



UNIVERSIDAD
DE GRANADA



Tecnologías Web

Grado en Ingeniería Informática

Tema 4 –Programación en el lado del cliente

Client-side JavaScript

© Javier Martínez Baena

jbaena@ugr.es

Departamento de Ciencias de la

Computación e Inteligencia Artificial

<http://decsai.ugr.es>



UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

1. El lenguaje JavaScript

2. Client-side JavaScript

1. Conceptos básicos

2. El objeto window

3. DOM

4. Eventos

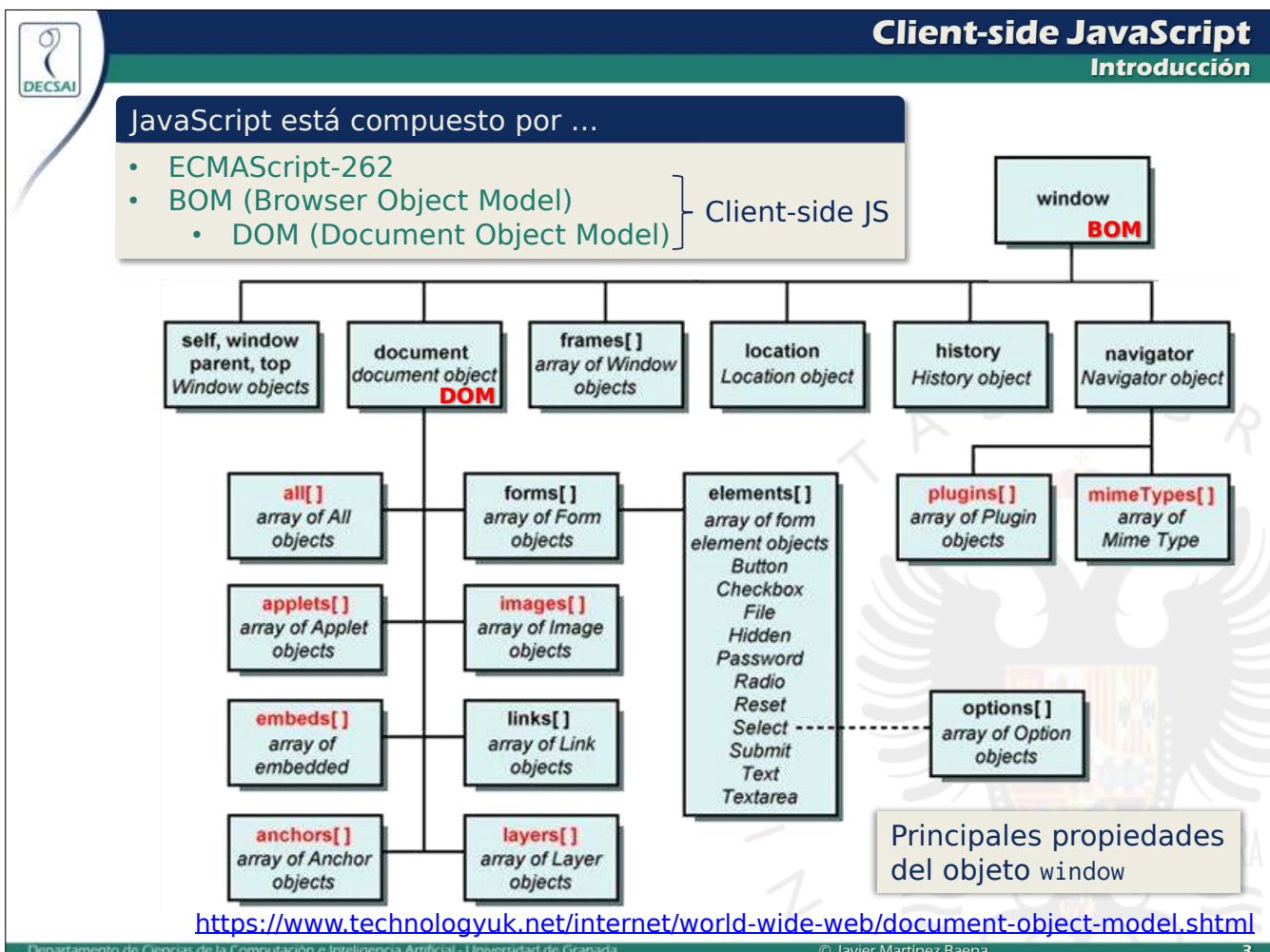
5. Validación de formularios

6. AJAX



Client-side JavaScript

Introducción



Client-side JavaScript

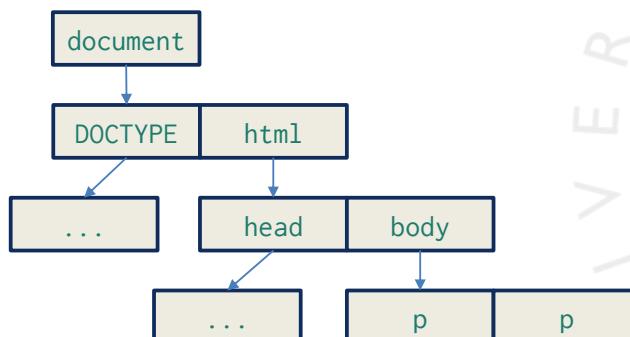
Introducción

Ejemplo: acceso directo a nodos del DOM

Acceso a los elementos del documento HTML desde el DOM

```

<!DOCTYPE html> <html> <head> <meta charset="utf-8">
<title>Ejemplo DOM</title>
<script> onload = function() {
  document.childNodes[1].childNodes[1].childNodes[1].textContent =
    document.childNodes[1].childNodes[1].childNodes[0].textContent;
}
</script>
</head><body><p id="parrafo1">Primer párrafo</p><p id="parrafo2">Segundo párrafo</p>
</body> </html>
  
