TEMA 7

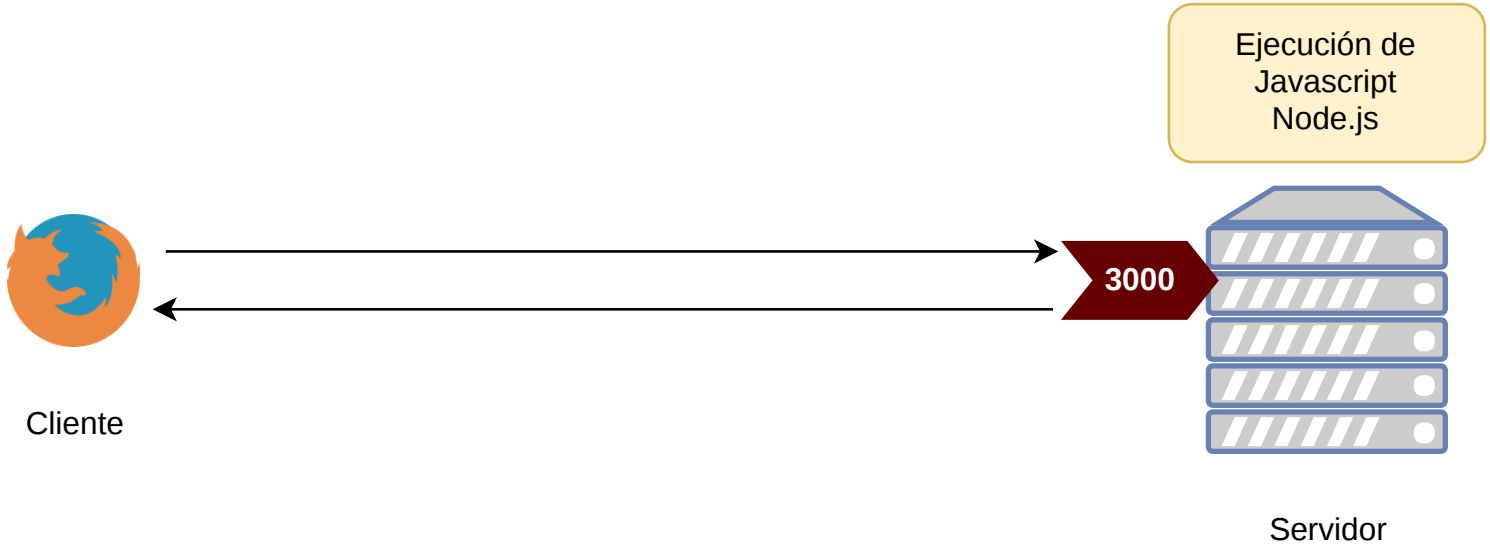
JAVASCRIPT EN EL NAVEGADOR: JQUERY

APLICACIONES WEB - GIS - CURSO 2017/18

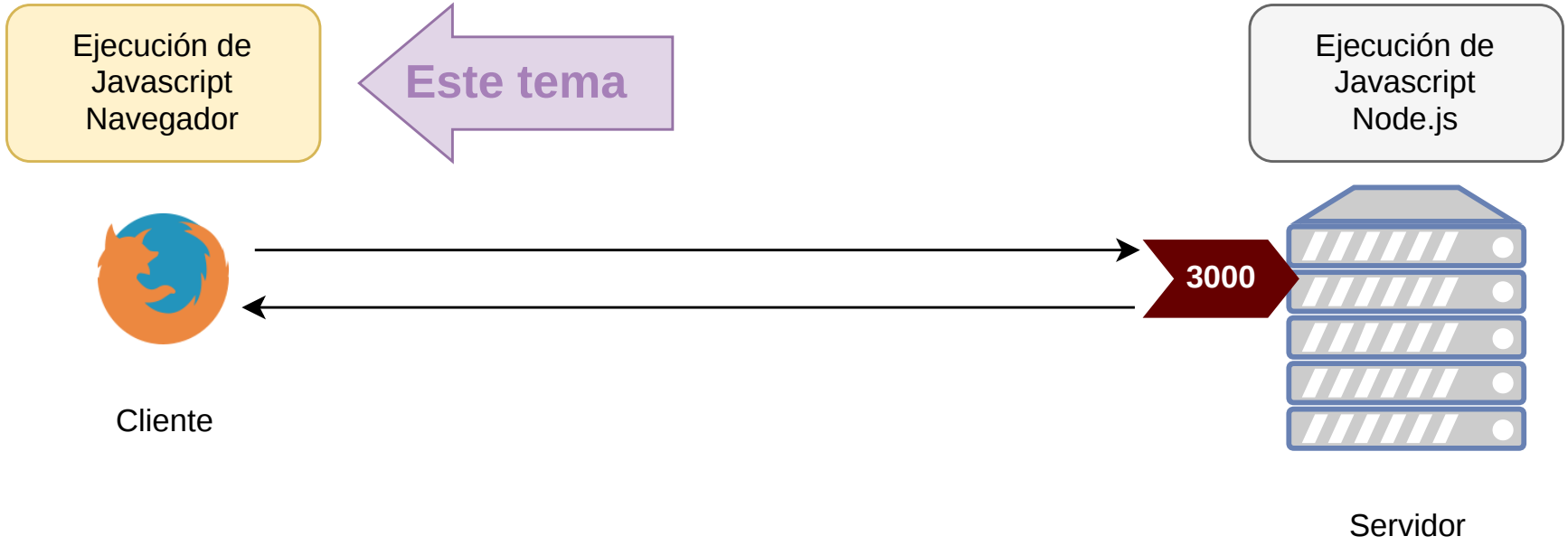
Esta obra está bajo una
Licencia CC BY-NC-SA 4.0 Internacional.

Manuel Montenegro [montenegro@fdi.ucm.es]
Dpto de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

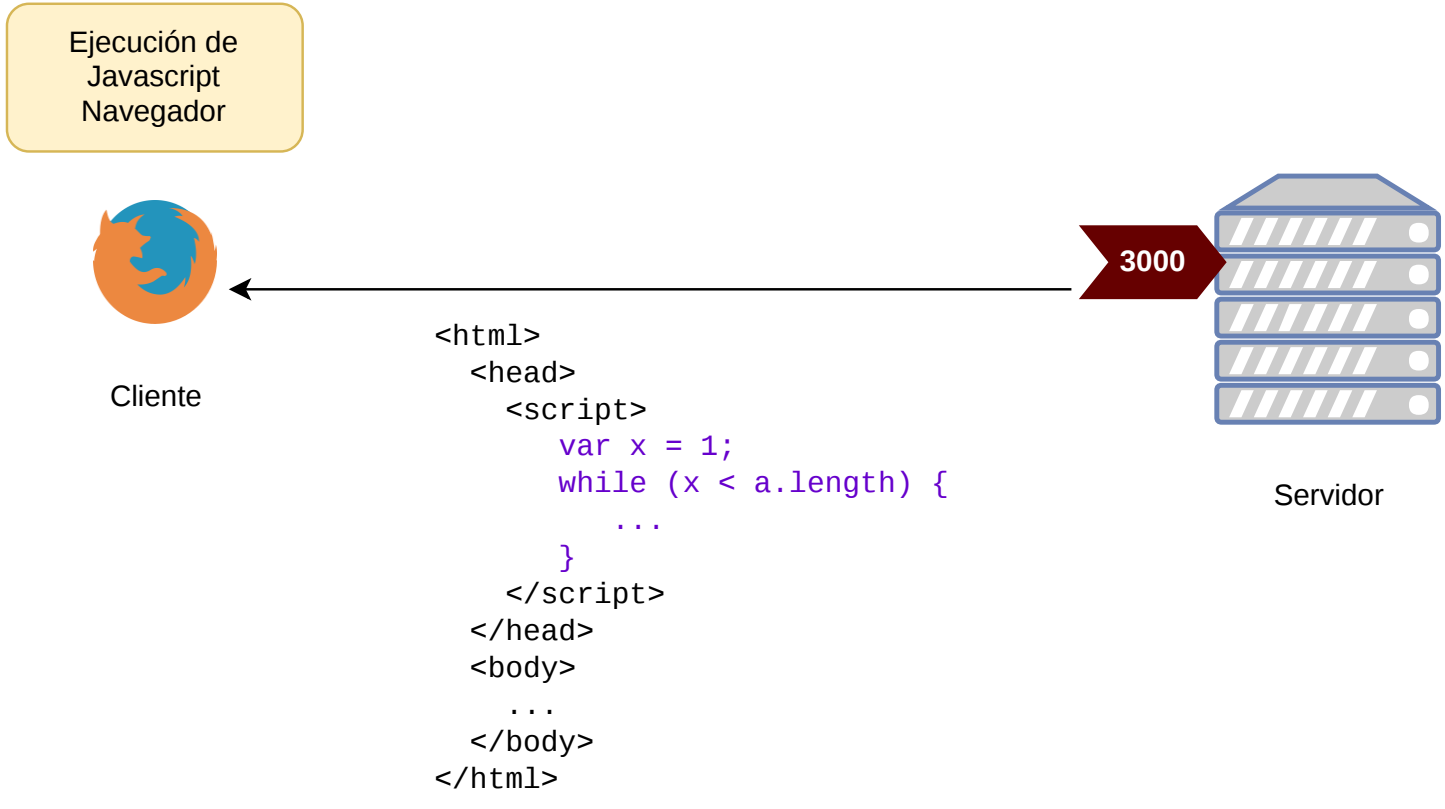
INTRODUCCIÓN



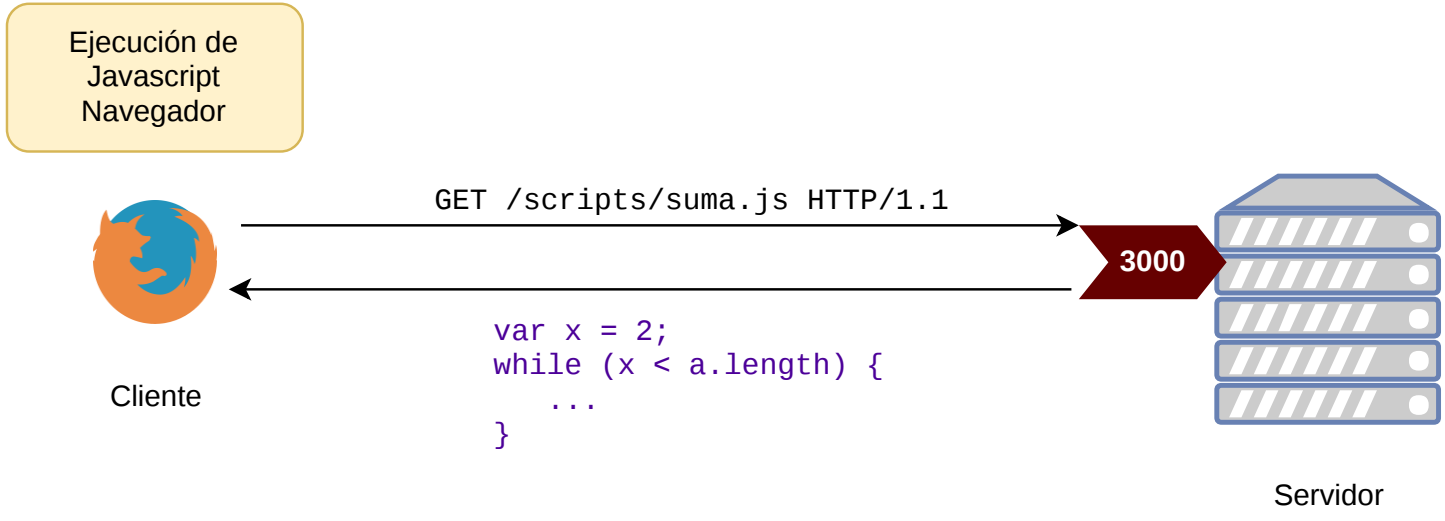
Temas 4, 5 y 6: hemos utilizado Javascript para implementar servidores HTTP y toda la funcionalidad que se ejecuta dentro del servidor: acceso a bases de datos, servir ficheros al cliente, lógica de aplicación, etc.



En **este tema** escribiremos programas en Javascript, pero pensados para ejecutarse en el navegador web.



Estos programas pueden estar integrados en un documento HTML proporcionado por el servidor...



...o bien pueden estar aparte, en ficheros proporcionados por el servidor, previa petición del navegador, por ejemplo, cuando aparece una etiqueta `<script>` con una referencia a un fichero `.js` externo:

```
<script src="scripts/suma.js"></script>
```

¡YA NO ESTAMOS EN NODE!

Aunque el código Javascript es proporcionado por el servidor, ahora se ejecutará en el navegador.

Podemos utilizar lo que hemos visto en el Tema 3 para realizar nuestros programas en el cliente...

...pero no las funciones específicas de Node.js y Express.js vistas en temas posteriores.

EN EL LADO DEL CLIENTE NO TENEMOS...

- El sistema de módulos visto en Node.js.
- La función `require` para importar módulos.
- La herramienta `npm`.
- Módulos core de node: `fs`, `path`, etc.
- Paquetes externos de Node.js: `mysql`, `express`, `ejs`, etc.
- etc.

EN CAMBIO, TENEMOS OTRAS COSAS NUEVAS

- Manipulación de elementos de la página web actualmente cargada (**DOM**).
- Funcionalidad relativa al historial de navegación (página anterior, siguiente, etc.)
- Peticiones al servidor: **AJAX** (tema siguiente).

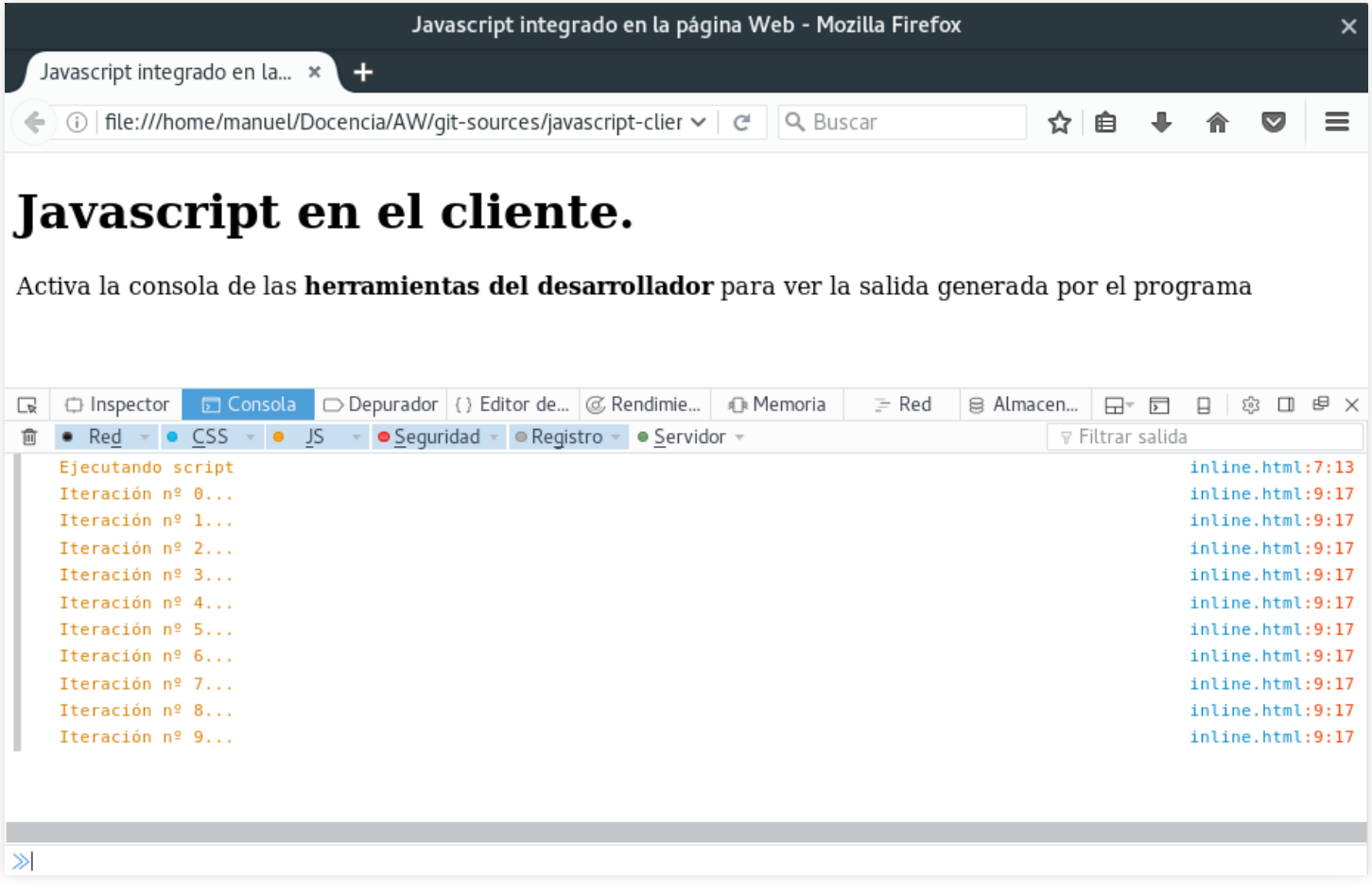
UN PRIMER EJEMPLO

Creamos un documento HTML con el siguiente contenido:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Javascript integrado en la página Web</title>
        <meta charset="utf-8">
        <script>
            console.log("Ejecutando script");
            for (let i = 0; i < 10; i++) {
                console.log(`Iteración nº ${i}...`);
            }
        </script>
    </head>

