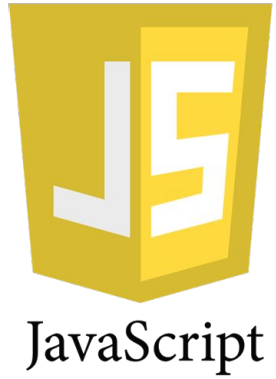
05

JavaScript

Javascript con React y Node JS

fundación esplai
ciudadanía comprometida

Javascript, un poco de historia...



Creado en 10 días en mayo de 1995 por Brendan Eich, trabajador de Netscape, como lenguaje de uso exclusivo en el entorno navegador/cliente.

En 2008 empieza a tomarse en serio...

En 2009 Ryan Dahl "extirpa" el motor de Javascript de Google Chrome (Javascript V8) y lo implementa como aplicación autónoma, abriendo la posibilidad de utilizarlo en el servidor.

Desde 2012 aprox, el crecimiento es imparable...

Modelo Full Stack JS:

Frameworks de cliente tales como React o Angular permiten crear páginas desde el propio navegador.

Los datos se solicitan via AJAX al servidor, donde puede responder una aplicación NodeJS, conectada a una base de datos JSON.

Características de Javascript

Es un lenguaje de programación:

- Sintaxis (palabras clave)
- Variables y estructuras de datos
- Operadores
- Estructuras de control
  - condicionales y bucles

Tipología:

- Orientado a objetos
  - sin clases!
- Débilmente tipado

ECMAScript 5 (2009)

- Modo estricto: 'use strict';
- Soporte JSON

ECMAScript 6 (2015)

- Clases y módulos

Cómo se utiliza javascript

El código Javascript puede estar ubicado en:

- Un archivo con extensión .js, que invocamos desde el documento html con:

  <script src="js/test.js"></script>

- Dentro del mismo html, en un bloque:

  <script> ... </script>

  (puede estar en head o body)

En el código javascript:

- Declaramos "funciones"
  - Objetos, métodos, datos...
- Escribimos sentencias (órdenes) directas.

Los "scripts" se ejecutarán:

- Como respuesta de una acción del usuario
- Como respuesta a un evento

Sintaxis básica

Las sentencias/órdenes javascript deben terminarse con ; (aunque no es imprescindible)

```
var a=100;
```

Podemos escribir y probar código Javascript en la consola del inspector del navegador. La utilizamos también para "debugar" y ver mensajes que se muestran mediante:

```
console.log("Texto del mensaje");
```

Los bloques de código o grupos de sentencias se encierran entre llaves {...}.:

```
function doble(a) {
    return a+a;
}
```

Los elementos del lenguaje son:

- valores
- operadores
- expresiones
- palabras clave
- comentarios.

Valores

Literales: números o cadenas de caracteres. Los números pueden o no tener decimales (separados por punto). Las cadenas entre comillas dobles o simples.
Variables: pueden declararse con la palabra clave (keyword) **var**. No es preciso indicar el tipo de variable en la declaración.
Constantes: variable que no puede modificar su valor una vez asignado, declarada con: **const**

```
var radio = 3.50;
const PI = 3.1415927;
var circulo = radio * radio * PI;
var mensaje_ko = "Clave incorrecta";
var mensaje_ok = 'Clave correcta';
```

Nombres válidos para un "identificador" (variable, nombre de función):

- **No** se pueden utilizar **palabras reservadas** de Javascript
- Solo pueden contener **letras, cifras, "_" o "$"**
- **No pueden empezar por un número**
- Deben empezar por _ $ o una letra
- Las letras en mayúsculas y minúsculas indican identificadores distintos (**distingue may/min**).

"Camel case":

- **nombreUsuario**
- **numeroDeRepeticiones**

Operadores

Aritméticos

- +, -, *, /, %

De asignación

- Símbolo “=”.
- Combinados: +=, -=, *=, /=, %=

De texto/cadena/string

- Concatenación: +
- texto + número = texto

De comparación

- ==, ===
- !=, !==
- <, >, <=, >=
- ? :

Lógicos

- && (AND)
- || (OR)
- ! (NOT)

Funciones

“Unidades básicas” de programación.
Formadas por "n" sentencias
Las podemos crear o "declarar" de varias formas:

```
function saluda() {
        console.log(“Hola mundo!”);
}

saluda = function () {
console.log(“Hola mundo!”) };
```

"Arrow function":

```
saluda = () => console.log("Hola mundo");
```

Una función puede **devolver un valor**, utilizando la orden/comando “return” al final de la misma:

```
function diasSemana() {
        return 7;
}
```

Para acceder al valor devuelto por la función debemos introducir su nombre seguido de los paréntesis ():

```
var dias = diasSemana();
console.log(diasSemana());
```

Importante: el número de () y {} abiertos y cerrados siempre debe coincidir!

