is omitted or is null, the event handler is referred to as direct or directly-bound. The handler is called every time an event occurs on the selected elements, whether it occurs directly on the element or bubbles from a descendant (inner) element.

When a selector is provided, the event handler is referred to as delegated. The handler is not called when the event occurs directly on the bound element, but only for descendants (inner elements) that match the selector. jQuery bubbles the event from the event target up to the element where the handler is attached (i.e., innermost to outermost element) and runs the handler for any elements along that path matching the selector.

Resumiendo, si ese parámetro *selector* se omite o es nulo, el *event handler* será **directo** y será invocado cada vez que ocurra un evento en los elementos seleccionados por el selector previo a la llamada del método. Pero cuando ese *selector* se indique explícitamente, el *event handler* se denominará **delegado** y el controlador no se invocará cuando el evento ocurra directamente en el elemento vinculado, sino sólo para los descendientes (elementos internos) que coincidan con el selector.

Por ejemplo, el siguiente código asigna un *event handler* directo sobre cualquier elemento `li` descendiente de un `ul`.

```
$("ul li").on('click', function () { });
```

La misma definición para para asignar un *event handler* delegado sería:

```
$('ul').on('click', 'li', function () { });
```

La **importancia** de los *event handler* delegados radica precisamente en su funcionamiento. Dado que el evento se gestiona en un elemento padre que ya existe en el DOM y no en los hijos, que pueden existir o no, son excelentes para manejar eventos en elementos creados dinámicamente sin tener que adjuntar manualmente manejadores de eventos cada vez que cambia la estructura del DOM.