    <body>
        <h1>Javascript en el cliente.</h1>
        <p>Activa la consola de las <strong>herramientas del
        desarrollador</strong> para ver la salida generada por el
        programa</p>
    </body>
</html>
```

Abrimos la página en el navegador y utilizamos las **herramientas del desarrollador** para visualizar la consola.

En Firefox: *Ctrl+Mayús+K*



Lo más recomendable es escribir el código Javascript en un **fichero separado**, y hacer referencia a él desde la página web. Para ello se utiliza también la etiqueta `<script>`

```
<!DOCTYPE html>
<html>
    <head>
        <title>Javascript integrado en la página Web</title>
        <meta charset="utf-8">
        <script src="myscript.js"></script>
    </head>

    <body>
        <h1>Javascript en el cliente.</h1>
        <p>Activa la consola de las <strong>herramientas del
        desarrollador</strong> para ver la salida generada por el
        programa</p>
    </body>
</html>
```

Referencia a script

Fichero `myscript.js`:

```
// myscript.js
// -----------
"use strict";

console.log("Ejecutando script");
for (let i = 0; i < 10; i++) {
    console.log(`Iteración nº ${i}...`);
}
```

Es posible incluir varios scripts en un mismo documento:

```
<head>
    <script src="script1.js"></script>
    <script src="script2.js"></script>
    <script src="script3.js"></script>
    ...
</head>
```

Los scripts se cargarán y ejecutarán en el mismo orden en el que aparecen en el documento.

Atención: las variables globales definidas en un script son accesibles desde los scripts incluidos posteriormente.

1. INTRODUCCIÓN
2. OBJETOS PREDEFINIDOS
3. DOCUMENT OBJECT MODEL (DOM)
4. jQUERY
5. SISTEMA DE MÓDULOS: REQUIRE.JS
6. BIBLIOGRAFÍA

OBJETOS PREDEFINIDOS

Los siguientes objetos están disponibles en cualquier navegador:

- Variables y funciones globales: `window`
- Manejo del historial: `history`
- Información del navegador: `navigator`
- Información sobre la pantalla: `screen`

EL OBJETO `window`

Contiene todas las funciones y variables globales definidas, más algunos métodos predefinidos:

- `alert()`
- `setTimeout()`, `clearTimeout()`, `setInterval()`

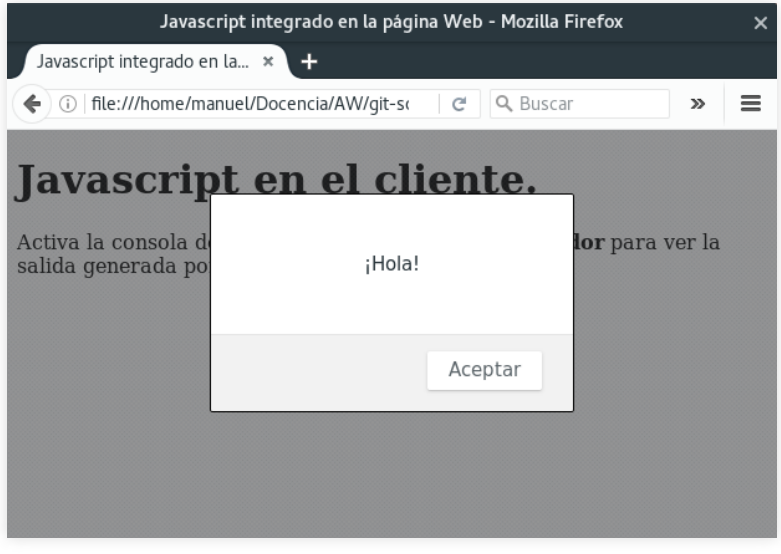
Toda variable y función global es implícitamente un atributo de `window`:

```
var str = "Mi cadena";
console.log(str);         // → Mi cadena
console.log(window.str); // → Mi cadena
```

EL MÉTODO `alert()`

Muestra una ventana emergente del navegador con un mensaje dado:

```
alert("¡Hola!");
```



La función es **bloqueante**: el script no continúa hasta que se pulse el botón *Aceptar*.

TEMPORIZADORES: `setTimeout()`, `setInterval()`

La llamada `setTimeout(fun, t)` ejecuta la función `fun` transcurrida una cantidad `t` de milisegundos.

```
function saludar() {
    console.log("Hola!");
}

setTimeout(saludar, 1000);
    // Imprime "¡Hola!" tras un segundo
```

equivalentemente:

```
setTimeout(() => { console.log("¡Hola!"); }, 1000);
    // Imprime "¡Hola!" tras un segundo
```

La función `setInterval(fun, t)` funciona de manera similar a `setTimeout`, pero ejecuta la función cada `t` milisegundos:

```
setInterval(saludar, 1000);
    // Imprime "¡Hola!" cada segundo.
```

Tanto `setTimeout()` como `setInterval()` devuelven un número identificador, que puede pasarse a `clearTimeout()` o `clearInterval()` para cancelar los temporizadores correspondientes:

```
// Saludamos cada 1000 milisegundos
let timerID = setInterval(saludar, 1000);

// Ponemos otro temporizador que, transcurridos 5000 ms,
// cancele el temporizador anterior:
setTimeout(() => { clearInterval(timerID); }, 5000);
```

EL MÉTODO open( )

Abre una ventana emergente con la URL dada.

¡Cuidado con los bloqueadores de ventanas emergentes!

```
var w1 = open("http://www.google.es");

var w2 = open(""); // Ventana en blanco
```

Devuelven un objeto de la clase `Window` [+], que sirve para manipular su contenido, posición, etc.

EL OBJETO `history`

Proporciona funcionalidad similar a los botones de *Atrás* y *Adelante* del navegador, que permiten navegar por las últimas páginas visitadas.

```
history.go(-1); // Navega a la última página visitada
history.go(-2); // Navega a la penúltima página visitada
history.go(-3); // Navega a la antepenúltima página visitada
...
history.go(1); // Página siguiente en el historial
               // (tras retroceder)
history.go(2);
...
history.back();  // equivale a history.go(-1)
history.forward();  // equivale a history.go(1)
```

EL OBJETO `navigator`

Proporciona información sobre el navegador:

```
console.log(navigator.appName);   // → Netscape
console.log(navigator.language);  // → es-ES
console.log(navigator.platform);  // → Linux x86_64
console.log(navigator.product);   // → Gecko
console.log(navigator.userAgent);
    // → Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:50.0)
    //   Gecko/20100101 Firefox/50.0
```

EL OBJETO `screen`

Contiene atributos sobre la resolución de pantalla:

- `width`, `height`: tamaño «total» de pantalla.

```
console.log(screen.width);  // → 1080
console.log(screen.height);  // → 1920
```

- `availWidth`, `availHeight`: tamaño disponible (excluyendo barra de tareas del SO, etc.)

```
console.log(screen.availWidth);  // → 1050
console.log(screen.availHeight);  // → 1920
```

1. INTRODUCCIÓN
2. OBJETOS PREDEFINIDOS
3. DOCUMENT OBJECT MODEL (DOM)
4. jQUERY
5. SISTEMA DE MÓDULOS: REQUIRE.JS
6. BIBLIOGRAFÍA

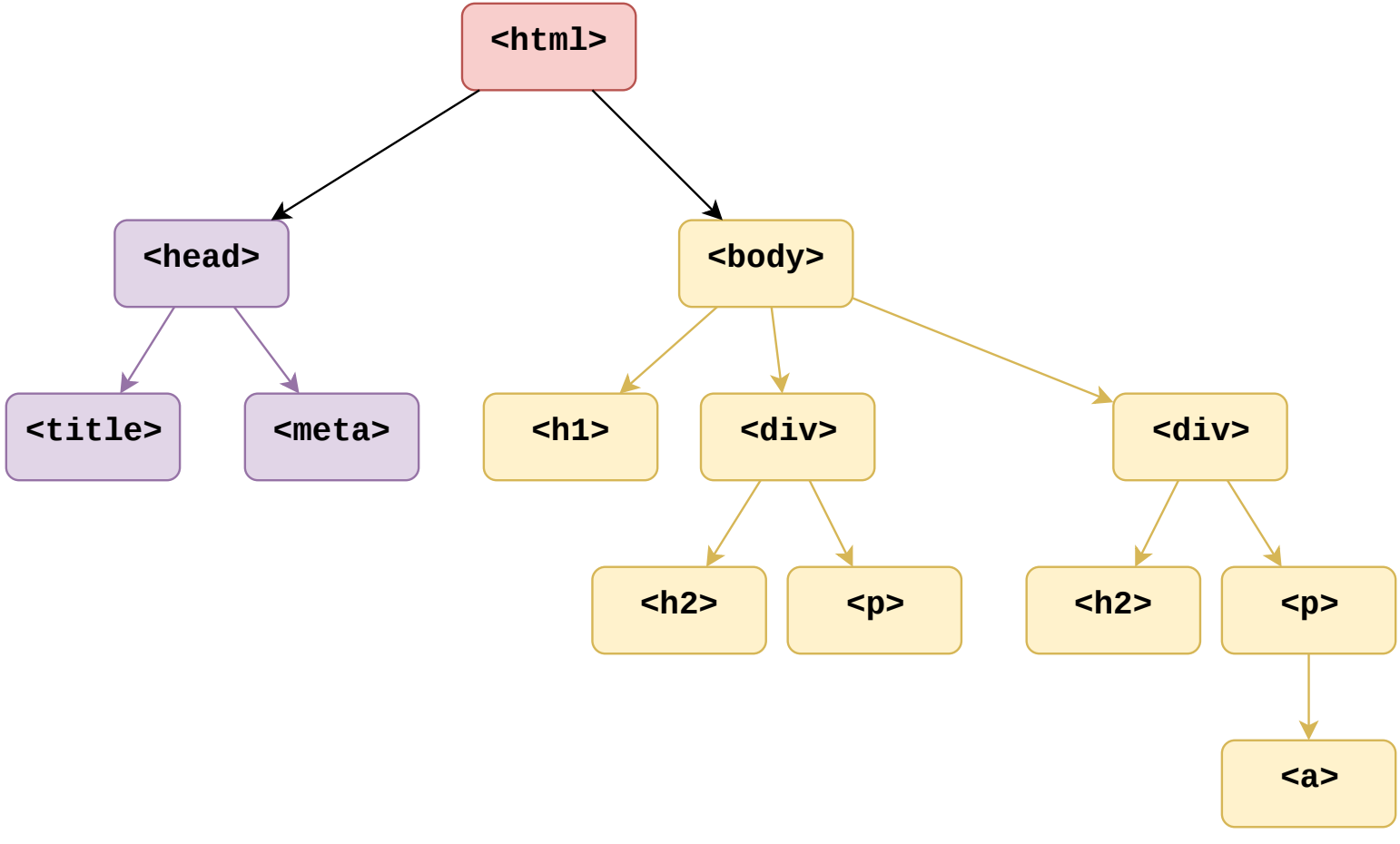
DOM (DOCUMENT OBJECT MODEL)

Un documento HTML consiste esencialmente en una serie de elementos anidados:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Bienvenido a mi página</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1 id="encabezado">¡Bienvenido!</h1>
        <div class="section">
            <h2>Novedades</h2>
            <p>No hay muchas novedades últimamente</p>
        </div>
        <div class="section">
            <h2>Contacto</h2>
            <p>Puedes contactar en la dirección de correo
            <a href="mailto:montenegro@fdi.ucm.es">
                    montenegro@fdi.ucm.es</a></p>
        </div>
```

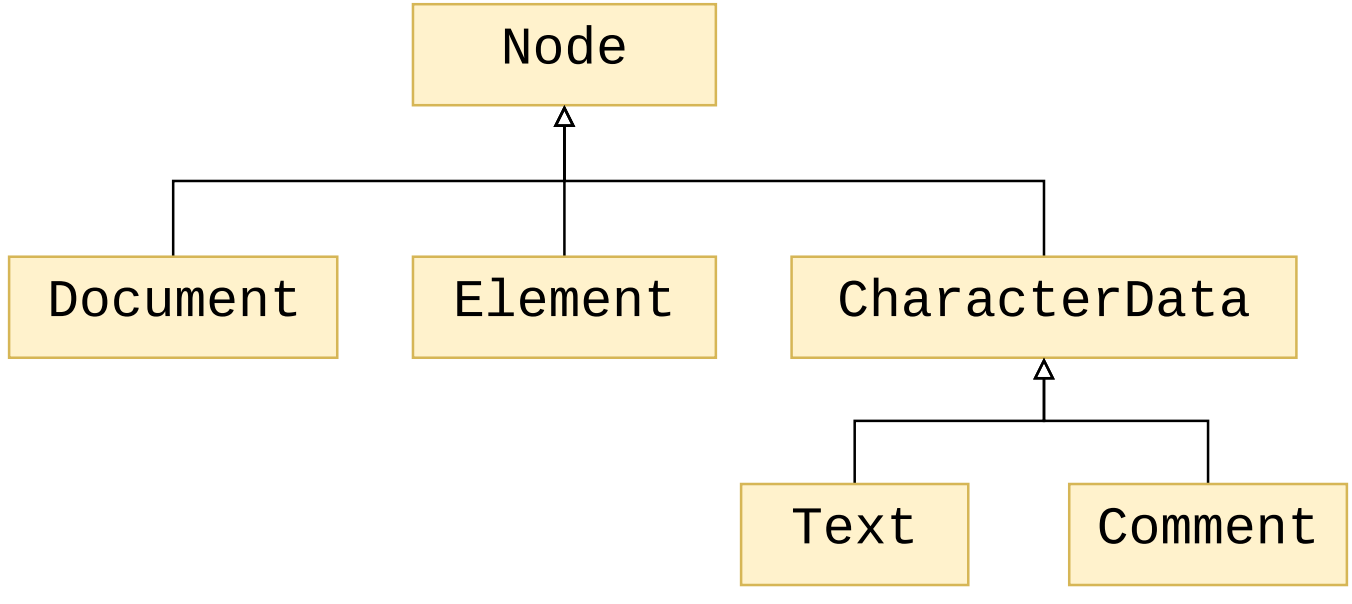
```
    </body>
</html>
```

Por tanto, un documento HTML puede visualizarse como una estructura en forma de árbol:



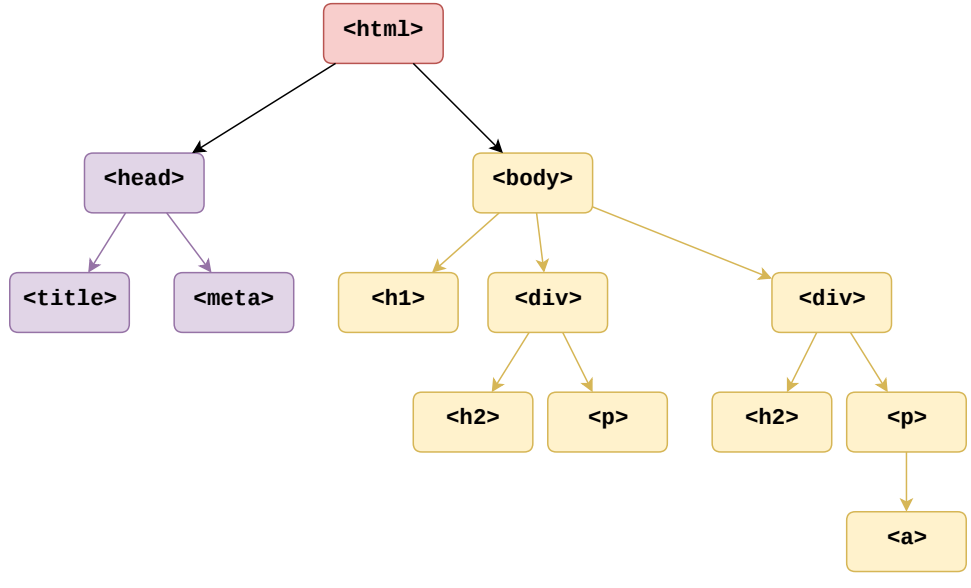
El **DOM** *(Document Object Model)* es una interfaz que permite nos permite acceder, navegar y modificar este árbol.

