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:

- Menos necesario que cuando surgió por la modernización de los estándares.
- Rendimiento no es óptimo.
- Su sintaxis que 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/.

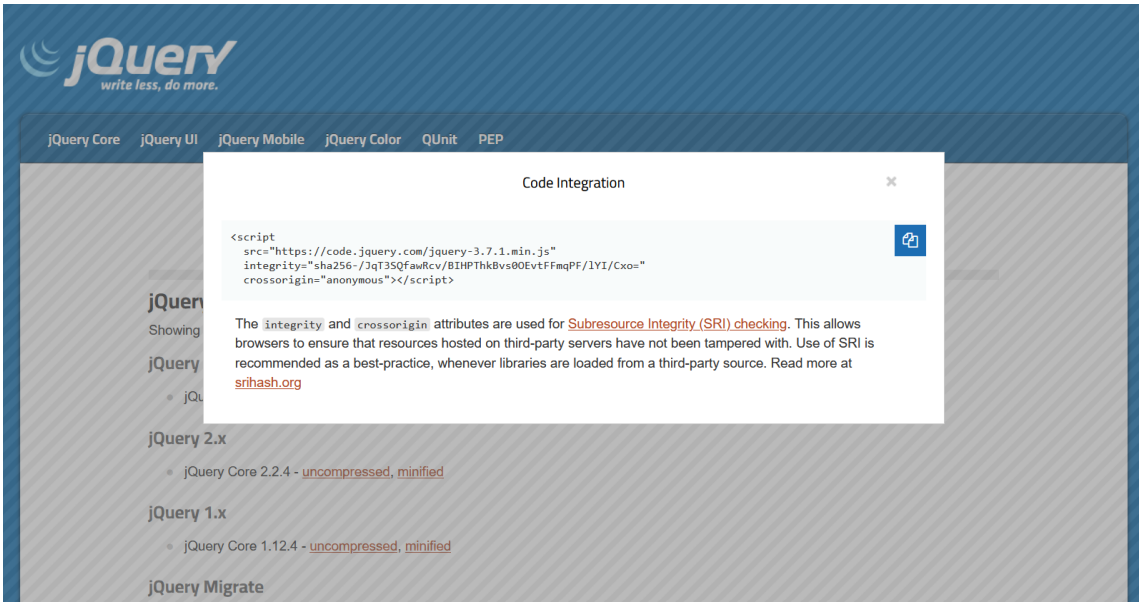
Nos dan a escoger entre descargar el archivo comprimido para producción, el no comprimido para desarrollo y el archivo *map*[1]. Dado que no vamos a revisar el código de jQuery, con el archivo comprimido (o *minificado*) nos vale.

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:

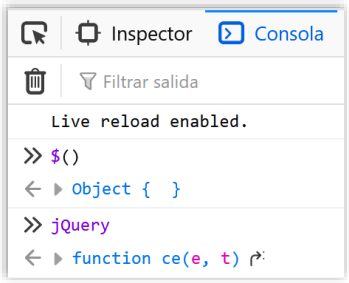
```
<script src="https://code.jquery.com/jquery-3.7.1.min.js" integrity="sha256-
/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
crossorigin="anonymous"></script>
```

[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.

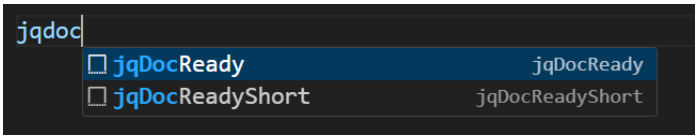
Para hacer nuestros primeros scripts con jQuery podremos optar por una de estas dos opciones, siendo ambas equivalentes:

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

O también:

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

En Visual Studio Code, podemos usar *Emmet* para escribir esto más rápidamente escribiendo `jqdoc`:



En general, podemos usar *Emmet* para abreviar cualquier función jQuery comenzando por escribir `jq`.

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:

```
>> $("p")
<- Object { 0: p.primero , 1: p , 2: p , 3: p , length: 4, prevObject: {…} }
     0: <p class="primero">
     1: <p>
     2: <p>
     3: <p>
     length: 4
     prevObject: Object { 0: HTMLDocument http://127.0.0.1:5500/5.%20jQuery/pruebas%20jquery.html, length: 1 }
     <prototype>: Object { jquery: "3.7.1", constructor: ce(e, t), length: 0, … }
```

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 loque 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. Métodos para manejar eventos

Revisa la documentación oficial de jQuery sobre eventos.

Como ya sabrás de JavaScript, los eventos son las distintas acciones que realizan los usuarios a las que ciertos elementos de una página web pueden responder.