Funciones /2

Una función también puede recibir valores, denominados parámetros, que actuarán como variables en su interior (pero no fuera de éste):

```
function suma(a,b) {
        return a+b;
}
console.log( suma(3,4) ); //muestra 7
console.log(a); // error! a no existe
```

Cuando el navegador "lee" un código Javascript detiene la carga de la página html y ejecuta cada sentencia línea por línea, de forma secuencial.

- **Las órdenes directas se ejecutan**
- **Las funciones quedan en memoria,** hasta que alguna orden directa las requiera. Por tanto, el código de la función no se "ejecuta" de forma directa.

Arrays o Listas

Son un tipo especial de variables que pueden contener **múltiples valores**, de distintos tipos. Las declaramos mediante corchetes [ ].

```
var finde = ["sábado", "domingo"];
```

Accedemos a cada valor por su índice, que siempre empieza por 0:

```
console.log(finde[0]); //muestra: sábado
```

Podemos crear un array mezclando distintos tipos de valores:

```
var edades= ["carlos", 30, "ana", 25,
"sandra", 22];
```

Una función puede recibir un número indeterminado de parámetros y convertirlos en un array:

```
function cuenta(...parametros){
  console.log("He recibido "+
        parametros.length + " parámetros.")
}
```

Arrays o Listas /2

Para gestionar los datos de las listas disponemos de varios métodos y atributos, los más interesantes son:

- push(x)
- unshift(x)
- length
- sort()
- indexOf(x)

Objetos (como estructuras de datos)

JavaScript es un lenguaje OOP atípico, porque no permite la definición de clases (hasta ECMA-6). Pero sí permite declarar objetos.

Los podemos utilizar como estructuras de datos "simples" o bien añadiendo métodos o funciones.

Los declaramos mediante { } y pares de datos, y accedemos a sus **propiedades** mediante el operador "punto", tal como vemos a la derecha.

```
var alumno = {
        nombre: "marta",
        edad: 22,
        notas: [8,6,7,8],
        email: "marta@gmail.com" };

console.log(alumno.nombre); // marta
console.log(alumno.edad); // 22
var resultados = alumno.notas;
console.log(resultados[2]); // 7
```

Destacar: el formato como se describen los objetos en Javascript es JSON!

Objetos (con métodos)

Los objetos Javascript pueden contener también métodos, que no son más que funciones definidas "dentro" del objeto. La función circulo() definida a continuación devuelve un objeto de este tipo:

```
function circulo(rad){
     return {
         radio: rad,
         area: function() {return this.radio * this.radio * Math.PI; },
         long: function() {return this.radio * 2 * Math.PI;
          };
     }
var circulo1 = circulo(2);
console.log( circulo1.area() ); // muestra 12.566

var circulo2 = circulo(3);
console.log( circulo2.long() ); // muestra 18.849
```

Estructuras de control

Un script o programa Javascript, como en todos los lenguajes, se ejecuta de forma secuencial (aunque se permiten funciones asíncronas).

Para controlar el flujo de ejecución de un programa, todos los lenguajes disponen de estructuras de control. Son de dos tipos:

- Ejecución condicional
- Bucles

Ejecución condicional:

- Se ejecuta o no un conjunto de sentencias dependiendo del valor de una condición.

Bucles:

- Se repite varias veces un mismo número de órdenes o sentencias, mientras se cumpla una condición predefinida.

Estructuras condicionales

Sentencia if...else

```
if (hoy=="domingo" || hoy=="sábado")
{
     console.log("Fiesta!");
}else{
     console.log("A trabajar…");
}
```

Variante if ... else if ... else if ... else ...

```
if (cond) {...}
     else if (cond2) {...}
     else {...}
```

Sentencia switch (múltiples condiciones)

```
switch (hoy) {
     case "sábado":
          console.log("Fiesta!");
          break;
     case "domingo":
          console.log("Resaca...");
          break;
     default:
          console.log("A trabajar");
          break;
}
```

Try...Catch...

- Utilizamos la estructura **Try...Catch** para control de errores.
- Es un tipo de estructura condicional, que ejecuta un bloque de sentencias si se produce una condición de error.
- Utilizaremos Try...Catch en bloques de código "proclives" a producir un error, por ejemplo al conectar a un servidor remoto.
- Podemos "generar" un error mediante **Throw**

```
function unaFuncion() {
    var mensaje, x;
    mensaje = document.getElementById("p01");
    mensaje.innerHTML = "";
    x = document.getElementById("demo").value;
    try {
        if(x == "") throw "está vacío";
        if(isNaN(x)) throw "no es un número";
        x = Number(x);
        if(x < 5) throw "es menor q 5";
        if(x > 10) throw "es mayor q 10";
    }
    catch(err) {
        mensaje.innerHTML = "El valor " + err;
    }
}
```

Bucles: For

El bucle "for", repite un bloque de código un número predeterminado de veces. Una variable "iterador" toma un valor distinto en cada iteración. Elemplo:

```
var veces=100;
for (var i = 1; i<veces; i=i+1) {
  console.log(i + " No hablaré en clase");
}
```