Esta interfaz se materializa en una serie de clases:



Cada nodo del árbol es una instancia de la clase `Node`.

La variable `document` contiene la raíz del árbol.



Los métodos de `document` permiten buscar elementos dentro del árbol. Entre ellos:

- `getElementById(id)`
- `getElementsByTagName(nombre)`
- `getElementsByClassName(nombre)`

```
let elemH1 = document.getElementById("encabezado");
```

Devuelve el objeto `Node` correspondiente al elemento HTML con el atributo `id="encabezado"`.

```
let elemsDiv = document.getElementsByTagName("div");
```

Devuelve un `NodeList` con todos los `<div>` del documento.

Los objetos `Node` tienen atributos y métodos para acceder y modificar las propiedades del nodo: estilo, posición, contenido, etc. Por ejemplo:

```
let elemH1 = document.getElementById("encabezado");
console.log(elemH1.textContent);    // → "¡Bienvenido!"
let elemsDiv = document.getElementsByTagName("div");
console.log(elemsDiv.length)  // → 2
```

La clase `Node` tiene atributos para navegar por el árbol DOM:

- `childNodes` (tipo `NodeList`)
- `firstChild` (tipo `Node`)
- `lastChild` (tipo `Node`)
- `nextSibling` (tipo `Node`)

Ver: https://developer.mozilla.org/es/docs/DOM

INCONVENIENTES DE UTILIZAR DOM

En la práctica no suelen utilizarse los métodos mostrados anteriormente para acceder directamente al árbol DOM.

Motivo: inconsistencias entre distintos navegadores.

Es más recomendable utilizar una librería Javascript que proporcione su propia API abstrayendo las diferencias entre los navegadores:

- Prototype.js
- jQuery

Este tema

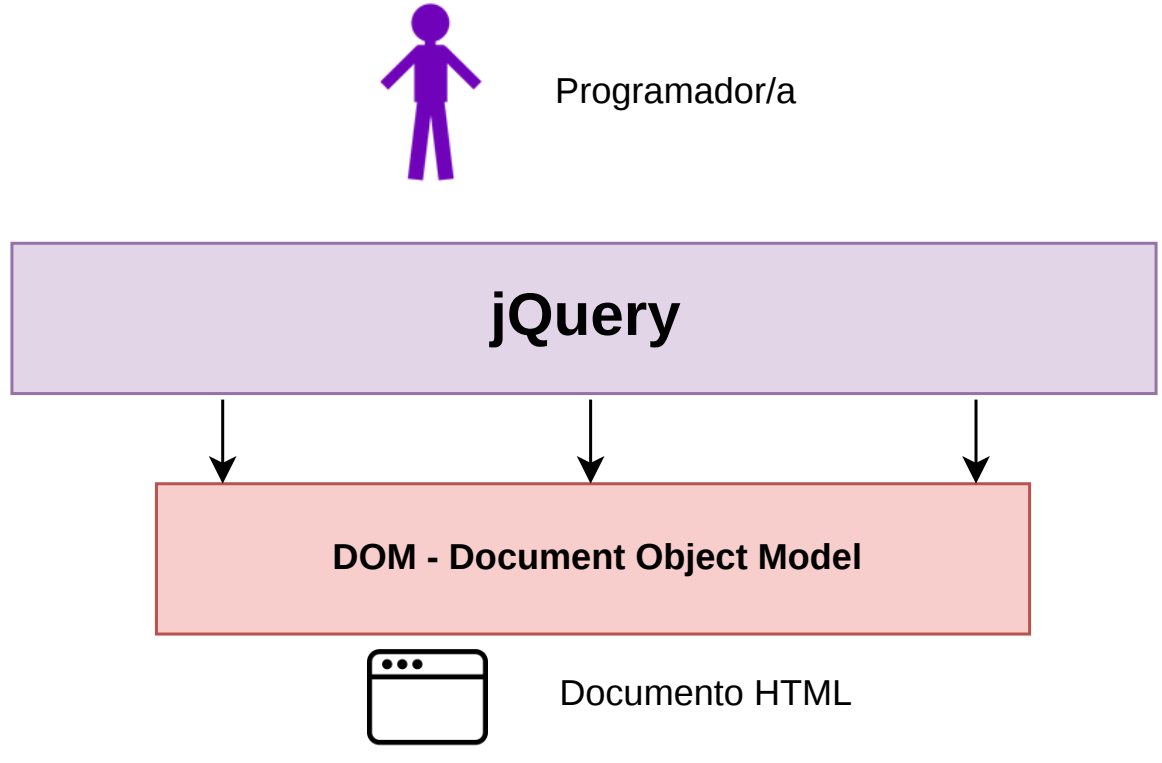
1. INTRODUCCIÓN
2. OBJETOS PREDEFINIDOS
3. DOCUMENT OBJECT MODEL (DOM)
4. jQUERY
5. SISTEMA DE MÓDULOS: REQUIRE.JS
6. BIBLIOGRAFÍA

jQUERY

Es una librería de Javascript que funciona en el lado del navegador web.

http://jquery.com/

Proporciona una capa de abstracción entre las funciones del DOM y el programador:



VENTAJAS

- Eliminación de inconsistencias entre navegadores.
- Código más claro y conciso.
- Extensible mediante *plugins*

INSTALACIÓN

https://jquery.com/download/

Descargamos la versión 3.2.1, disponible en el fichero `jquery-3.2.1.min.js`, o bien en el fichero `jquery-3.2.1.js`

Incluimos este fichero en el documento HTML:

```
<!DOCTYPE html>
<html>
    <head>
        ...
        <script src="jquery-3.2.1.min.js"></script>
    </head>
    ...
</html>
```

ALTERNATIVA MÁS SENCILLA: CDN

En lugar de descargar la librería e incluirla en nuestra aplicación web, es posible enlazar una copia de la librería directamente en un CDN:

```
<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
```

Un CDN (*Content Delivery Network*) es un conjunto de servidores que albergan copias del mismo contenido estático para optimizar su descarga por parte del cliente.

https://en.wikipedia.org/wiki/Content_delivery_network

LA VARIABLE $

Al cargar la librería jQuery se define una **variable global** nueva que contiene los métodos de jQuery.

El nombre de esta variable es `$`

(sí, `$` es un nombre de variable válido en Javascript)

Esta variable tiene varios usos:

- Como función: `$(document)`, `$("header")`, etc.
- Como objeto: `$.ajax()`, `$.get()`, etc.

Funciones disponibles:

- Inicialización de la página
- Búsqueda y navegación en el DOM
- Manipulación del DOM
- Creación de nuevos elementos en el DOM
- Posicionamiento y dimensiones
- Manejo de eventos
- Animaciones
- AJAX (siguiente tema)

INICIALIZACIÓN DE LA PÁGINA

A menudo queremos realizar acciones sobre el árbol DOM nada más cargarse la página.

Por ejemplo: mostrar los datos inciales, asociar callbacks con eventos, etc.

```
<html>
    ...
    <script src="init.js"></script>
    ...
    <button id="miBoton"> ... </button>
    ...
</html>
```

```
// init.js
let boton = document.getElementById("miBoton");
// ... configurar botón ...
```

¡Cuidado!

```html
<html>
    ...
    <script src="init.js"></script>
    ...
    <button id="miBoton"> ... </button>
    ...
</html>
```

```
// init.js
let boton = document.getElementById("miBoton");
// ... configurar botón ...
```

¿Cuándo se ejecuta este código? Puede que el *script* se ejecute antes de que el navegador haya cargado el DOM.

En este caso, `miBoton` no está disponible.

INICIALIZACIÓN DEL DOM

Con jQuery podemos asociar un *callback* a la inicialización del DOM. La función *callback* será llamada cuando el DOM se haya cargado en memoria.

La función `$()` sirve para esto.

```
$(callback);
```

EJEMPLO

```
// Indicamos al navegador que ejecute la función inicializar()
// cuando se cargue el DOM de la página.
$(inicializar);

function inicializar() {
    let boton = document.getElementById("miBoton");
    // ... configurar botón ...
}
```

Podemos introducir directamente una función anónima en la llamada a `$()`:

```
$(() => {
    let boton = document.getElementById("miBoton");
    // ... configurar botón ...
})
```

LAS MÚLTIPLES FACETAS DE $

1. `$(`*`funciónCallback`*`)`

   Acciones a realizar tras inicializar el DOM.

La lista continuará...

BÚSQUEDA Y NAVEGACIÓN EN EL DOM

Para manipular el DOM de la página con jQuery:

1. Seleccionar los elementos del árbol a modificar.
2. Aplicar las modificaciones que queramos sobre estos elementos.

En esta sección veremos solamente el primer paso.

Una **selección** es un conjunto de nodos del DOM.

En jQuery, las selecciones son instancias de una determinada clase, cuyo nombre es $.

A continuación veremos una función que nos permitirá obtener instancias de esta clase.

El nombre de esta función es... $.

SELECCIÓN A PARTIR DE UN ELEMENTO DEL DOM

Si hemos obtenido un elemento del árbol mediante las funciones del DOM de Javascript, podemos obtener una selección compuesta por ese único elemento.

```
let boton = document.getElementById("miBoton");
let seleccion = $(boton);

seleccion.css("color", "red");
```

¡No te encariñes con esto! En breve veremos una forma de seleccionar un elemento a partir de su identificador.

LAS MÚLTIPLES FACETAS DE $

1. `$(funciónCallback)`
   Acciones a realizar tras inicializar el DOM.
2. `$(elementoDOM)`
   Seleccionar un único elemento.

La lista continuará...

SELECCIÓN A PARTIR DE SELECTORES CSS

Es posible llamar a `$` pasando una cadena de texto como parámetro. Esta cadena recibe el nombre de **selector**.

```
$("div.entrada > input")
```

La sintaxis de los selectores es análoga a la utilizada en CSS.

Ver: selectores CSS

La expresión `$(...)` devuelve la selección que contiene los elementos afectados por el selector.

Una vez seleccionados los nodos, podemos utilizar las funciones de manipulación del DOM mencionadas anteriormente:

```
// Asigna un color de fondo a todos los elementos <li>
let seleccion = $("li");
seleccion.css("background-color", "#FFFFD0");

// Lo anterior es equivalente a esto.
$("li").css("background-color", "#FFFFD0");
// Utilizaremos esta forma a partir de ahora.

// Añade una clase a los elementos <p> de la clase 'entradilla'
$("p.entradilla").addClass("seleccionado");
```

SELECTORES DISPONIBLES

- Todos los nodos del DOM:

```
$("*")
```

- Todos los elementos `<p>` del documento:

```
$("p")
```

- Todos los elementos `<li>` del documento que tengan la clase `seleccionado`:

```
$("li.seleccionado")
```

- Todos los elementos con la clase `.seleccionado`.

```
$(".seleccionado")
```

- Elemento cuyo `id` sea `barra_principal`

```
$("#barra_principal")
```

Selección mediante identificador

- Elementos `<p>` con la clase `inciso` contenidos dentro de un elemento `<div>`:

```
$("div p.inciso")
```

- Igual que antes, pero solo los `<p>` que son hijos directos de `<div>`:

```
$("div > p.inciso")
```

- Elementos `<div>` que se encuentran después de un `<form>`.

```
$("form + div")
```

- Enlaces cuyo atributo `href` comience por `"http://"`.

```
$("a[href^='http://']")
```

Referencia: http://api.jquery.com/category/selectors/

RECORRIDO EN EL DOM

Algunos métodos de los objetos selección permiten obtener otras selecciones. Por ejemplo:

Hijos directos de los elementos `<header>`:

```
$("header").children()
```

Hijos de los elementos `<header>` que contengan la clase `seleccion`:

```
$("header").children(".seleccion")
```

Descendientes (directos o indirectos) de los elementos `<div>` que contengan la clase `seleccion`:

```
$("header").find(".seleccion")
```

Otras relaciones:

- `.parent()`
  Padres de los elementos seleccionados:
- `.parents(`*`[selector]`*`)`
  Antecesores de los elementos seleccionados:
- `.parentsUntil(selector)`
  Padres de los elementos seleccionados hasta llegar a un nodo que ajuste con el selector indicado.

Referencia: https://api.jquery.com/category/traversing/tree-traversal/

ACCEDER A LOS ELEMENTOS DE LA SELECCIÓN

- `.length`: Número de elementos seleccionados.

```
$("input[type='text'])").length
```

- `.eq(n)`: Elemento `n`-ésimo.

```
$("input[type='text'])").eq(2) // Segundo elemento
```

LAS MÚLTIPLES FACETAS DE $

1. `$(funciónCallback)`
   Acciones a realizar tras inicializar el DOM.
2. `$(elementoDOM)`
   Seleccionar un único elemento.
3. `$(selectorCSS)`
   Seleccionar uno o varios elementos.

La lista continuará...

MANIPULACIÓN DE ELEMENTOS DEL DOM

Las selecciones jQuery tienen métodos para manipular las propiedades y el estilo de todos los objetos seleccionados.

```
// Selector: elemento con ID "encabezado"
// Manipulación: cambiar su color de fondo a gris

$("#encabezado").css("background-color", "gray");
```

PROPIEDADES DE UN NODO DEL DOM

Sabemos que algunos elementos HTML tienen un conjunto de propiedades que depende del tipo de elemento:

- Imágenes: `src`, `alt`, `title`, ...
- Componentes de formularios: `type`, `checked`, `value`, etc.
- Enlaces: `href`, etc.
- Cualquier elemento: `class`, `id`, etc.
- etc.

El código HTML especifica sus valores iniciales,

```
<input type="text" name="edad" value="0">
```

pero estos valores pueden modificarse dinámicamente.

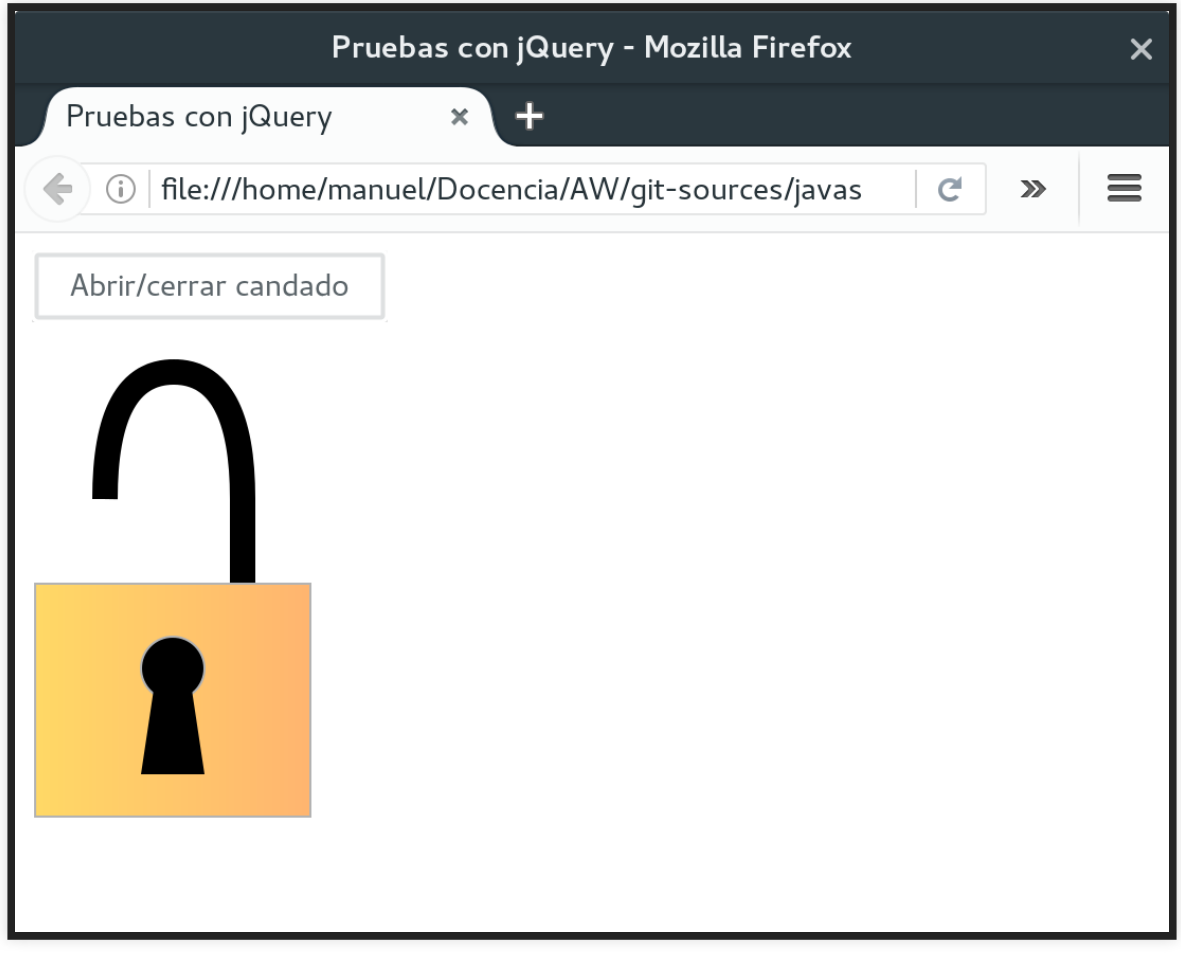
EL MÉTODO `prop`

`.prop(`*`nombrePropiedad, valorPropiedad`*`)`

Modifica una propiedad de los elementos seleccionados.

```
...
<img src="img/CandadoAbierto.svg" id="candado">
...
```

```
let candadoAbierto = true;

function cambiarCandado() {
    candadoAbierto = !candadoAbierto;
    if (candadoAbierto) {
        $("#candado").prop("src", "img/CandadoAbierto.svg");
    } else {
        $("#candado").prop("src", "img/CandadoCerrado.svg");
    }
}
```