Para asignar un evento a un elemento del DOM simplemente tenemos que aplicar el método jQuery relacionado con ese evento. Por ejemplo, digamos que queremos asignar un 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 se llama `click`. Este método recibe como parámetro un controlador de eventos (*event handler*), que es una función que se ejecuta cuando un evento específico se dispara en el elemento.

```
$(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");
  });
});
```

4.1. El método *on()*

El método `on()` se utiliza para establecer un controlador de eventos para uno o varios elementos específicos en una página web. La sintaxis para utilizar lo 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 nombre del evento para el que se establece el controlador de eventos (por ejemplo, `'click'`, `'keypress'`, `'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. En cambio, el método `click()` es un método específico para el evento `click` y solo se puede utilizar para asignar un controlador de eventos para ese evento.

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');
  }
});
```

4.2. Tipos de eventos

jQuery divide los eventos en 4 categorías:

- Eventos de ratón: `click`, `dblclick`, `mouseenter`, `mouseleave`...
- Eventos de teclado: `keypress`, `keydown`, `keyup`...
- Eventos de navegador: `resize` y `scroll`.
- Eventos de formulario: `submit`, `change`, `focus`...

Todos estos métodos son bastante autodescriptivos. Sin embargo, hay ciertas diferencias entre los métodos relacionados con los eventos de teclado.

Método	Descripción	Uso habitual
`keydown`	Se activa **cuando se pulsa** una tecla. Se activa **antes** de que el valor de la tecla se registre en el elemento de entrada.	Detectar acciones específicas del teclado (moverse a través de un formulario o activar una función con una tecla).
`keypress`	Se activa **cuando se presiona** y suelta una tecla. Se activa **después** de que el valor de la tecla se registre en el elemento de entrada.	Validar entradas de usuario, como limitar el número de caracteres que pueden ingresarse en un campo de texto
`keyup`	Se activa **cuando se suelta** una tecla. Se activa **después** de que el valor de la tecla se registre en el elemento de entrada	Detectar cuando el usuario ha terminado de ingresar un valor en un elemento de entrada.

Fíjate en el siguiente ejemplo, diseñado para aclarar estas diferencias:

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

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

    $("#textbox").on("keypress", function () {
      console.log("keypress event");
    });

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

Escribe un carácter alfanumérico en el *textbox*, podrás ver en consola `"keydown event"`, `"keypress event"` y `"keyup event"` en ese orden, `keydown` se activa primero cuando se presiona una tecla, `keypress` se activa cuando se presiona y suelta una tecla y `keyup` se activa cuando se suelta una tecla.

Parece no haber **diferencias entre `keypress` y `keyup`**, pero hay una muy importante. Volvamos al ejemplo anterior. Pulsa ahora una tecla que no sea alfanumérica. Una tecla de función (`F1`, `F2`…), `shift`, `alt`, `ctrl`… ¿Qué ocurre? Se muestra por consola `"keydown event"` y `"keyup event"`, pero ni rastro de `"keypress event"`. La página del método de la documentación de jQuery nos lo explica:

This is similar to the keydown event, except that modifier and non-printing keys such as Shift, Esc, and delete trigger keydown events but not keypress events.

Un evento HTML relacionado con estos 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="input-text">

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

El problema de usar este evento es que no podemos saber si el usuario ha pulsado una tecla no alfanumérica (`F1`, `F2`, `ctrl`, `shift`…). Para eso tendríamos que usar el evento `keydown` y acceder al *keycode* pulsado mediante `event.which`.

```
$("#input-text").on("keydown", function (event) {
  console.log(event.which);
});
```

Puedes leer más sobre este evento aquí.

5. 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()`	Determine 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.

6. 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 el padding, pero no el borde) 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 el padding, el borde y opcionalmente el margen) 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 el borde) 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 borde y opcionalmente el margen) 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

7. 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 solo 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:

```
<script>
  $(document).ready(function () {
    $("button").click(function () {
      $("#example").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.

8. Métodos para manejar formularios

Revisa la documentación oficial de jQuery sobre manejar formularios.

Método	Explicación
`.blur()`	Asigna un manejador de eventos al evento "blur" de JavaScript, o activa ese evento en un elemento. El evento blur se dispara cuando un elemento pierde el foco.
`.change()`	Asigna un manejador de eventos al evento "change" de JavaScript, o activa ese evento en un elemento.
`.focus()`	Asigna un manejador de eventos al evento "focus" de JavaScript, o activa ese evento en un elemento.
`.focusin()`	Asigna un manejador de eventos al evento "focusin".
`.focusout()`	Asigna un manejador de eventos al evento "focusout" de JavaScript.
`.select()`	Asigna un manejador de eventos al evento "select" de JavaScript, o activa ese evento en un elemento.
`.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()`	Asigna un manejador de eventos al evento "submit" de JavaScript, o activa ese evento en un elemento.
`.val()`	Obtiene el valor actual del primer elemento en el conjunto de elementos coincidentes o establece el valor de cada elemento coincidente.

Podría parecer que hay dos pares de eventos que hacen los mismo: `blur` y `focusout` por un lado y `focus` y `focusin` por otro. Lo cierto es que no son iguales, pero para explicar la diferencia tenemos que introducir el concepto de ***event bubbling***.