1 No hablaré en clase
2 No hablaré en clase
3 No hablaré en clase
...

Una variante del bucle for permite recorrer los elementos de un array, iterando sobre cada uno de ellos:

```
var nums=[4,6,3,7];
var total=0;
for (valor in nums){
        total = total + nums[valor];
}
console.log("Suma: "+total);
```

O bien de este otro modo, haciendo uso del método forEach y una arrow function:

```
var nums=[4,6,3,7], total=0;
nums.forEach(n => total+=n);
console.log("Suma: "+total);
```

Bucles: while

El bucle **while** permite repetir un bloque de código mientras sea cierta una condición. En el momento en que deje de ser cierta termina el bucle. Si la condición es falsa al inicio, no se ejecuta el bucle ninguna vez.

Una variante de while es "**do..while**", el cual siempre ejecuta el bucle al menos una vez, puesto que evalúa la condición de continuidad al final.

Otra forma de "salir" de un bucle es mediante la sentencia **break**.

```
var i = 4;

// while
while (i < 10) {
    console.log("Número: "+i);
    i++;
}

//do...while
do {
    console.log("Número: "+i);
    i++;
}
while (i < 10);
```

Código "entendible": Comentarios y nomenclatura

El código no se escribe una vez y se deja para siempre. Hay que mantenerlo: buscar errores, optimizarlo, modificar o añadir requerimientos... Por ello es importante:

- Comentar el código
- Utilizar identificadores adecuados
- **Identar!** (automático en IDE)

¡**Imprescindible cuando trabajamos en equipo!**

Comentar (lo necesario):

- Línea: //
- Bloque multilínea: /* .... */

Identificadores: preferentemente en "camel case", ejemplos:

- buscarClientePorNombre()
- codigoPostal
- fechaEntrada, fechaSalida
- numeroRepeticiones
- etc....

Fechas. Objeto "Date"

JS almacena las fechas en un objeto especial de tipo Date. Podemos crear un objeto de este tipo mediante "new":

- var hoy = new Date();

Si queremos crear un objeto para el día 24 del 12 de 2018:

- var dia = new Date(2018,11,24);

Una vez tenemos el objeto Date, podemos obtener los siguientes datos:

getFullYear()	año AAAA
getMonth()	mes 0..11
getDate()	dia del mes
getHours()	Hora
getMinutes()	Minutos
getSeconds()	Segundos
getTime()	ms 1/1/1970
getDay()	dia semana 0..6
Date.now()	ms ahora

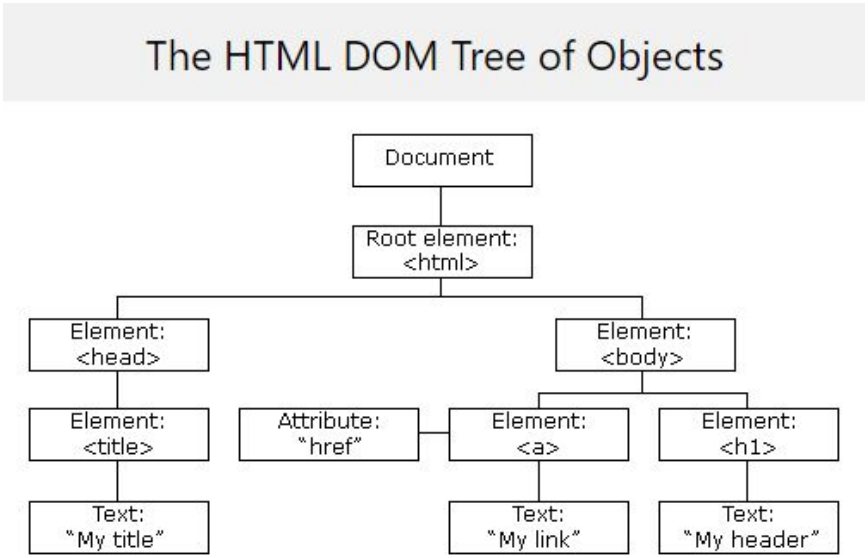
DOM:
Document Object Model

Document Object Model

Cuando el navegador carga una página HTML, crea una representación del mismo en memoria, un **objeto denominado DOM.**

Javascript tiene acceso a leer y modificar este objeto, y de este modo podemos interactuar con los elementos HTML de la página desde nuestras aplicaciones js.

Así, con js **podemos modificar elementos**, características css, **eliminar o añadir nuevos objetos** o **vincular eventos** (funciones) a las acciones del usuario sobre cualquiera de ellos.



Ejemplo: modificación de un <h1>

HTML:

```
<h1 id="elTitulo">...</h1>
```

JS:

```
document.getElementById("elTitulo").innerHTML = "Mi página";
```

jQuery

jQuery

jQuery es una biblioteca de funciones Javascript de código abierto, que podemos usar libremente en nuestras aplicaciones web.

jQuery no hace nada que no se pueda realizar con Javascript, pero simplifica mucho la interacción con el DOM y las funciones Ajax, aspectos esenciales de JS.