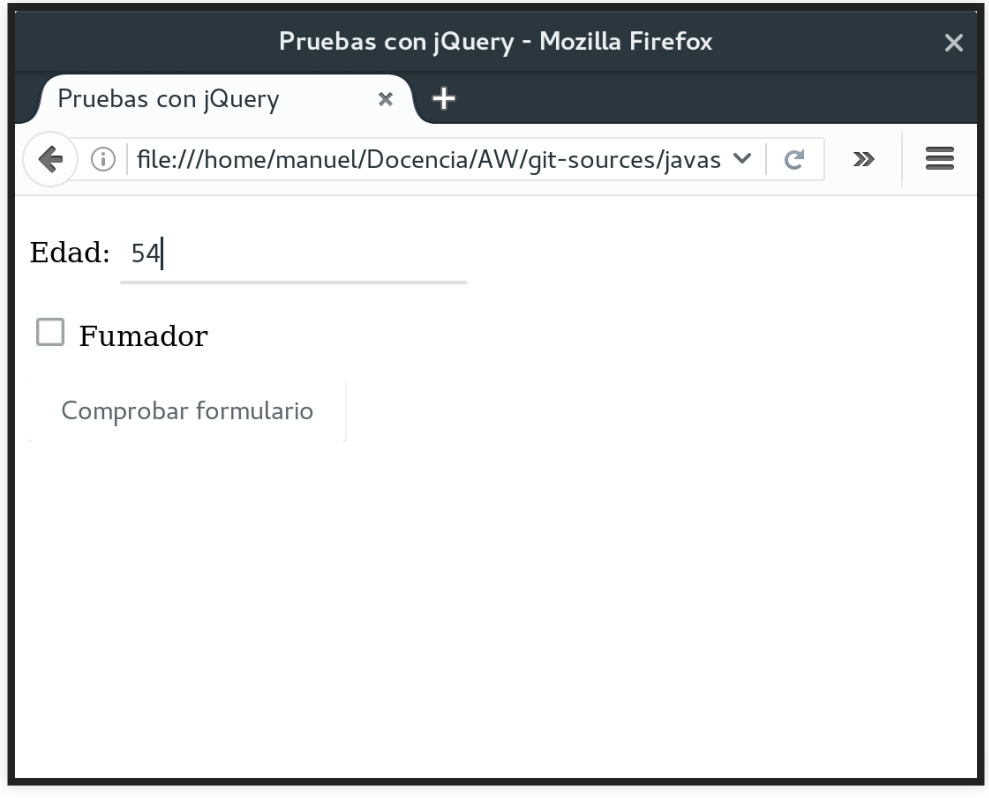

Pruebas con jQuery - Mozilla Firefox
Pruebas con jQuery
file:///home/manuel/Docencia/AW/git-sources/javas
Abrir/cerrar candado

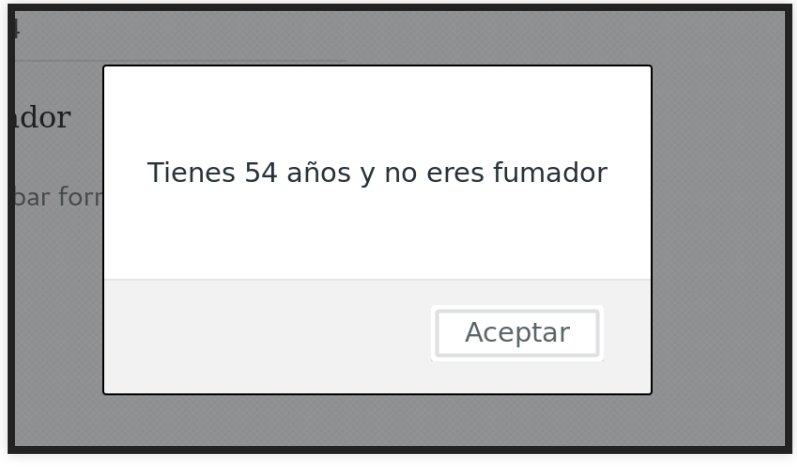
`.prop(`*`nombrePropiedad`*`)`

Obtiene el valor actual de la propiedad.

```
...
<p>Edad: <input type="text" name="edad" id="campoEdad"></p>
<p><input type="checkbox" name="fumador" id="campoFumador">
    Fumador</p>
...
```

```
function mostrarInfo() {
    let edad = $("#campoEdad").prop("value");
    let fumador = $("#campoFumador").prop("checked");
    alert(`Tienes ${edad} años y ` +
          `${fumador ? '' : 'no'} eres fumador`);
}
```


Pruebas con jQuery - Mozilla Firefox
Pruebas con jQuery
file:///home/manuel/Docencia/AW/git-sources/javas
Edad: 54
Fumador
Comprobar formulario


Tienes 54 años y no eres fumador
Aceptar

MODIFICAR CLASES Y ESTILOS

Se pueden añadir, modificar o eliminar clases CSS de los elementos utilizando `.prop("class", "...")`.

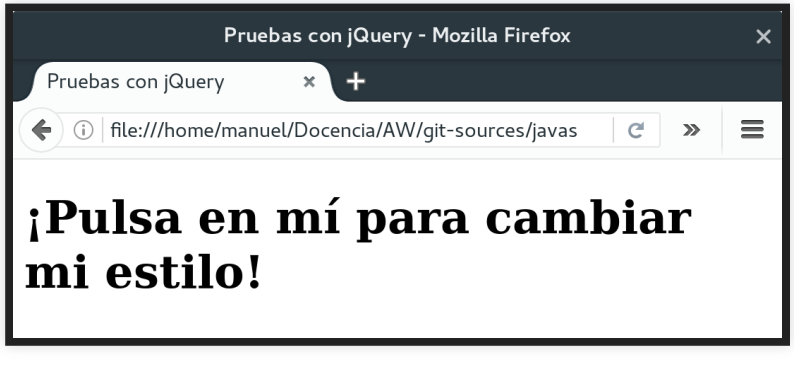
Pero es más fácil utilizar los siguientes métodos:

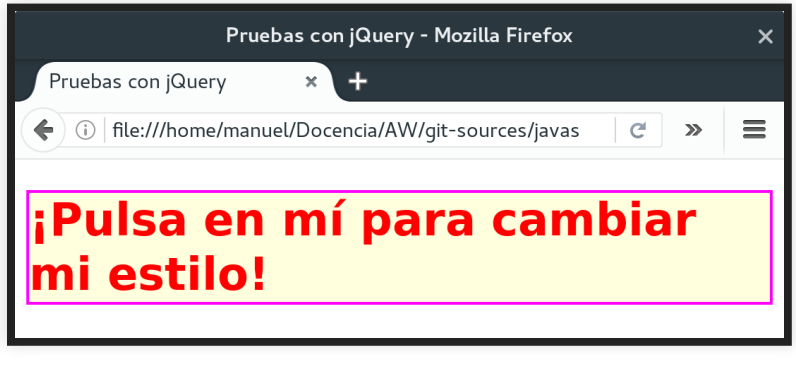
- `.addClass(`*`nombreClase`*`)`
- `.removeClass(`*`nombreClase`*`)`
- `.toggleClass(`*`nombreClase`*`)`
- `.hasClass(`*`nombreClase`*`)`

EJEMPLO

```
<h1>¡Pulsa en mí para cambiar mi estilo!</h1>
```

```
$(() => {
    let cabecera = $("h1");
    cabecera.on("click", () => {
        cabecera.toggleClass("rojo");
    });
});
```





Mediante el método `.css()` podemos obtener y modificar una propiedad CSS de modo individual.

`.css(nombrePropiedad, valorPropiedad)`

```
// Todos los encabezados <h1> tendrán el texto de color verde
$("h1").css("color", "green");
```

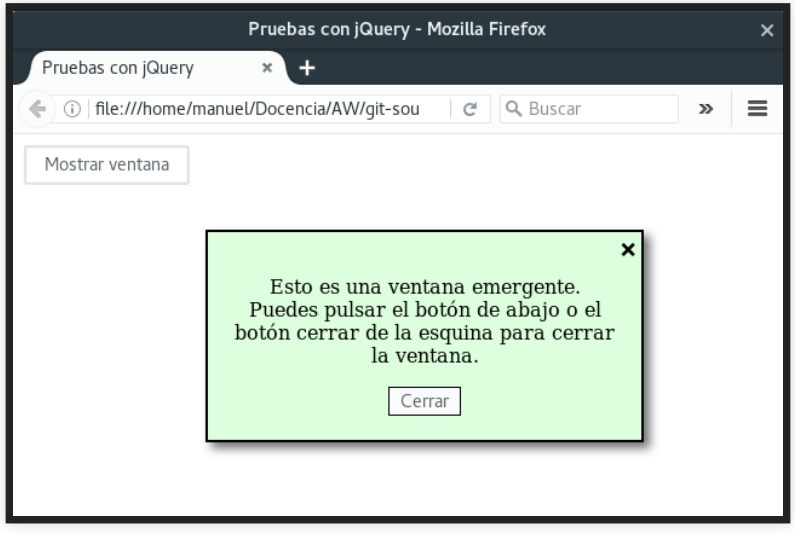
`.css(nombrePropiedad)`

```
console.log($("h1").eq(0).css("font-size")); // → "32px"
```

MOSTRAR/OCULTAR ELEMENTOS

Puede utilizarse el método `.css("display", "...")`, pero es más conveniente utilizar los métodos `.show()` y `.hide()`

EJEMPLO




```
<button id="mostrarVentana">Mostrar ventana</button>
<div id="ventana">
    <p>
        Esto es una ventana emergente. Puedes pulsar el botón de
        abajo o el botón cerrar de la esquina para cerrar la
        ventana.
    </p>
    <button id="cerrar">Cerrar</button>
    <span class="cerrar">&#x274C;</span>
</div>
```

```
function abrirVentana() {
    $("#ventana").show();
}

function cerrarVentana() {
    $("#ventana").hide();
}

$(document).ready(function() {
    $("#mostrarVentana").on("click", abrirVentana);
    $("#ventana span.cerrar").on("click", cerrarVentana);
    $("#cerrar").on("click", cerrarVentana);
});
```

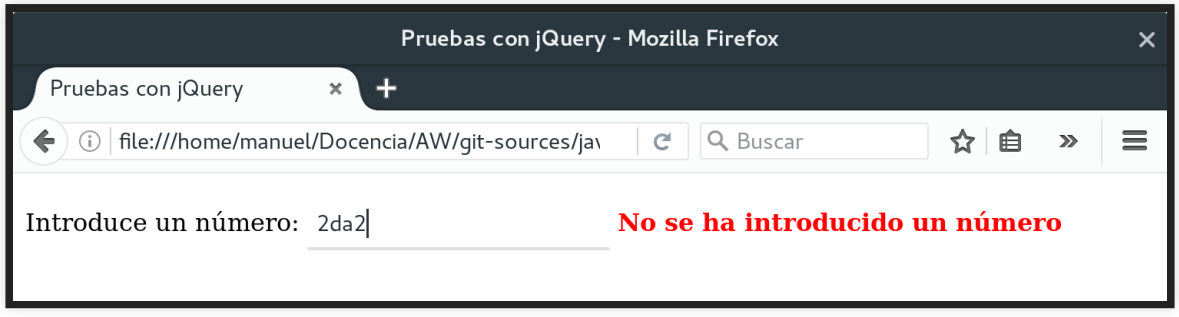
CAMBIAR EL CONTENIDO DE UN ELEMENTO

Obtener contenido contenido en el interior de un elemento:

```
.text()
```

Modificar contenido en el interior de un elemento:

```
.text(nuevoTexto)
```




```html
<p>
    Introduce un número: <input type="text" id="campoNumero">
    <span id="mensaje" class="incorrecto"></span>
</p>
```

```javascript
$("#campoNumero").on("change", function() {
    // Obtenemos valor actual
    let valor = $(event.target).prop("value").trim();

    if (valor === "") {
        $("#mensaje").text("El campo está vacío");
    } else if (isNaN(Number(valor))) {
        $("#mensaje").text("No se ha introducido un número");
    } else {
        $("#mensaje").text("");
    }
});
```

Si se introducen etiquetas HTML en la cadena pasada a `.text()`, éstas se «escapean», de manera que se evita la inyección de código HTML en el elemento seleccionado.

```
$("#mensaje").text("El campo está <em>vacío</em>");
```

En el caso en que queramos que se interpreten las etiquetas, debemos utilizar el método `.html()`

```
$("#mensaje").html("El campo está <em>vacío</em>");
```

PROPIEDADES DEFINIDAS POR EL USUARIO

Además de las propiedades predefinidas en los elementos del DOM (`class`, `value`, `src`, etc.), el programador puede añadir propiedades a cualquier elemento del DOM.


```
<p id="elem">Esto es un elemento del DOM que almacena un número</p>
<button id="incrementar">Incrementar</button>
<button id="obtener">Obtener valor actual</button>
```

.data(*nombreAtributo*)

Obtiene el valor de una propiedad personalizada del elemento seleccionado.

```
// Obtenemos el valor inicial del atributo 'number' contenido
// dentro del elemento #elem

var n = $("#elem").data("number");
```

.data(*nombreAtributo, nuevoValor*)

Cambia el valor de una propiedad personalizada de los elementos seleccionados.

```
// Al pulsar el botón Incrementar, se incrementan la propiedad
// 'number' del párrafo.
$("#incrementar").on("click", () => {
    let elemento = $("#elem");
    let num = elemento.data("number");
    elemento.data("number", num + 1);
});

// Al pulsar el botón Obtener, se muestra el valor actual de la
// propiedad 'number' del párrafo
$("#obtener").on("click", () => {
    alert($("#elem").data("number"));
});
```

Es posible especificar los valores iniciales de este tipo de propiedades en el propio documento HTML. Para ello basta con añadir un atributo a la etiqueta correspondiente con el nombre de la propiedad precedida por el prefijo `data-`

```
<p id="elem" data-number="1">
        Esto es un elemento del DOM que almacena un número
</p>
```

Si hacemos esto en nuestro ejemplo, ya no será necesario inicializar este atributo en el código Javascript:

```
// Ya no es necesario inicializar la propiedad en el código
$("#elem").data("number", 1 );
```

COMBINAR OPERACIONES DE MANIPULACIÓN

Todas las operaciones de manipulación vistas hasta ahora pueden encadenarse para ser aplicadas en una misma selección.

```
$("ul > li.opcion").addClass("seleccionada").data("numOpcion", 0);
```

```
$("img").prop("width", "90%")
        .css("border-bottom", "2px solid black")
        .css("border-top", "1px solid red")
        .on("click", pulsarImagen);
```

CREACIÓN E INSERCIÓN DE NUEVOS ELEMENTOS EN EL DOM

CREACIÓN DE NODOS

También podemos aplicar la función $ a una cadena de texto que contenga etiquetas HTML.