Event bubbling es un concepto en el desarrollo de JavaScript que se refiere a cómo los eventos se propagan desde el elemento en el que se originaron hasta los elementos padre en la jerarquía del DOM. 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`, primero se activará el evento asociado al `div` y luego el evento asociado al `form`.

En cuanto a los eventos jQuery que hablábamos:

- `blur` se activa cuando un elemento pierde el foco, mientras que `focusout` se activa cuando un elemento o uno de sus descendientes pierde el foco.
- `focus` se activa cuando un elemento recibe el foco, mientras que `focusin` se activa cuando un elemento o uno de sus descendientes recibe el foco.

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

```
<div id="padre">
  <input id="hijo1" type="text" value="Hijo 1"><br>
  <input id="hijo2" type="text" value="Hijo 2"><br>
</div>
<input id="hijo2" type="text" value="Un vecino">
<script>
  $(function () {
    $("#hijo1").on("blur", function () {
      console.log("hijo1 perdió el foco");
    });
    $("#hijo1").on("focus", function () {
      console.log("hijo1 ganó el foco");
    });
    $("#padre").on("focusout", function () {
      console.log("El padre perdió el foco");
    });
    $("#padre").on("focusin", function () {
      console.log("El padre ganó el foco");
    });
  });
</script>
```

8.1. Comprobación de *checkboxes*

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

8.1.1. 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
```

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

```
<!-- Desmarcado al cargar la página -->
<input type="checkbox" name="cb" id="cb">

<!-- Marcado al cargar la página -->
<input type="checkbox" name="cb" id="cb" checked>
```

Podemos acceder a los atributos usando jQuery con `attr()`. Pero el problema está en que el atributo no refleja el estado del checkbox, sino el valor predeterminado al cargar la página. Así que hacer lo siguiente es un error:

```
if ($('#cb').attr('checked')) {
  console.log("IS checked!");
}
```

8.2. Comprobación de *radiobuttons*

Para comprobar cuál de los *radiobuttons* de un grupo es el que está seleccionado debemos usar 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 () {
    $('input[name="opcion"]').change(function () {
      if ($('#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:

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

9. 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. 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. Puedes ir descomentando una a una las líneas comentadas para ver dónde se inserta el `<h2>` en el DOM en cada caso.

```
<input type="button" value="Insertar" id="btn">
<p>Párrafo 1</p>
<p>Párrafo 2</p>
<p>Párrafo 3</p>

<script>
  $(document).ready(function () {
    $("#btn").click(function () {
      // $("p").before("<h2>before</h2>");
      // $("<h2>insertBefore</h2>").insertBefore("p");
      // $("p").prepend("<h2>prepend</h2>");
      // $("<h2>prependTo</h2>").prependTo("p");
    });
  });
</script>
```

10. 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 su documentación. En esta sección solo veremos el más importante, `jQuery.ajax()`. Algunos de los parámetros más importantes que se utilizan 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.
`complete`	Especifica la función a ejecutar una vez que la solicitud se ha completado, independientemente de si fue exitosa o falló. La función recibirá el objeto jqXHR y el estado de la solicitud como argumentos.

10.1. Petición GET sin parámetros

En el siguiente ejemplo cargamos la url externa *mi-otra-pagina.html* y, si no se detecta error, ejecutamos una función que guardará el resultado en el parámetro que hemos pasado (*htmlexterno*) y lo utilizará para rellenar un div con el método *html*.

```
<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) {
          $("#cargaexterna").html(data);
        },
      });
    });
  });
</script>
```

10.2. 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").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.

10.3. 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.

Supongamos que hemos recibido una respuesta en formato JSON de una llamada Ajax a un servidor que contiene el siguiente texto:

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

Para convertir este texto en un objeto manipulable en JavaScript podemos hacer lo siguiente:

```
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. Supongamos que 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").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);
        }
      }
    });
  });
});
```

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": 0, "error_msg": "Error en la fecha de nacimiento" }
```

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

10.4. 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').click(function () {
      $("#result").load("ajax-test-load.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.

10.5. 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.

Veamos un ejemplo, modificando un poco el ejemplo que vimos en el apartado de petición GET sin parámetros:

```
<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>
```

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.

11. Otros

11.1. La función *each()*

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", "coral");
    })
  });
</script>
```

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

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

11.2. El objeto *$(this)*

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

11.3. El método *preventDefault()*

El método `preventDefault()` se utiliza para evitar que se ejecute el comportamiento predeterminado de un evento, permitiendo así al desarrollador manipular el comportamiento del evento de acuerdo a sus necesidades. Por ejemplo, si tiene un enlace en una página web y desea evitar que el usuario sea redirigido a otra página al hacer clic en el enlace, puede utilizar el método `preventDefault()` en el evento `click` del enlace para evitar la redirección.

```
$("a#link").click(function (event) {
  event.preventDefault();
});
```