Es un estándar de facto, utilizado por la ampliamente por aplicaciones web y otras bibliotecas tales como Bootstrap.

Ejemplo de uso: dado el siguiente elemento html

```
<p id=”miparrafo”>Texto anterior</p>
```

Para cambiar su contenido mediante **javascript**:

```
document.getElementById("miparrafo").
innerHTML = “Texto nuevo”;
```

Para cambiarlo mediante **jQuery**:

```
$(“#miparrafo”).html(“Texto nuevo”);
```

Sintaxis de jQuery

$(selector).acción()

- $ es el prefijo que utilizamos para acceder a las funciones jQuery
- El (selector) realiza la consulta en el dom y devuelve el/los elementos coincidentes. Soporta prácticamente **la misma sintaxis que en CSS**.
- acción() es el método o función javascript que se aplica a los elementos seleccionados

Algunos ejemplos:

$(this).hide()	Oculta el elemento actual
$(“p”).hide()	Oculta todos los elementos <p>
$(“.ocultar”).hide()	Oculta todos los elementos con class=”ocultar”
$(“#este”).hide()	Oculta sólo el elemento con id=”este”

Selectores jQuery

$("*")	Selecciona todos los elementos del DOM
$(this)	Selecciona el elemento actual(*)
$("p.intro")	Selecciona todos los elementos <p> elements con class="intro"
$("p:first")	Selecciona el primer elemento <p> del DOM
$("ul li:first-child")	Selecciona el primer de todos los
$("[href]")	Selecciona todos los elementos con atributo href
$("a[target='_blank']")	Todos los elementos <a> con target="_blank"
$("a[target!='_blank']")	Todos los elementos <a> con target distinto de "_blank"
$("tr:even")	Todas las líneas de tabla (tr) pares
$("tr:odd")	Todas las líneas de tabla impares

Orden de ejecución de los scripts

El navegador ejecuta los scripts javascript/jquery en el orden en que se cargan (<link>) o aparecen en el documento HTML.
Es importante que la biblioteca jQuery se cargue antes de cualquier otro documento JS referenciado o bloque <script>, para garantizar que las funciones estén disponibles cuando se necesitan.
Una página web no se carga de forma inmediata, ni todas sus partes “llegan” a la vez al navegador. Si queremos asegurar que las órdenes javascript se ejecutan cuando la página esté lista debemos utilizar la siguiente función:

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

O bien su forma abreviada y equivalente:

```
$(function(){
        // código Javascript/jQuery...
});
```

Ejemplo: queremos ocultar todas las imágenes de la página que tienen la clase “ocultar”. Si utilizamos la siguiente orden:

```
<script>
        $(“img.ocultar”).hide();
</script>
```

Solo afectará a las imágenes que ya han "llegado" cuando el navegador lee el “script”, si alguna todavía está “en camino” en ese momento no se ocultará y será visible en la página final. Por el contrario, este script:

```
<script>
        $(document).ready(function(){
                $(“img.ocultar”).hide();
        });
</script>
```

Ocultará las imágenes solo cuando verifique que **todo** el contenido haya sido cargado.

Eventos

Los eventos son situaciones que se producen en la página como resultado de la acción del usuario. Por ejemplo, al pasar el ratón por encima de un elemento o al clicar sobre éste.

jQuery nos permite asignar fácilmente una función a un evento, de modo que se ejecuten unas instrucciones de código como respuesta a la acción del usuario.

Estos son los eventos más habituales:

Eventos de mouse	Eventos de teclado	Eventos de formulario	Eventos de documento/ventana
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

Eventos y carga de la página

El siguiente código asigna un evento a todos los párrafos de la página, de modo que cuando cliquemos sobre uno de ellos cambiará el color del texto a "rojo".

```
$("p").click(function(){
        $(this).css({"color":"red"});
});
```

Esta sentencia asocia el evento a todos los párrafos <p> que existan en la página en ese momento. Por ello, deberíamos incluir el código en un bloque $(document).ready() para asegurarnos de que la página esté totalmente cargada.

Aún así, determinadas partes de la página pueden cargarse con posterioridad como resultado de un proceso ajax. Para que estas nuevas partes queden también “cubiertas” por la función, utilizaremos esta otra forma de definir el evento:

```
$(document).on("click", "p", function(){
        $(this).css("color","red");
});
```

En este caso, se asocia el evento "click" al documento, y cuando se produce la acción jQuery busca todos los elementos <p> que existan en ese instante y les aplica el cambio de color.

Efectos "especiales" con jQuery

Una técnica muy habitual en las interfaces de usuario es dejar determinados elementos ocultos, para que aparezcan en el momento en que se desee como resultado de un evento: menús desplegables, pestañas de navegación, etc.