Al hacer esto, se crea un árbol DOM que refleja dicho código y se devuelve su selección.

```
let seleccion = $("<div>Nuevo elemento</div>");
```

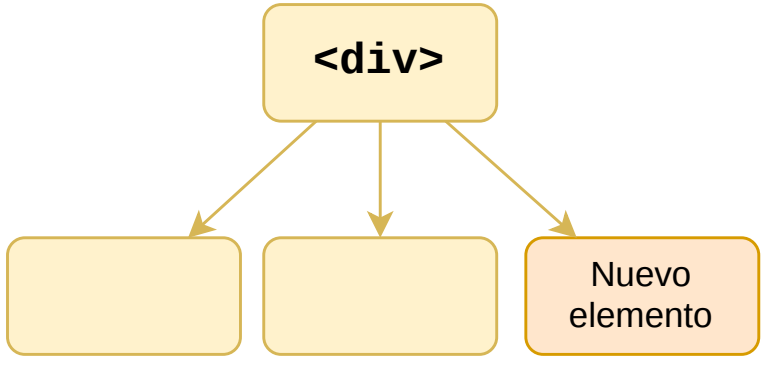
```
$("<img>").prop("src", imagen).addClass("imagen-ciudad");
```

Estos elementos no forman parte del documento HTML actual, a menos que se añadan al árbol DOM.

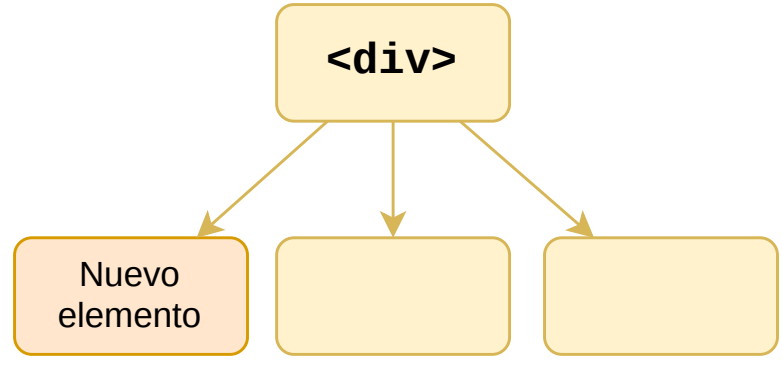
AÑADIR NODOS AL DOM

.append(*elem*)
.prepend(*elem*)

```
$("div").append(elem);
```

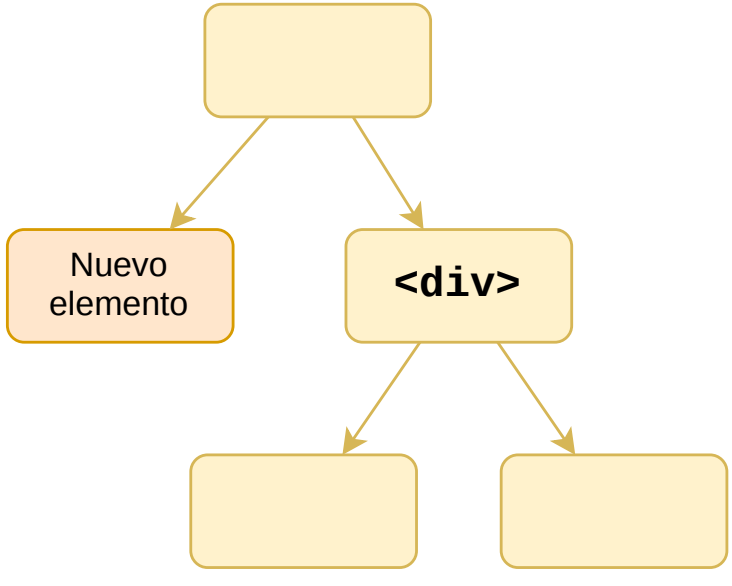



```
$("div").prepend(elem);
```

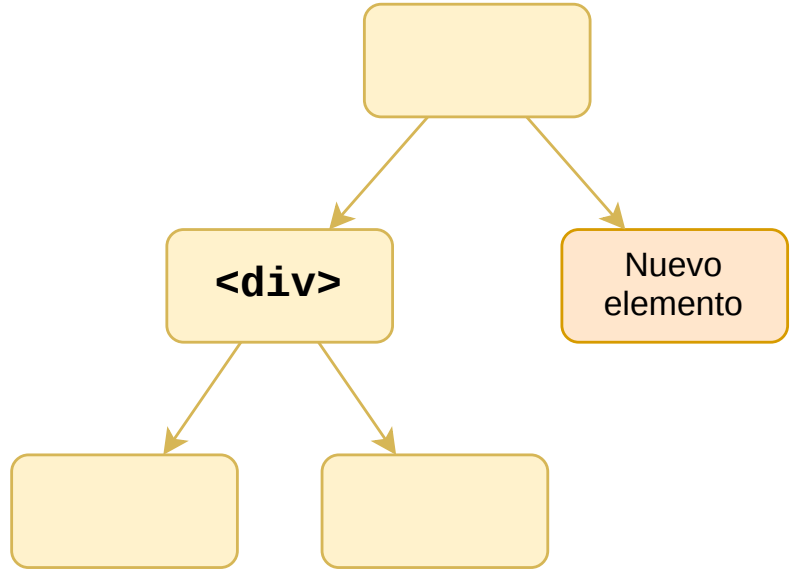


.before(*elem*)
.after(*elem*)

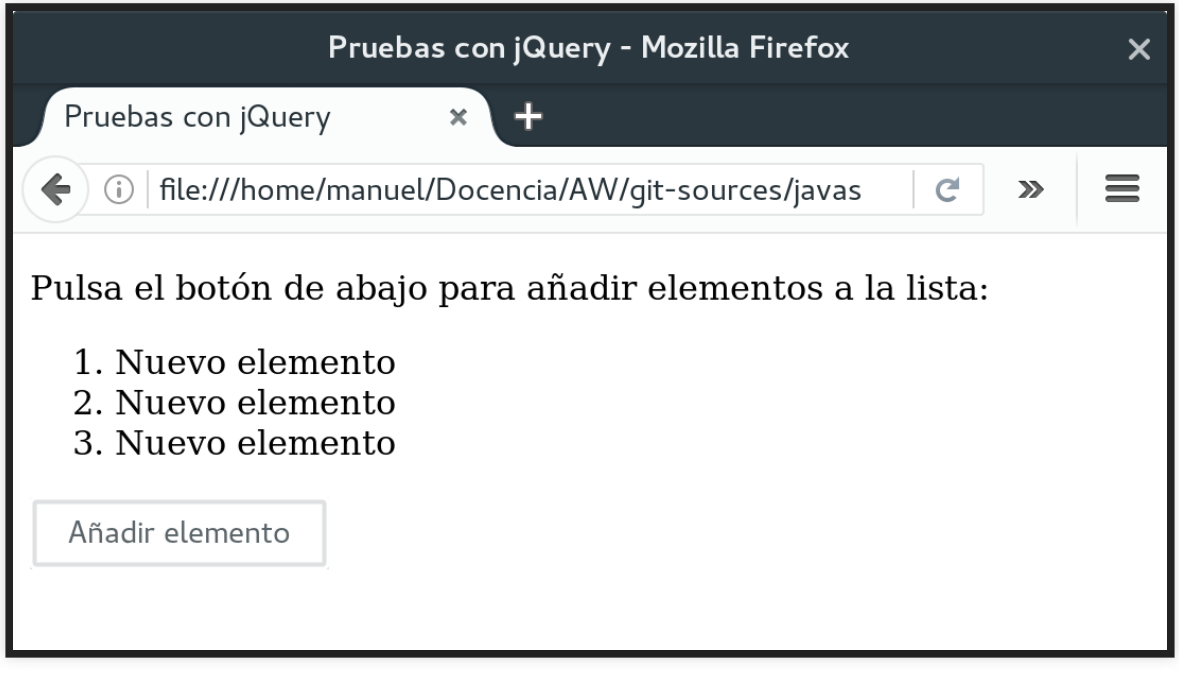
```
$("div").before(elem);
```




```
$("div").after(elem);
```



EJEMPLO




```
<p>Pulsa el botón de abajo para añadir elementos a la lista:</p>
<ol id="listaNumerada">
</ol>
<button id="anyadirElemento">Añadir elemento</button>
```

```
$(() => {
    $("#anyadirElemento").on("click", () => {
        let nuevoElemento = $("<li>Nuevo elemento</li>");
        $("#listaNumerada").append(nuevoElemento);
    });
});
```

OTRAS FUNCIONES

- `.remove()`
  Elimina del documento los objetos seleccionados.
- `.clone()`
  Hace una copia del objeto seleccionado.
- `.wrap(`*`elem`*`)`
  Envuelve los objetos seleccionados en el objeto pasado como parámetro.

Más información:
http://api.jquery.com/category/manipulation/

LAS MÚLTIPLES FACETAS DE $

1. `$(funciónCallback)`
   Acciones a realizar tras inicializar el DOM.
2. `$(elementoDOM)`
   Seleccionar un único elemento.
3. `$(selectorCSS)`
   Seleccionar uno o varios elementos.
4. `$(codigoHTML)`
   Crear nuevos elementos y seleccionarlos.

La lista continuará...

POSICIONAMIENTO Y DIMENSIONES

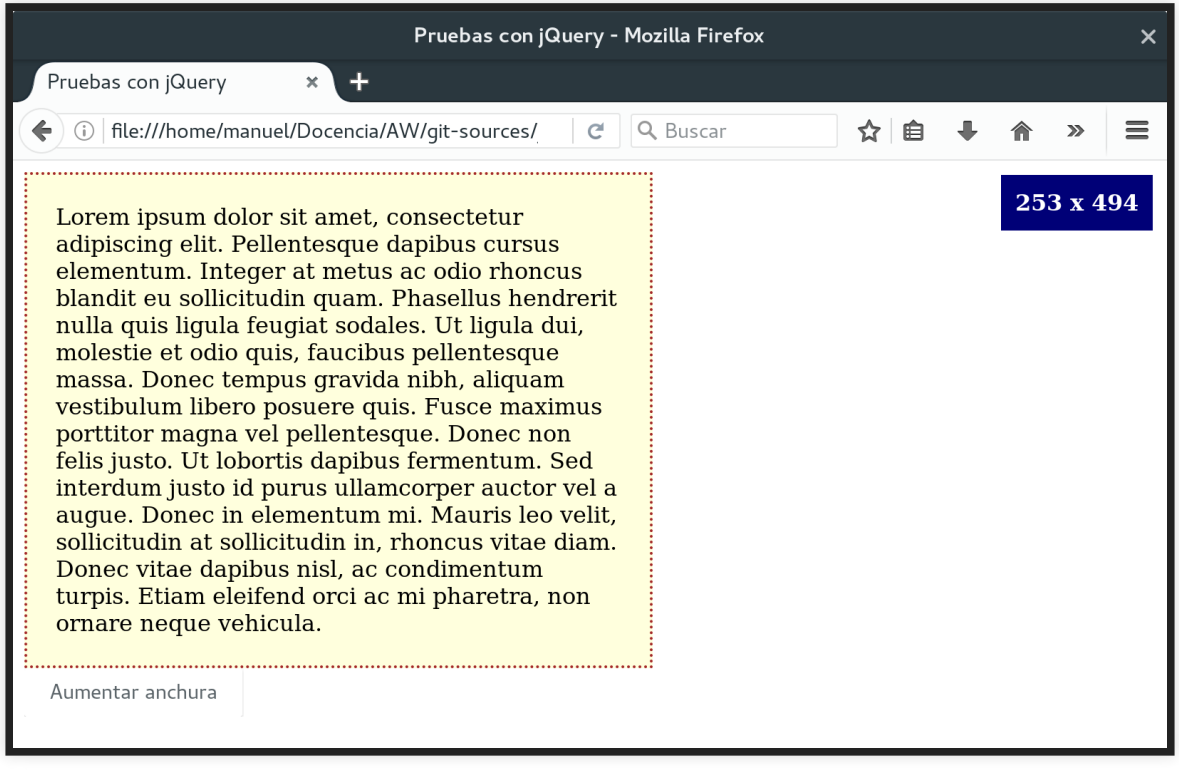
DIMENSIONES DE UN OBJETO

El manejo de las dimensiones mediante las llamadas `.css("width")` y `.css("height")` es bastante complejo, ya que los valores de estas propiedades pueden estar expresados en distintas unidades: pixels, puntos, porcentajes, etc.

Para facilitar esta tarea, existen sendos métodos `.width()` y `height()` que devuelven las dimensiones del objeto seleccionado en píxeles.

- `.width()`, `.height()`
  Obtienen la anchura y la altura del elemento seleccionado.
- `.width(`*`anchura`*`)`, `.height(`*`altura`*`)`
  Modifican la anchura y altura del elemento seleccionado.

EJEMPLO




```
<div class="parrafo">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit...
</div>
<div class="tamaño"></div>
<button id="aumentarAnchura">Aumentar anchura</button>
```

```
// Actualiza la etiqueta de la esquina superior derecha con las
// dimensiones del elemento pasado como parámetro
function actualizarEtiqueta(elem) {
    let ancho = Math.round(elem.width());
    let alto = Math.round(elem.height());
    $("div.tamaño").text(`${ancho} x ${alto}`);
}


$(() => {
    let parrafo = $("div.parrafo");
    actualizarEtiqueta(parrafo);

    // Cuando se pulsa el botón de aumentar anchura...
    $("#aumentarAnchura").on("click", () => {
        // Obtenemos la anchura actual y establecemos la nueva
        let anchoActual = parrafo.width();
        parrafo.width(anchoActual + 20);
        // Actualizamos la etiqueta con la nueva dimensión
        actualizarEtiqueta(parrafo);
    });
});
```

VARIANTES

- `.innerWidth()`, `.innerHeight()`
  Obtienen o cambian la anchura o altura sin tener en cuenta el borde ni los márgenes (pero sí el *padding*).
- `.outerWidth()`, `.outerHeight()`
  Obtienen o cambian la anchura incluyendo *padding*, bordes y márgenes.

POSICIONAMIENTO

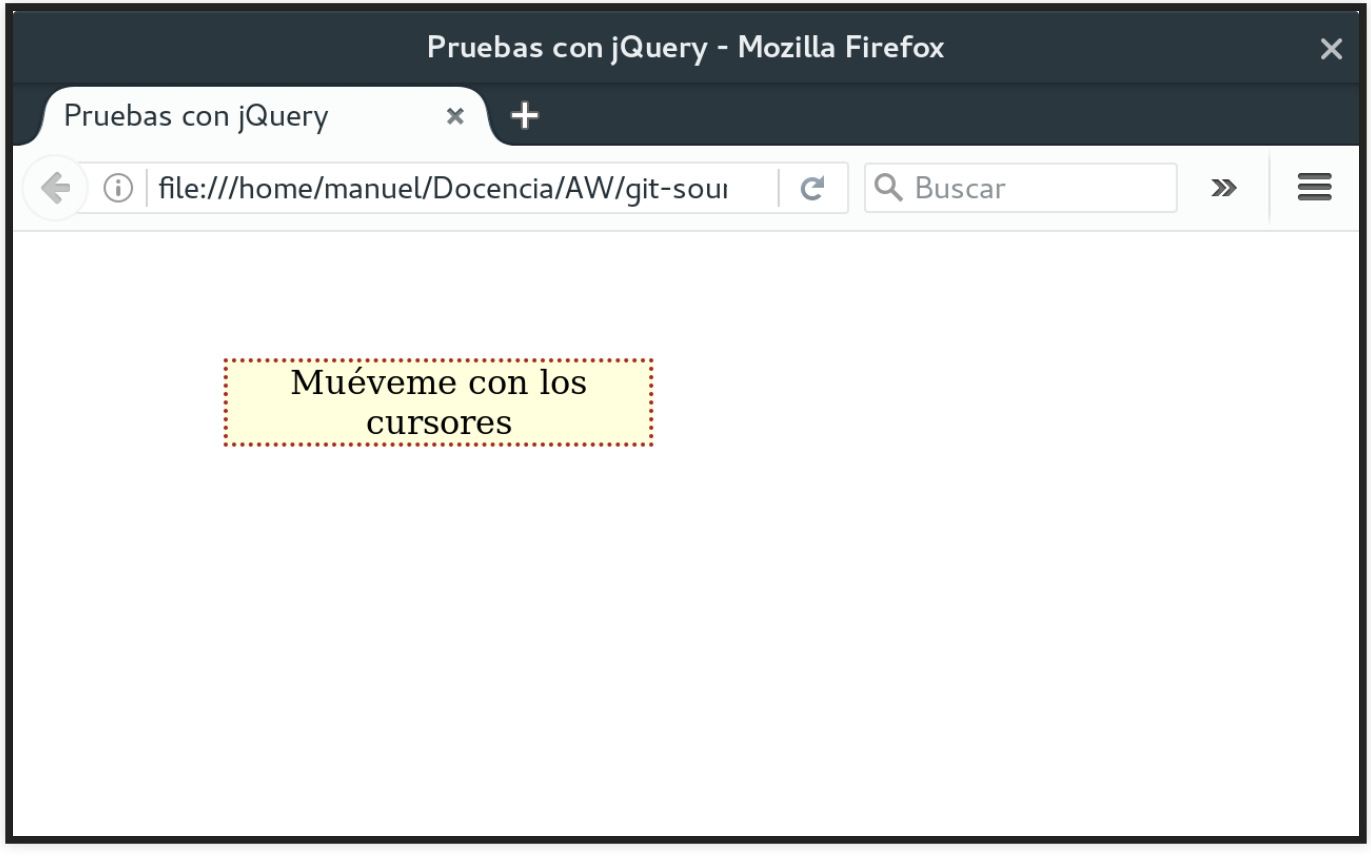
Existen dos métodos para obtener la posición del elemento seleccionado:

- `.offset()`
  Devuelve posición con respecto al documento.
- `.position()`
  Devuelve posición con respecto al elemento padre en el DOM.