```



Primer párrafo

Primer párrafo

Ejemplo: acceso al DOM a través de API

Modificación de la página en respuesta a un evento

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
    <script>
      function cambiame() {
        var elem = document.getElementById("parrafo");
        elem.innerHTML = "Texto modificado";
      }
    </script>
  </head>
  <body>
    <p id="parrafo" onclick="cambiame()">Texto de prueba</p>
  </body>
</html>
```

html

head

body

meta

title

p

Modificar elemento

Acceso a un elemento del documento a través del DOM

Programando el evento para ejecutar el código JS

Ejemplo: modificando la estructura del DOM

Modificación de la página

```
<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>Ejemplo DOM</title>
<script> onload = function() {
  document.childNodes[1].childNodes[1].appendChild(
    document.childNodes[1].childNodes[1].childNodes[0].cloneNode(true));
} </script>
</head><body><p>Primer párrafo</p><p>Segundo párrafo</p></body></html>
```

document

DOCTYPE

html

...

head

body

...

p

p

p

Primer párrafo

Segundo párrafo

Primer párrafo



JS embebido en la página web

```
<script>
...
</script>
```

JS en fichero externo

```
<script src="fichero.js"></script>
```

JS en atributos de elementos HTML para programar eventos

```
<p id="parrafo" onclick="... código JS ... ">Texto de prueba</p>
```

El código JS incluido en elementos de tipo `<script>` se ejecuta una vez
Para que haya interactividad: **programar eventos**

Manejador de eventos (event handler): es una función JS registrada por el navegador y que se ejecuta cuando ocurre un determinado evento

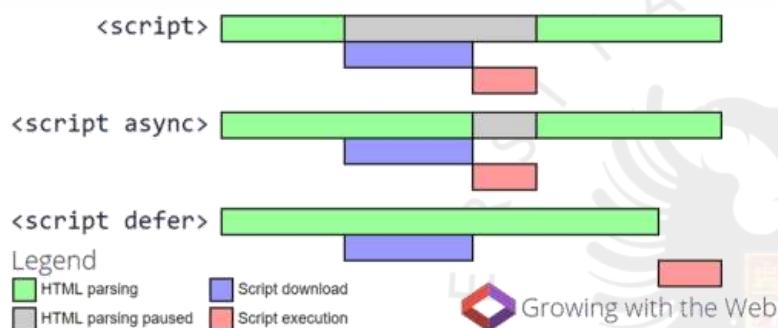


- Cuidado: acoplamiento mínimo HTML-JS
- Se desaconseja programar eventos dentro del documento HTML mediante atributos



La carga y ejecución del código JS (desde fichero)

- **Por defecto es síncrona**: hasta que el fichero JS no se ha cargado por completo no se sigue procesando el documento HTML
- **Diferida** (defer): se pospone la ejecución del JS hasta que el documento HTML ha sido cargado por completo y está listo para usarse. Si hay varios se ejecutan en el orden en que han sido incluidos.
- **Asíncrona** (async): comienza la ejecución de JS tan pronto como pueda sin bloquear la carga del documento HTML

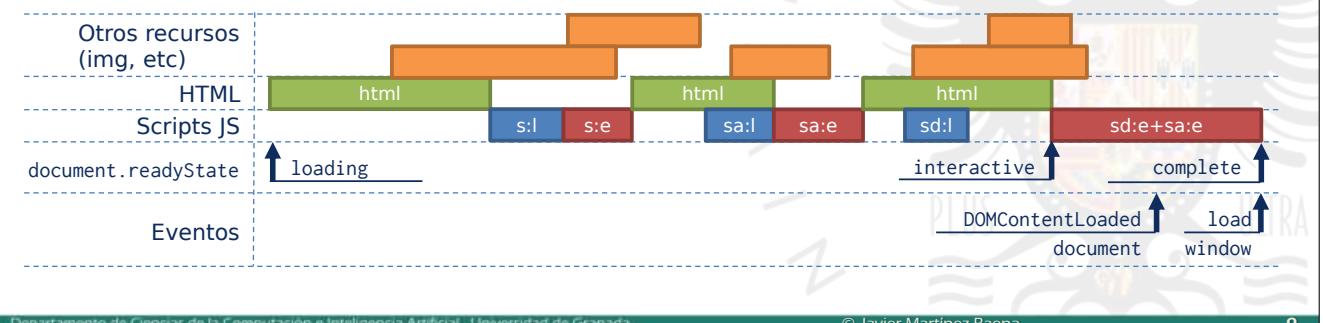


- No tienen porqué estar soportados por el navegador
- Se pueden considerar recomendaciones para optimizar la carga
- El código JS debería funcionar independientemente de este atributo

Client-side JavaScript

Time-line de carga y ejecución de scripts JavaScript

1. El navegador crea el objeto document vacío.
Comienza a analizar el documento HTML (parse) y generando document.
 2. Cuando se encuentra un elemento <script> detiene el parsing del HTML, lo carga y lo ejecuta.
 3. Cuando se encuentra un elemento <script async> comienza su descarga en paralelo con el parsing del HTML. Una vez descargado detiene el parsing del HTML y ejecuta el script.
 4. Cuando se encuentra un elemento <script defer> comienza su descarga en paralelo con el parsing del HTML. No lo ejecutará hasta después de acabar el parsing del HTML.
 5. Al finalizar el parsing del HTML ejecuta los scripts diferidos. Puede que se ejecuten también algunos asíncronos que falten por descargar



Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada

© Javier Martínez Baena

9

Client-side JavaScript

El evento upload

El evento `onload` se genera cuando la página se ha cargado por completo