El motivo de dejar que determinados elementos queden ocultos es evitar nuevas llamadas al servidor, así con una sola carga de la página tenemos toda la información necesaria y mediante javascript podemos controlar su visibilidad, mejorando la interactividad de la aplicación.

Para ello jQuery provee varias funciones específicas:

hide() y show()

- Oculta o Muestra un elemento

toggle()

- Conmuta la visibilidad, oculta si está visible o muestra si está oculto.

fade()

- Muestra/oculta de forma progresiva

hide(), show()

Las funciones permiten ocultar (hide) o mostrar (show) elementos del DOM.

Hide equivale a asignar la propiedad css display=none al elemento, por tanto estas funciones son equivalentes:

$(“p.ocultar”).hide();

$(“p.ocultar”).css(“display”,”none”);

Show equivale a asignar la propiedad css display=block, como en el caso anterior serían equivalentes:

$(“p.mostrar”).show();

$(“p.mostrar”).css(“display”,”block”);

Ambas funciones aceptan como parámetros la duración del efecto y una función callback, que será ejecutada al finalizar la transición.

$ (“div.fotos > img”).show(1000);

1000 es el tiempo en milisegundos (1 segundo), también se puede indicar “slow” o “fast”

Si se desea que el método “show” restablezca un elemento como “inline-block”, se puede utilizar el siguiente método:

$("#id54").show(500).css("display", "inline-block");

toggle(), toggleClass()

Toggle () modifica la propiedad display none/block de un elemento, equivaldria a alternar entre hide() y show().

```
$("button").click(function(){
   $("p").toggle();
});
```

addClass()
removeClass()

Una función similar, que utilizaremos incluso de forma más habitual, es toggleClass(). Esta permite añadir o quitar una clase a un elemento html, por ejemplo:

```
$("button").click(function(){
   $("p").toggleClass("texto-rojo");
});
```

En este caso asignaríamos la clase "texto-rojo" a los <p> que no la tengan, y la quitaríamos de aquellos que la tengan.

Método fade()

Los métodos “fade” permiten ocultar o mostrar un elemento de forma progresiva, modificando su transparencia. Son los siguientes:

`$(selector)` `.fadeIn(velocidad,callback)`	Muestra el elemento seleccionado por “selector”, a la velocidad indicada. Una vez mostrado ejecutaría la función callback.
`$(selector)` `.fadeOut(velocidad,callback)`	Similar al anterior, ocultando el elemento
`$(selector)` `.fadeToggle(vel,callback)`	Alterna in/out
`$(selector)` `.fadeTo(vel,trans,callback)`	Modifica la transparencia de un elemento desde su estado actual al indicado por el parámetro “trans”, de forma gradual durante el tiempo indicado por “vel”, ejecutando la función opcional callback al terminar.

Slide, ver http://www.w3schools.com/jquery/jquery_slide.asp
Animate, ver http://www.w3schools.com/jquery/jquery_animate.asp

Encadenar métodos

jQuery permite encadenar métodos fácilmente.
El siguiente ejemplo:

```
$(“#parrafo1”).fadeOut(“fast”)
.css(“color”,”red”).fadeIn(“slow”);
```

Realiza las siguientes funciones:

- Localiza el elemento con id=”parrafo1”
- Lo oculta de forma gradual rápida.
- Lo cambia a color rojo.
- Lo vuelve a mostrar de forma gradual más lentamente.

Sería el equivalente a:

```
$(“#parrafo1”).fadeOut(“fast”);
$(“#parrafo1”).css(“color”,”red”);
$(“#parrafo1”).fadeIn(“slow”);
```

Pero en este segundo caso obligamos a javascript a localizar tres veces el elemento “parrafo1” en la página, mientras que encadenando órdenes tan sólo se localiza una vez.

Los métodos encadenados siempre se ejecutan en el mismo orden en que están escritos.