Ambos devuelven un objeto con dos atributos: `left` (coordenada x) y `top` (coordenada y).

El método `offset()` permite, además, cambiar la posición de un objeto.

EJEMPLO




```
<div class="parrafo">
     Muéveme con los cursores
</div>
```

```
const IZQUIERDA = 37;
const DERECHA = 39;
const ARRIBA = 38;
const ABAJO = 40;

$(() => {
    let parrafo = $("div.parrafo");

    $("body").on("keydown", (evt) => {
        let incremento = { x: 0, y: 0 };

        switch (evt.which) {
            case IZQUIERDA: incremento.x = -10; break;
            case DERECHA: incremento.x = 10; break;
            // ...
        }

        let current = parrafo.offset();
        parrafo.offset({
            left: current.left + incremento.x,
            top: current.top + incremento.y
        });

        evt.preventDefault();
    });
});
```

Luego veremos para qué sirve esto

EVENTOS

Para asociar un evento a una selección basta con utilizar el método `.on()`, indicando el tipo de evento a capturar y una función manejadora del evento.

```
$("div").on("mousedown", () => {
    // Manejar evento
});
```

EVENTOS DE RATÓN

- `click`, `dblclick`
  Pulsación completa del botón izquierdo (pulsar+liberar).
- `contextmenu`
  Pulsación completa del botón derecho (pulsar+liberar).
- `mousedown`, `mouseup`
  Pulsación y liberación de un botón, respectivamente.
- `mouseenter`, `mouseleave`, `mousemove`
  Entrada/salida del puntero en un área, y su movimiento.

EVENTOS DE TECLADO

- `keydown`, `keyup`
  Pulsación o liberación de una tecla.
- `keypress`
  Pulsación + liberación

EL OBJETO `event`

Según el evento que capturemos, puede ser necesario conocer más información sobre cómo se produce el evento.

Por ejemplo:

- Con `mousemove`, ¿dónde está el puntero del ratón?.
- En `mousedown`, ¿qué botón del ratón se ha pulsado?.
- En `keypress`, ¿qué tecla se ha pulsado?.
- En `keydown`, ¿se ha pulsado también alguna de las teclas auxiliaries (Ctrl, Alt, ...)?
- etc.

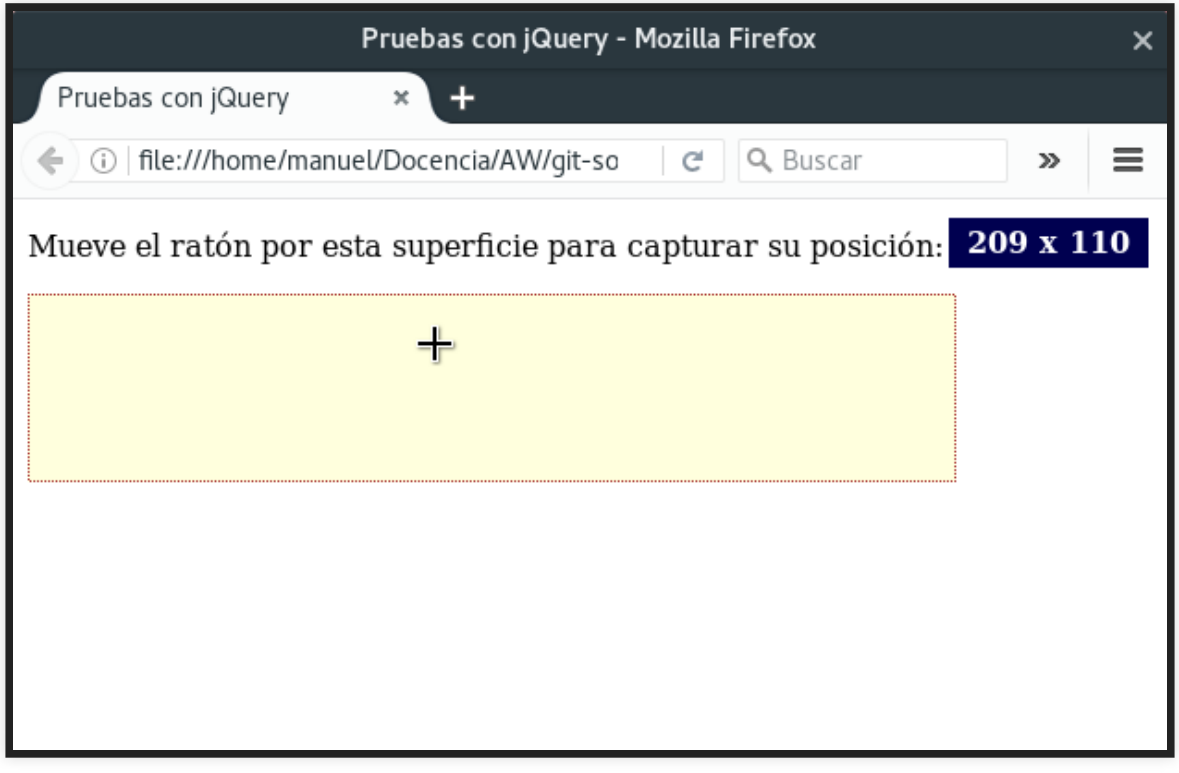
La función manejadora de eventos que recibe el método `.on()` recibe un objeto de la clase `Event` que contiene esta información adicional.

```
$("#elem").on("mousedown", (event) => {
    console.log(`Posición: ${event.pageX}, ${event.pageY}`);
    console.log(`Botón pulsado: ${event.which}`);
});
```

Información del objeto `event`:

- `target`: Elemento del DOM que produjo el evento.
- `timeStamp`: Hora del evento.
- `which`: Botón o tecla pulsado, en caso de eventos de ratón y teclado.
- `ctrlKey`, `metaKey`, `shiftKey`, `altKey`: Indica si se ha pulsado alguna tecla auxiliar.
- `pageX`, `pageY`: coordenadas del ratón, con respecto al documento HTML.

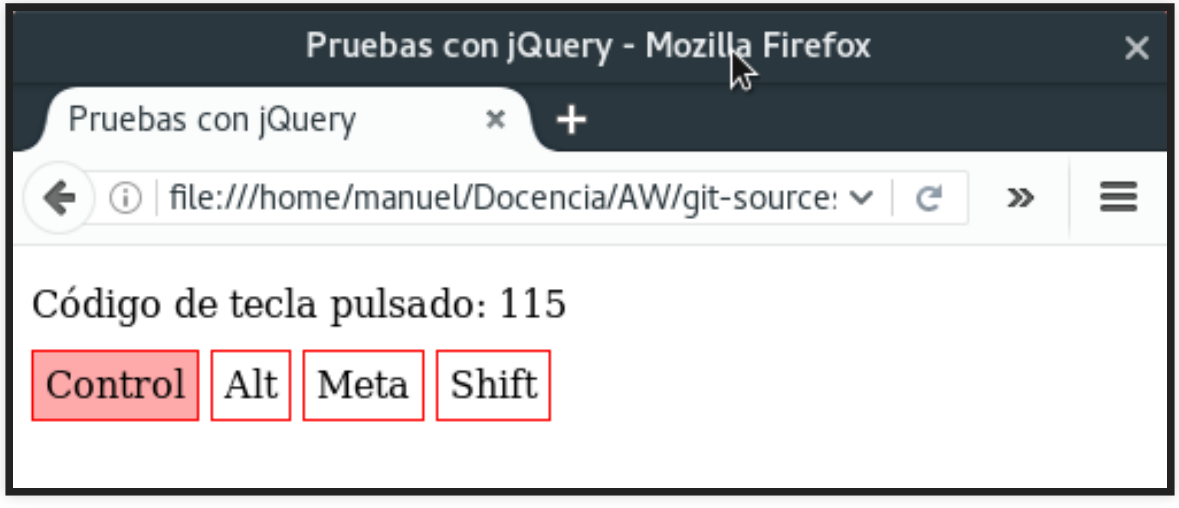
EJEMPLO: MOVIMIENTO DEL RATÓN




```
<p>Mueve el ratón por esta superficie ...</p>
<div id="superficie"></div>
<div id="posicion"></div>
```

```
$(() => {
    $("#superficie").on("mouseenter", () => {
        $("#posicion").show();
    });

    $("#superficie").on("mouseleave", () => {
        $("#posicion").hide();
    });

    $("#superficie").on("mousemove", (event) => {
        $("#posicion").text(
            `${event.pageX} x ${event.pageY}`
        );
    });
});
```

EJEMPLO: PULSACIÓN DE TECLAS




```
<p>Código de tecla pulsado: <span id="codigoTecla"></span></p>
<p>
    <span class="indicador" id="ctrl">Control</span>
    <span class="indicador" id="alt">Alt</span>
    <span class="indicador" id="meta">Meta</span>
    <span class="indicador" id="shift">Shift</span>
</p>
```

```
$(() => {
    $(document).on("keydown", function(event) {
        $(".indicador").removeClass("activo");
        $("#codigoTecla").text(event.which);

        if (event.ctrlKey) {
            $("#ctrl").addClass("activo");
        }

        if (event.metaKey) {
            $("#meta").addClass("activo");
        }

        if (event.altKey) {
            $("#alt").addClass("activo");
        }

        if (event.shiftKey) {
            $("#shift").addClass("activo");
        }
    });
});
```

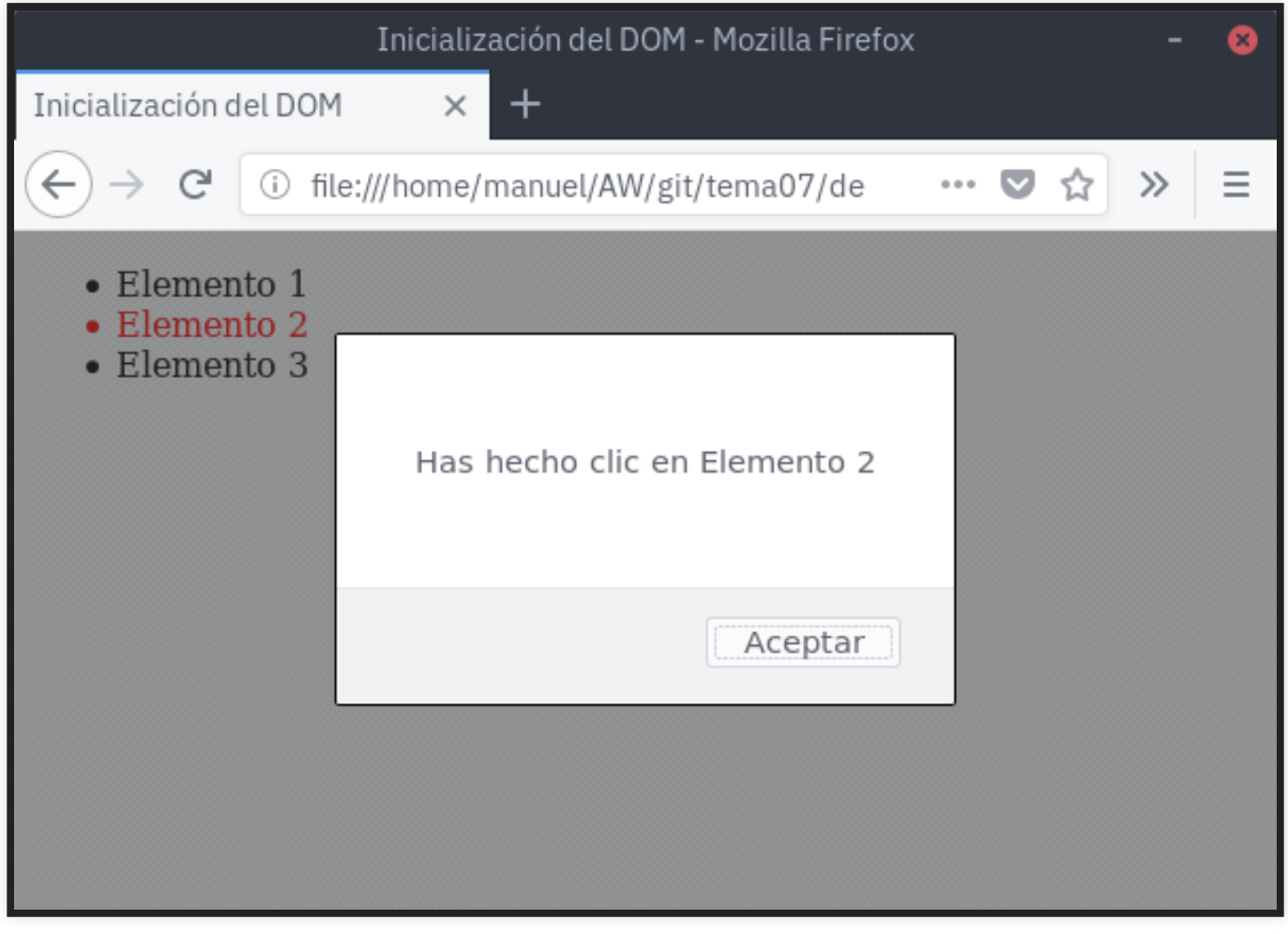
EVENTOS DIRECTOS VS. EVENTOS DELEGADOS

Partimos del siguiente código:

```
<ul id="listaElementos">
    <li>Elemento 1</li>
    <li>Elemento 2</li>
    <li>Elemento 3</li>
</ul>
```

Capturamos el evento de pulsación sobre cada `<li>`:

```
$("#listaElementos li").on("click", (event) => {
    // event.target contiene un elemento del DOM
    // Construimos una selección a partir de él:
    let elementoPulsado = $(event.target);
    // Mostramos mensaje con el contenido del <li>:
    alert(`Has hecho clic en ${elementoPulsado.text()}`);
});
```


Inicialización del DOM - Mozilla Firefox
Inicialización del DOM
file:///home/manuel/AW/git/tema07/de
Elemento 1
Elemento 2
Elemento 3
Has hecho clic en Elemento 2
Aceptar

Supongamos que permitimos añadir nuevos `<li>` de manera dinámica:

```
<button id="anyadir">Añadir nuevo elemento</button>
```

Capturar pulsación del botón:

```
let contador = 3;

$("#anyadir").on("click", () => {
    contador++;
    let newElem = $(`<li>Elemento ${contador}</li>`);
    $("#listaElementos").append(newElem);
});
```


Inicialización del DOM - Mozilla Firefox
Inicialización del DOM
file:///home/manuel/AW/git/tema07/de
Elemento 1
Elemento 2
Elemento 3
Elemento 4
Elemento 5
Elemento 6
Elemento 7
Añadir nuevo elemento

Habíamos capturado la pulsación sobre los <li>:

```
$("#listaElementos li").on("click", (event) => { ... });
```

¿Afecta solamente a los <li> existentes en el momento de ejecutar esta sentencia? ¿O también afecta a los <li> añadidos *a posteriori* de manera dinámica?

Cuando capturamos un evento de la siguiente forma:

`$(selectorCSS).on(evento, callback)`

La función *`callback`* se ejecuta **exclusivamente** en aquellos elementos que se ajusten al *`selectorCSS`* indicado, **en el momento de asociar el evento a la función *`callback`***.

Si, después de ejecutar el método `on()`, añadimos al DOM nuevos elementos que coincidan con el *`selectorCSS`*, estos **no se verán afectados** por la llamada a `on()`.

En nuestro ejemplo, si pulsamos sobre los elementos numerados desde el 4 en adelante, no pasará nada.

Cambiamos la línea

```
$("#listaElementos li").on("click", (event) => { ... });
```

por la siguiente:

```
$("#listaElementos").on("click", "li", (event) => { ... });
```

Esta última captura de eventos afecta a los elementos `<li>` que estén por debajo de `#listaElementos`, tanto los actualmente existentes, como los que se introduzcan después dinámicamente.

Eventos directos

`$(selectorCSS).on(evento, callback)`

Afectan a los elementos que ajusten con `selectorCSS` en el momento de ejecutar esta llamada.