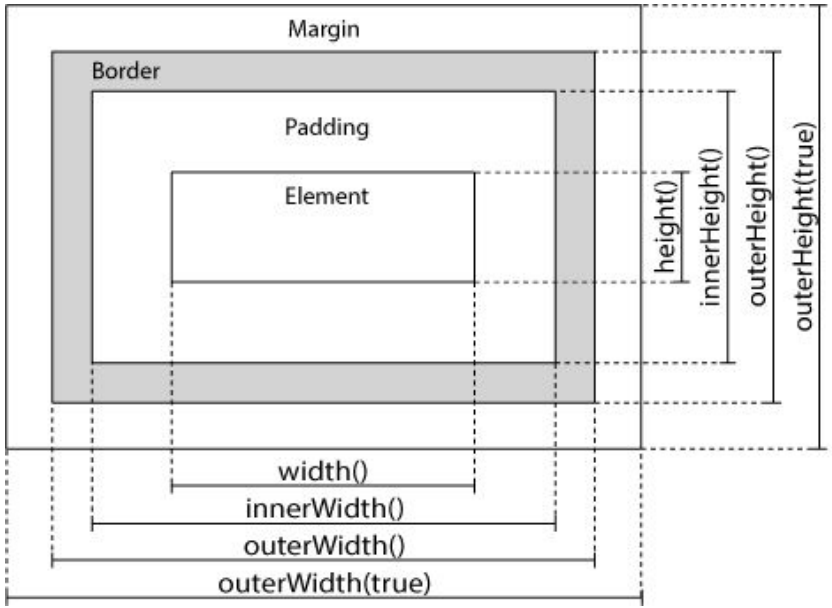
Acceder y modificar el DOM

`text()`	Devuelve o establece el valor de texto de un elemento. Ejemplos: **`var x = $("#parrafo54").text();`** **`$("#parrafo55").text("Nuevo texto para párrafo 55");`**
`html()`	Similar al anterior, pero afecta a todo el contenido del elemento, incluyendo marcas html
`val()`	Devuelve o establece el valor de campos <input>
`attr()`	Permite leer o establecer el valor del atributo de un elemento **`$("#id22").attr("href","http://www.google.com")`**
`css()`	Permite leer o establecer una propiedad css de un elemento **`$("p").css("color","red").css("font-size","15px");`** **`$("p").css({"color": "red", "font-size": "15px" });`**
`addClass()`	Añade una clase a un elemento (o conjunto de elementos) **`$("p").addClass("letra-grande")`**
`removeClass()`	Elimina una clase de un elemento `$("p").removeClass("letra-grande")`
`toggleClass()`	Añade o elimina una clase de un elemento `$("p").toggleClass("parrafo-activo")`
`hasClass("xx")`	Devuelve true o false si el elemento tiene la clase "xx"

Acceder o modificar el tamaño de los elementos

De forma muy similar a los ejemplos anteriores, jQuery permite consultar o modificar el tamaño de los objetos del DOM con los siguientes métodos, que corresponden al gráfico de la derecha:

- width()
- height()
- innerWidth()
- innerHeight()
- outerWidth()
- outerHeight()



Creando contenido con jQuery

append()

append() es un método de **jQuery** que permite añadir contenido al elemento (o elementos) al cual se aplica. Por ejemplo:

**$("ul#lista1").append("
Uno más
");**

...añade nuevos elementos a una lista

El contenido se añade al "interior" del elemento. Por ejemplo, si hacemos append a un , el contenido añadido se sitúa entre el y el :

html antes:

orden jquery:
$("#marca").append("<h1>Hola</h1>");

html después:
<h1>Hola</h1>

Creación de entidades

jQuery permite también crear una nueva entidad de forma muy simple. Una vez creada, utilizamos los métodos ya conocidos para asignar sus atributos, texto,etc.
Ejemplo de creación de un título:

var titulo = $("<h1>").text("Hola");

A continuación lo añadimos a algun elemento, o al propio body:

$("body").append(titulo);

Ejemplo de creación de imagen:

html antes:

orden jquery:
var moto = $("<img>");
moto.**attr**("src", "moto01.jpg");
moto.**addClass**("destacada");
$("#marca").append(moto);

html después:

<img src="moto01.jpg" class="destacada">

Creación de múltiples entidades

A menudo necesitamos crear varias entidades a la vez, a partir de un array de datos. En estos casos, para recorrer el array de datos podemos utilizar el método each() de jQuery. Veamos como llenar mediante jquery una lista vacía:

html antes

```
<ul id="listaItems"></ul>
```

código javascript:

```
var lista=["uno","dos","tres","cuatro"];
$(lista).each(function(index,item){
        $("#listaItems").append("<li>"+item+"</li>
");
});
```

html después:

```
<ul id="listaItems">
<li>uno</li> <li>dos</li>
<li>tres</li> <li>cuatro</li>
</ul>
```

Llenando tablas

Las tablas html <table> son un recurso habitual para mostrar datos procedentes de una lista.

Si conocemos de antemano las columnas que se van a mostrar, podemos crear en html los títulos en el apartado "thead" y llenar mediante javascript el contenido de "**tbody**" a partir de un listado.

Veamos un ejemplo con la tabla siguiente:

html antes:

```
<table width="100%" id="agenda">
        <thead>
        <tr>
        <th width="50%">Nombre</th>
        <th width="50%">Tel</th>
         </tr>
        </thead>
        <tbody></tbody>
</table>
```

Llenando tablas /2

código javascript:

```
var lin1 = $("<tr>");
lin1.append($("<td>").text("Ginger"));
lin1.append($("<td>").text("222333"));

var lin2 = $("<tr>");
lin2.append($("<td>").text("Fred"));
lin2.append($("<td>").text("334455"));

$("table#agenda tbody").append(lin1,lin2);
```

html después (solo parte tbody):

```
...
<tbody>
<tr>
        <td>Ginger</td>
        <td>222333</td>
</tr>
<tr>
        <td>Fred</td>
        <td>334455</td>
</tr>
</tbody>
...
```

¿Para qué crear entidades mediante Javascript/jQuery?

¿No podemos escribir directamente el código html con el contenido?

El motivo de utilizar Javascript es para mejorar la experiencia de usuario en aplicaciones web interactivas.

Si utilizamos páginas creadas en el servidor, cualquier cambio en el contenido, por pequeño que sea, implica recargar de nuevo la página.

Mediante Javascript podemos alterar el contenido de la página "en el lado del cliente", en el mismo navegador. Por tanto, podemos crear páginas interactivas, muy similares a una aplicación de escritorio "tradicional".

Pero qué ocurre cuando necesitamos alguna información que está en el servidor?

JSON

JSON Javascript Object Notation

- JSON es un formato de datos liviano, utilizado principalmente para intercambio de datos
- Al ser un formato de texto plano, es fácil de enviar y recibir
- JSON es "autodescriptivo" y fácil de entender "a simple vista"
- La sintaxis simple de JSON facilita su uso en cualquier lenguaje o plataforma
- Es el lenguaje de almacenamiento de datos preferido en bdd no-sql

JSON Ejemplos

JavaScript tiene una función incorporada para convertir una cadena, escrita en formato JSON, en objetos JavaScript nativos: **JSON.parse()**

De igual modo, existe una función para convertir desde un objeto o variable hacia JSON: **JSON.stringify(Objeto).**

Utilizado en combinación con localStorage nos permite guardar datos en el lado del cliente.

```
//Guardar datos:
myObj = { name:"Anna", edat:21, curs:"Java" };
myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);
```

```
//Leer datos:
text = localStorage.getItem("testJSON");
obj = JSON.parse(text);
document.getElementById("demo").innerHTML =
obj.name;
```

AJAX...

AJAX: el punto de inflexión

AJAX(Asynchronous JavaScript And XML)

Ajax surge en 2005 aunque se apoya en tecnologías desarrolladas años antes.

Mediante Ajax una función Javascript incluida en una página web puede establecer una conexión con un servidor, y esperar su respuesta.

La comunicación es asíncrona, lo cual significa que el navegador no se detiene para esperar la respuesta. De esta forma no se interfiere en la experiencia de uso de la página/aplicación.

El formato de datos habitual en las comunicaciones Ajax es JSON. También XML, aunque cada vez menos.

La tecnología Ajax ha supuesto un revulsivo en las aplicaciones web, mejorando notablemente su rango de aplicación. Prácticamente todas las webs modernas utilizan Ajax, algunos ejemplos:

- Google maps
- Google drive, docs, sheets...
- Wallapop (carga progresiva de contenidos)

Ajax mediante jQuery

A la derecha vemos un ejemplo de petición Ajax mediante jQuery.

Al pulsar un botón se dispara la petición. El método $.getJSON es una versión simplificada de $.ajax.

El método toma como argumentos la url del servidor y una función callback que recibirá el resultado y se ejecutará cuando este llegue.

En este caso, la función muestra en consola un listado de nombres.

```
$(document).on("click", "button#ajax", function ()
{
      url="http://apiremota.com/clientes/todos";

      $.getJSON(url, function (data) {
            data.listado.forEach(function(cliente){
                  console.log(cliente.nombre);
            });
      });
});
```

Pruebas Ajax en modo local

Para probar los métodos Ajax de jQuery necesitaríamos conectar a un servidor remoto, o a una máquina virtual, donde una aplicación de servidor (php, java...) nos enviase los datos.

Alternativamente, podemos efectuar una conexión directa con un archivo de datos local en formato JSON, que simulará la respuesta de un servidor remoto.

Problema: Google Chrome no lo permite*... pero Edge o Firefox sí.

(requiere ser iniciado con un parámetro especial)

Esta petición ajax, dirigida a un archivo local, funcionaría correctamente con Edge, por ejemplo:

```
$.getJSON("ajax/clientes.json",
        function (data) {
                // codigo
        });
});
```

Conexión Ajax en remoto

La siguiente url facilita un JSON que podemos utilizar para pruebas. Se trata del listado de estaciones de bicing en barcelona:

https://api.citybik.es/v2/networks/bicing

Ejemplo de acceso:

```
$.getJSON(
"https://api.citybik.es/v2/networks/bicing",
        function (data) {
            var estacions = data.network.stations;
            console.log(estacions);
        });
```

LISTAS!

fundación esplai
ciudadanía comprometida

Manejo de listas con javascript /1

Para gestionar datos de forma eficiente con Javascript es muy útil conocer los siguientes métodos relacionados con el manejo de listas:

- filter y find
- map
- forEach
- sort

Todos los métodos pertenecen al objeto Array. Normalmente les pasaremos una **arrow function**, pero también puede ser una función "normal".