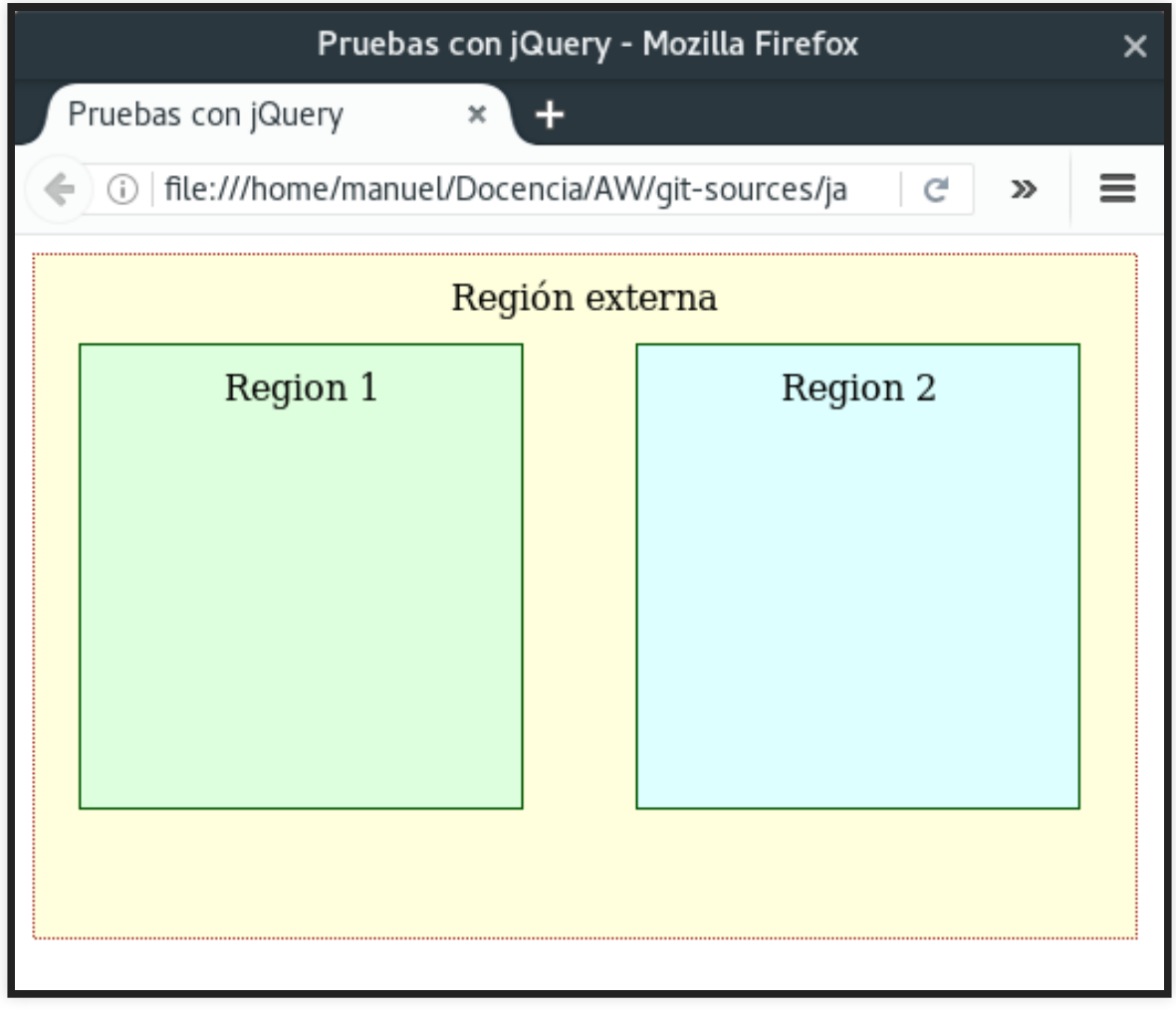
Eventos delegados

`$(selectorCSS1).on(evento, selectorCSS2, callback)`

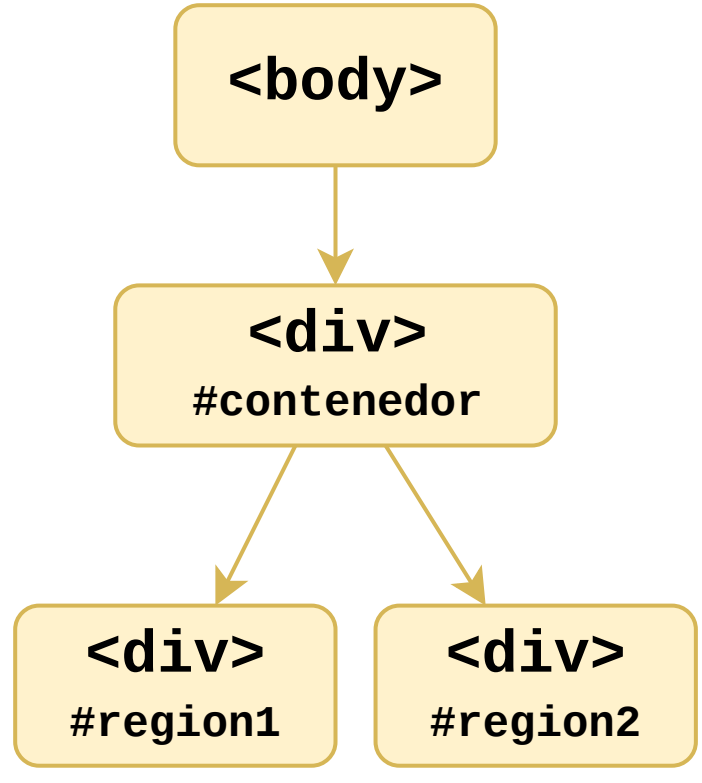
Afectan a los elementos que ajusten con `selectorCSS2` que se encuentren contenidos dentro de alguno de los elementos que ajusten con `selectorCSS1`. Afecta tanto a los elementos **presentes** como a los **futuros**.

PROPAGACIÓN DE EVENTOS

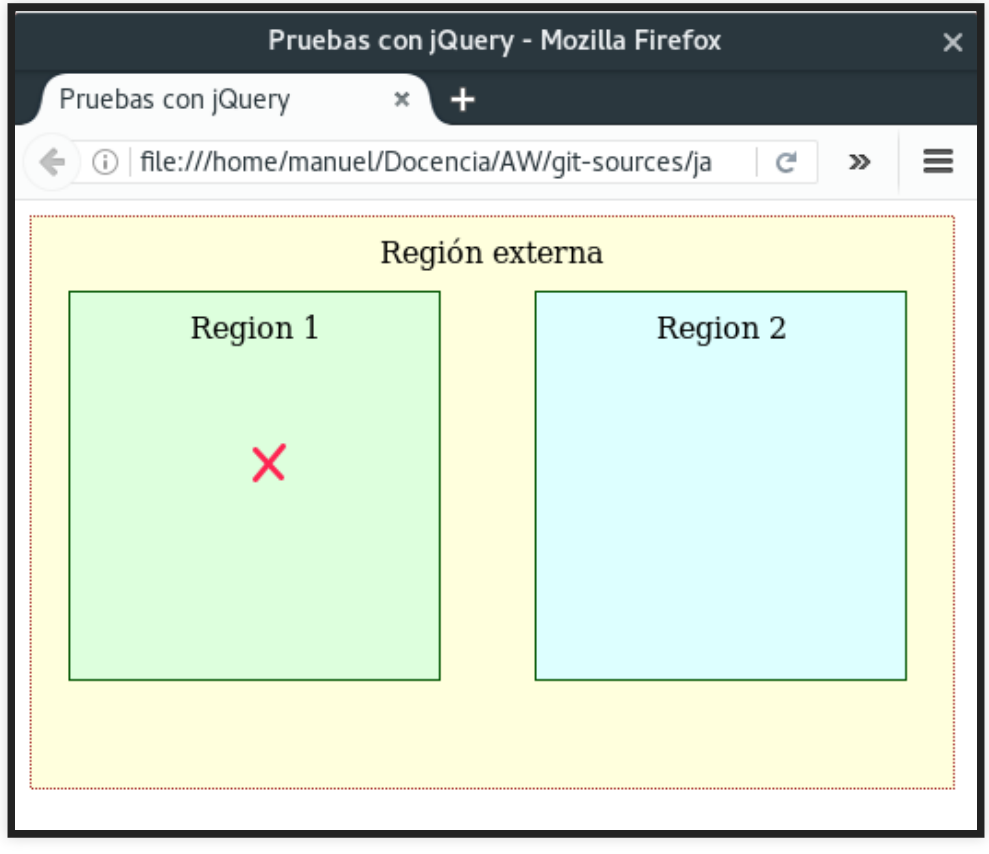
Sabemos que los elementos de HTML pueden estar anidados unos dentro de otros:


Pruebas con jQuery - Mozilla Firefox
Pruebas con jQuery
file:///home/manuel/Docencia/AW/git-sources/ja
Región externa
Region 1
Region 2

```html
<div id="contenedor">
    Región externa
    <div id="region1">
        Region 1
    </div>
    <div id="region2">
        Region 2
    </div>
</div>
```

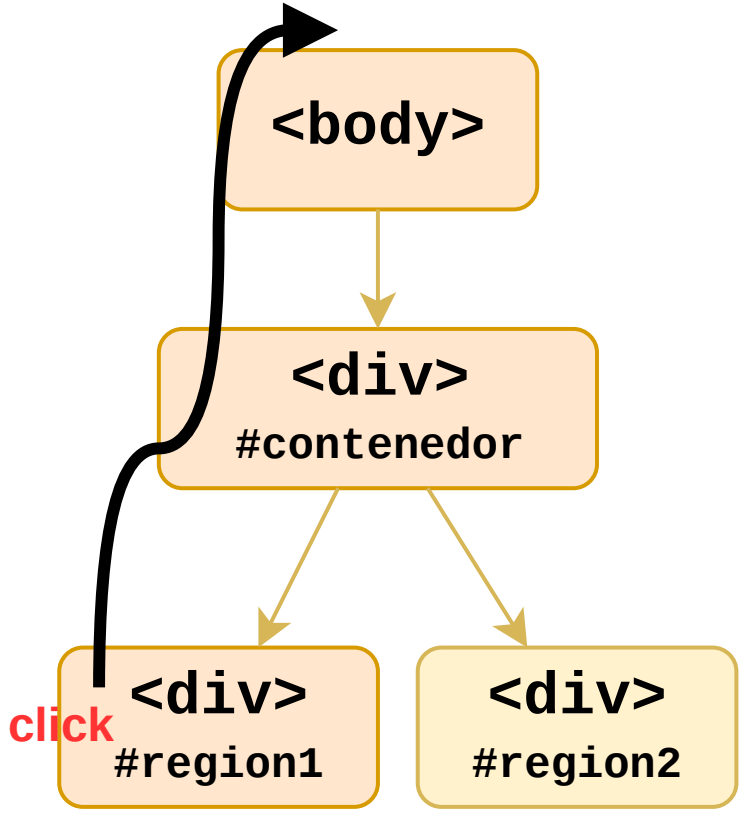


Supongamos que hacemos clic sobre la región 1:



¿Quién recibe el evento `click`? ¿La region 1? ¿La región externa? ¿Ambas?

Los eventos se propagan de manera ascendente en el DOM, desde los elementos «internos» hasta los «externos», y **en ese orden** (*bubble propagation*)



Supongamos el siguiente código Javascript:

```
$("body").on("click", () => {
    console.log("Se ha pulsado en el cuerpo de la página");
});

$("#contenedor").on("click", () => {
    console.log("Se ha pulsado en la región externa");
});

$("#region1").on("click", () => {
    console.log("Se ha pulsado en la región 1");
});

$("#region2").on("click", () => {
    console.log("Se ha pulsado en la región 2");
});
```

Al hacer clic en la región 1:

```
Se ha pulsado en la región 1
Se ha pulsado en la región externa
Se ha pulsado en el cuerpo de la página
```

También reciben el evento

Al hacer clic en la región 2:

```
Se ha pulsado en la región 2
Se ha pulsado en la región externa
Se ha pulsado en el cuerpo de la página
```

También reciben el evento

Es posible impedir la propagación del evento hacia los elementos superiores del árbol DOM.

Para ello hay que llamar al método `.stopPropagation()` el objeto `Event` recibido:

```
...
$("#region1").on("click", (event) => {
    console.log("Se ha pulsado en la región 1");
    event.stopPropagation();
});
...
```

Al hacer clic en la región 1:

```
Se ha pulsado en la región 1
```

El evento no se propaga hacia arriba

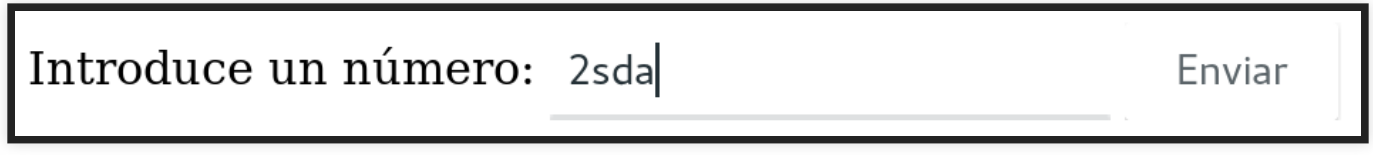
ACCIONES POR DEFECTO

Algunos tipos de eventos llevan implícita una determinada acción por parte del navegador:

- Hacer clic en un enlace: el navegador salta a la URL correspondiente.
- Hacer clic en el botón *Enviar* de un formulario: el navegador realiza una petición GET/POST a la URL indicada en el attributo `action` del formulario.
- Pulsar una combinación de teclas (p. ej. *Ctrl*+*S*), que realiza una determinada acción en el navegador (p. ej. guardar la página actual en el cliente).

Es posible evitar la ejecución de estas acciones utilizando el método `preventDefault()` de la clase `Event`, tras capturar un evento.

EJEMPLO: VALIDACIÓN DE UN FORMULARIO




```
<form action="correcto.html" method="GET" id="formulario">
    Introduce un número:
    <input type="text" name="num">
    <input type="submit" value="Enviar">
</form>
```

Si asociamos un evento al botón `Enviar` del formulario, podemos comprobar que el valor introducido es un número.

```
$("#formulario input[type=submit]").on("click", function() {
    let valor = $("#formulario input[type=text]").prop("value");
    if (isNaN(Number(valor))) {
       alert("No has introducido ningún número!");
    }
});
```

Sin embargo, esta solución no funciona: cuando el usuario introduce una entrada no numérica en el cuadro de texto, además de mostrarse el mensaje con `alert(...)`, se salta a la URL de destino del formulario (`correcto.html`).

Para evitar el envío del formulario en caso de no ser validado, hemos de inhibir la acción por defecto mediante el método `preventDefault()`.

```
$("#formulario input[type=submit]").on("click", function(event) {
    let valor = $("#formulario input[type=text]").prop("value");
    if (isNaN(Number(valor))) {
       alert("No has introducido ningún número!");

       // Inhibir el envío del formulario
       event.preventDefault();
    }
});
```

ADVERTENCIA

Hemos visto un ejemplo de validación de formularios que se ejecuta en el lado del navegador.

Esta validación es beneficiosa, en tanto que se evita el envío de formularios incorrectos al servidor.

No obstante, la validación del lado del cliente **no exime de la validación en el lado del servidor**.

ANIMACIONES Y EFECTOS

EFECTOS

Además de los métodos `show()` y `hide()`, es posible utilizar efectos visuales para mostrar y ocultar elementos.

- `fadeOut([`*duración*`])`, `fadeIn([`*duración*`])`,`fadeTo(`*duración*`, `*opacidad*`)`
  Desaparición / aparición gradual.
- `slideDown([`*duración*`])`, `slideUp([`*duración*`])`
  Desaparición / aparición mediante desplazamiento.

Ejemplo:

```
$("#elem").fadeOut(500); // Tarda medio milisegundo en desaparecer
```

ANIMACIONES PERSONALIZADAS

Método `.animate(*propiedades*, *t*)`

Modifica las propiedades CSS de la selección de manera gradual para que alcancen los valores indicados en el objeto *`propiedades`*. La transición dura *`t`* milisegundos.

```
// Se hace un desplazamiento y un fundido simultáneos durante
// un segundo
$("#elem").css("position", "relative")
          .animate({ top: "100px", opacity: "0" }, 1000);
```

EXTENSIONES DE jQUERY

jQuery es un framework modular, que permite la incorporación de extensiones (*plug-ins*)

- Validación de formularios:
  https://github.com/jzaefferer/jquery-validation
  https://github.com/victorjonsson/jQuery-Form-Validator
- **jQuery UI**: componentes de interfaz de usuario, drag-and-drop, etc.
  http://jqueryui.com/

- **jQueryMobile**: componentes de interfaz de usuario enfocados a dispositivos móviles.
  http://jquerymobile.com/
- **QUnit**: ejecución de tests de unidad.
  http://qunitjs.com/

MÓDULOS JAVASCRIPT EN EL NAVEGADOR

El sistema de módulos de Node no es directamente aplicable al cliente.