FILTER: Aplica una condición sobre cada elemento de una lista, devolviendo una lista nueva con los elementos de la primera que cumplen la condición. Ej:

```
var lista=[1,2,3,4,5,6];
var pares = lista.filter(el => el%2===0);
console.log(pares); // [2,4,6]
```

FIND: Similar a la anterior, pero devuelve un solo elemento de la lista, el primero que cumple la condición. Ej:

```
var primerPar = lista.find(el => el%2===0);
console.log(primerPar); // 2
```

Manejo de listas con javascript /2

MAP: Aplica una función a cada elemento de una lista, devolviendo una lista nueva con los resultados. Ej:

```
var lista=[1,2,3,4,5,6];
var dobles= lista.map(el => el*2);

console.log(dobles); // [2,4,6,8,10,12]
```

FOREACH: Aplica una función a cada elemento de una lista, **sin devolver nada**. Ej:

```
var lista=["barcelona", "girona", "mataró"];
lista.foreach( (el, indice) => {
        console.log("elemento: " +el);
        console.log("indice: "+indice);
});
```

Generaría la siguiente salida en consola:

elemento: barcelona
índice: 0
elemento: girona
índice: 1
elemento: mataró
índice: 3

Manejo de listas con javascript /3

SORT: Ordena una lista. Atención: modifica la lista directamente, no crea una nueva!

```
var lista=[4,2,5,1,3,6];
console.log(lista); // [4,2,5,1,3,6]

lista.sort();
console.log(lista); // [1,2,3,4,5,6]
```

Array.sort funciona directamente para elementos numéricos o strings, pero no para objetos. En este caso deberemos crear una función que compare dos objetos a,b y devuelva 1,0,-1 segun a>b, a=b, a<b respectivamente.

Por ejemplo, tenemos una lista de objetos del tipo: [{ nombre: xx, edad: yy], ...] y queremos ordenarla por edad, debemos crear una función que compare objetos:

```
function comparaEdad(a,b) {
        if (a.edad===b.edad) return 0;
        if (a.edad>b.edad) return 1;
        return -1;
}
```

Y finalmente podemos ordenar la lista "personas" utilizando esta función:

```
personas.sort(comparaEdad);
```

Objecto "Set"

Set es un tipo de objeto especial en javascript que permite crear una lista de elementos únicos.

Para añadir elementos al Set utilizamos la función set.add(x) donde x es el valor/elemento a añadir.

Si este elemento ya existe no se añadirá.

En cualquier momento podemos obtener un array a partir del con Array.from(Z) donde Z es la variable que apunta al Set.

Para crear un Set:

```
let Z = new Set();
```

Para crear un Set a partir de un array:

```
let Z = new Set([1,2,3]);
```

Métodos y propiedades:

- add()
- has()
- delete()
- length

Obtener un array a partir de un Set:

- let A = Array.from(Z)
- let A = [...Z]

Validación de formularios con Javascript (jQuery)

fundación esplai
ciudadanía comprometida

Validación de formularios

Una de las tareas más habituales de javascript en una aplicación web es la validación de formularios previa a su envío.

Esto se hace capturando el evento "submit" del formulario. Mediante jQuery podemos hacerlo de la siguiente forma:

```
$(document).on("submit", "#id-formulario",
function(evento){
        evento.preventDefault();
        // código...
        $("#id-formulario").submit();
});
```

OTRAS APIS...
Google Maps

Utilización de la API de Google Maps

Una API no sólo puede facilitar datos en formato JSON o XML, también nos permite acceder a funciones que potencian nuestras aplicaciones. Es el caso de Google Maps.

Para utilizarla deberemos obtener una API KEY que google nos facilita en la siguiente url:

https://developers.google.com/maps/documentation/javascript/get-api-key

Una vez tengamos la API deberemos introducir el siguiente script en nuestro html:

```
<script
src="https://maps.googleapis.com/maps/api/js?key=
NUESTRA_API_KEY&callback=initMap"
  type="text/javascript"></script>
```

A partir de entonces podremos acceder a los métodos de la API desde Javascript.

No es necesario usar jQuery, pero tampoco hay problema en que esté cargado.

API Google Maps /2

El primer paso será inicializar el "plugin" maps:

```
var map; //variable global
function initMap() {
    var center = {lat: 41.408672, lng: 2.174464};
    map = new google.maps.Map(
        document.getElementById('map'), {zoom: 12,
center: center});
}
```

La función anterior recibe las coordenadas lat,lng donde queremos centrar el mapa y el id del div donde lo aparecerá, y que deberemos tener en nuestro html:

```
<div id="map"></div>
```

API Google Maps /3

Ubicando estaciones:

```
var coords = [...]; // array de objetos lat,long creado a partir de estaciones
var markers=[];

function addMarkersToMap(){
   coords.forEach(function(element){
      var latlng = new google.maps.LatLng(element.lat,element.long);
      var marker = new google.maps.Marker({
         position: latlng,
         map: map
      });
      markers.push(marker);
   });
}
```