Por tanto, no tenemos función `require`, ni módulos que exporten sus funciones mediante `module.exports`.

INCLUSIÓN DIRECTA DE FICHEROS .JS

Es posible separar el código Javascript de una página web en distintos ficheros `.js` e incluirlos mediante las respectivas etiquetas `<script>`:

```
<html>
    <head>
        <script src="modulo1.js"></script>
        <script src="modulo2.js"></script>
        ...
    </head>
    ...
```

El resultado de esto es equivalente a importar un único fichero `.js` en el que se concatenen todas las definiciones de `modulo1.js`, seguidas de las de `modulo2.js`, etc.

¿Qué problema conlleva esto?

Supongamos las siguientes implementaciones:

`modulo1.js`

```
function minombre() {
    ...
}
```

`modulo2.js`

```
...
minombre = 5;
...
```

Existe un **conflicto de nombres** entre los dos módulos. Si un mismo identificador se define en ambos módulos, las definiciones de `modulo2.js` prevalecen sobre las de `modulo1.js`.

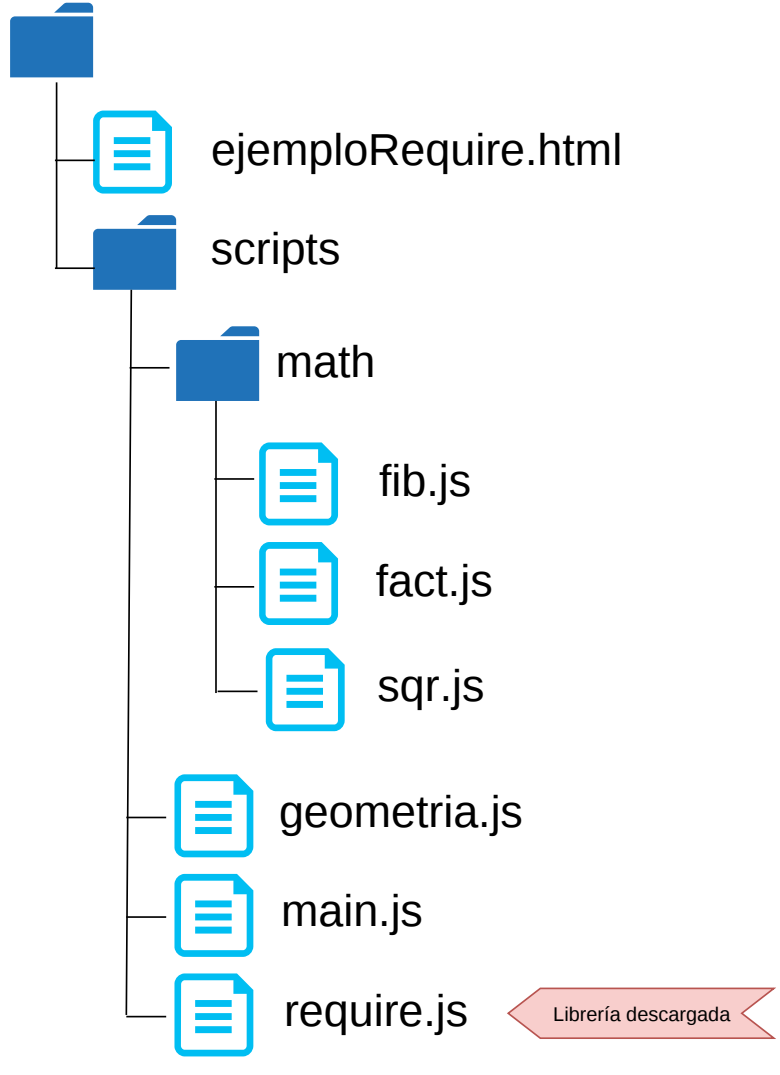
LA LIBRERÍA REQUIRE.JS

http://requirejs.org/

Proporciona un sistema de módulos que evita que los identificadores definidos en distintos módulos interfieran entre sí.

Tiene la misma finalidad que el sistema de módulos de Node, pero está basada en otro modelo: AMD (*Asynchronous Module Definition*).

En los siguientes ejemplos, supondremos la siguiente estructura de directorios:



DEFINICIÓN DE UN MÓDULO

```
"use strict";

define(["dep1", "dep2", ...], (mod1, mod2, ...) => {

    // Definiciones del módulo

    return objeto_o_funcion_exportado;
});
```

La función `define` recibe dos parámetros:

- Una lista con los nombres de las dependencias de los módulos que se está definiendo.
- Una función que recibe los objetos exportados por dichas dependencias y devuelve el valor exportado por el módulo.

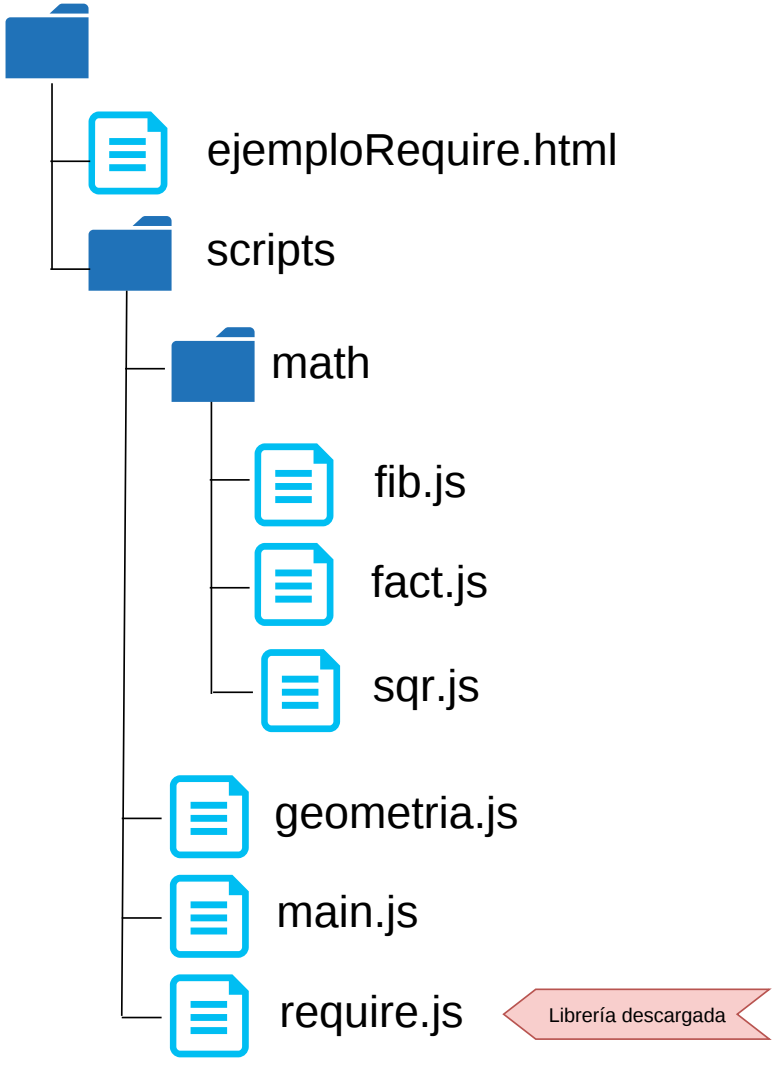
La definición de un módulo mediante Require.js

```
"use strict";

define(["dep1", "dep2", ...], (mod1, mod2, ...) => {

    // Definiciones del módulo
    return objeto_o_funcion_exportado;

});
```

es equivalente al siguiente código en Node:

```
let mod1 = require("dep1");
let mod2 = require("dep2");
...
// Definiciones del módulo
...
module.exports = objeto_o_funcion_exportado;
```

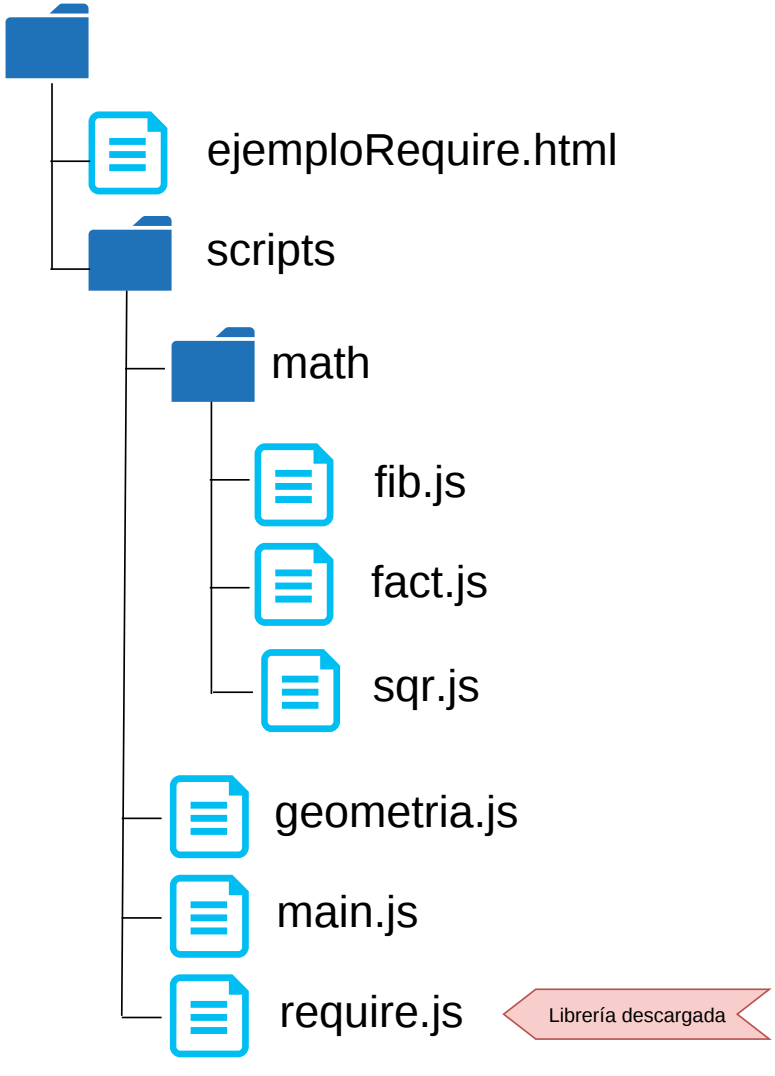
EJEMPLO



fib.js

```
define([], () => {

    var PHI = (1 + Math.sqrt(5)) / 2;

    function fibMasEficiente(n) {
        var p1 = Math.pow(PHI, n);
        var p2 = Math.pow(1 - PHI, n);
        return Math.round((p1 - p2) / Math.sqrt(5));
    }

    function fib(n) {
        console.assert(typeof(n) === "number",
            `fib: ${n} is not a number`);
        return fibMasEficiente(n);
    }

    return fib;

});
```

Se exporta una única función



fact.js

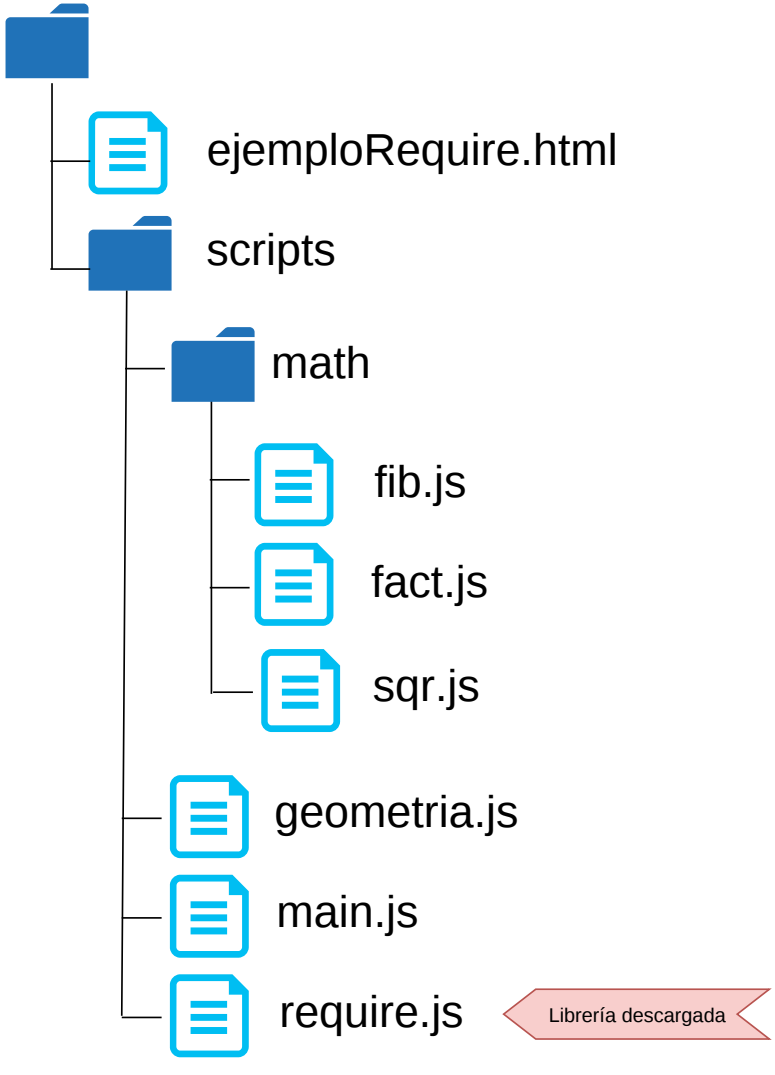
```
define([], () => {
    function fact(n) {
        var result = 1;
        for (var i = 1; i <= n; i++) {
            result = result * i;
        }
        return result;
    }

    return fact;
});
```

Se exporta una función

fact.js

```
define([], () => {

    function sqr(n) {
        return n * n;
    }

    return sqr;
});
```

Se exporta una función



geometria.js

```
define(["math/sqr"], (sqr) => {
```

Este módulo depende de **sqr**

```
    function areaCuadrado(lado) {
        return sqr(lado);
    }

    function areaCirculo(radio) {
        return Math.PI * sqr(radio);
    }

    function perimetroCuadrado(lado) {
        return 4 * lado;
    }

    function perimetroCirculo(radio) {
        return 2 * Math.PI * radio;
    }

    return {
       areaCuadrado: areaCuadrado,
       areaCirculo: areaCirculo,
       perimetroCuadrado: perimetroCuadrado,
       perimetroCirculo: perimetroCirculo
    };

});
```

Se exporta un objeto

main.js

```
define(["math/sqr", "math/fib", "geometria"], (sqr, fib, g) => {

    console.log(`4^2 = ${sqr(4)}`);
    console.log(`fib(6) = ${fib(6)}`);
    console.log(`Area de circulo de radio 10: ${g.areaCirculo(10)}`);

});
```

Para utilizar este módulo en el documento HTML es necesario importar el script `require.js` pasando como atributo `data-main` el nombre del módulo principal:

```
<script src="scripts/require.js" data-main="scripts/main">
</script>
```

Antes de definir el módulo principal suele incluirse una llamada a `requirejs.config` indicando las opciones de configuración:

```
"use strict";

requirejs.config({
    // Directorio en el que se encuentran los scripts
    baseUrl: "scripts",
});

define(["math/sqr", "math/fib", "geometria"], (sqr, fib, g) => {
        ...
});
```

Por defecto, al introducir un módulo `nombreMod` como dependencia, se busca `nombreMod.js` en el directorio indicado por la opción `baseUrl`.

Si se quiere, puede establecerse manualmente una correspondencia entre nombres lógicos de módulos y nombres en el sistema de ficheros. Para ello se utiliza la opción `paths`.

EJEMPLO: USO DE jQUERY

```
requirejs.config({
    baseUrl: "scripts",
    paths: {
        jquery: "../jquery/jquery-3.1.1.min"
    }
});
```

No introducir extension .js

El siguiente módulo cargará el fichero `../jquery/jquery-3.1.1.min.js` como dependencia:

```
define(["jquery"], ($) => {
    $("#miDiv").on("click", () => {
        $("#aviso").show();
    })
});
```

BIBLIOGRAFÍA

- B. Bibeault, Y. Katz, A. De Rosa
  jQuery in Action, 3rd edition
  Manning Publications, 2015
- Documentación de la API de jQuery
  http://api.jquery.com/