```
console.log("Comienza script");
function f() {
    console.log("Ejecutando función ...")
}
window.onload=f;
console.log("Acaba script");
```

Comienza script
Acaba script
Ejecutando función ...

```
<script>
function cambiame() {
    var elem = document.getElementById("parrago");
    elem.innerHTML = "Texto modificado";
    console.log("Elemento modificado");
}
cambiame();
</script>
```

✗ ▾ TypeError: elem is null [Saber ...3_cambiame2.html:9:9
más]
cambi... ...transparencias/src/tw03_cambiame2.html:9:9
<anón... ...transparencias/src/tw03_cambiame2.html:12:7

```
<body>
<p id="parrafo" onclick="cambiame()">Texto de prueba</p>
</body>
```

```
window.onload = cambiame;
```


UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

- 1. El lenguaje JavaScript
- 2. Client-side JavaScript
 - 1. Conceptos básicos
 - 2. El objeto window
 - 3. DOM
 - 4. Eventos
 - 5. Validación de formularios
 - 6. AJAX

»»»

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL - UNIVERSIDAD DE GRANADA

© Javier Martínez Baena



BOM: El objeto window

El objeto window

Es el objeto global de JS client-side que permite:

- Acceder a algunos datos del navegador (URL, history, ...)
- Mostrar algunas cajas de diálogo desde la página web
- Abrir y cerrar ventanas de navegación
- Establecer temporizadores
- ...

```

graph TD
    window["window"] --- self["self, window"]
    window["window"] --- parent["parent"]
    window["window"] --- top["top"]
    window["window"] --- document["document"]
    window["window"] --- frames["frames[]"]
    window["window"] --- location["location"]
    window["window"] --- history["history"]
    window["window"] --- navigator["navigator"]
    
```

Departmento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada

© Javier Martínez Baena



BOM: El objeto window

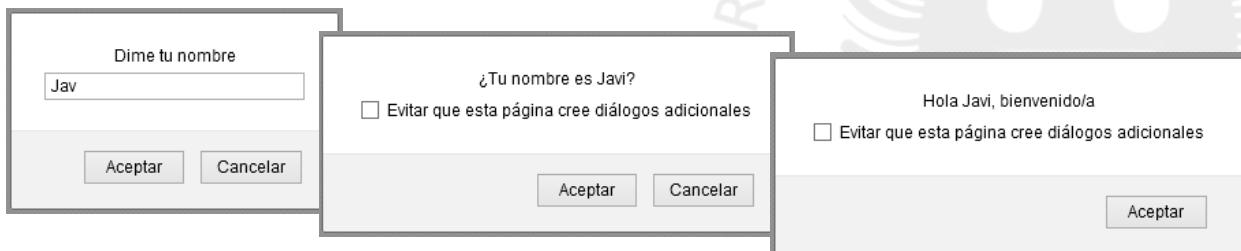
Cajas de diálogo

Cajas de diálogo para comunicación básica con el usuario

- `alert()`. Muestra un mensaje y espera a que el usuario cierre el diálogo
- `confirm()`. Muestra un mensaje y espera a que el usuario pulse “Aceptar” o “Cancelar”
- `prompt()`. Muestra un mensaje y espera a que el usuario escriba un texto. Devuelve el texto escrito

Los tres son diálogos modales (bloquean la ejecución hasta cerrarlos)

```
var nombre = prompt("Dime tu nombre");
var correcto = confirm("¿Tu nombre es " + nombre + "?");
if (correcto)
    alert("Hola " + nombre + ", bienvenido/a")
else
    alert("Pues es lo que me acabas de decir ...")
```



`window.prompt` es lo mismo que `prompt` (por ser `window` el objeto global)

BOM: El objeto window

URL del navegador



Propiedades de `window.location`

Información sobre la ubicación del documento actual

`href`: URL completa de la página

`protocol`: protocolo de la URL

`hostname`: nombre del host

`port`: puerto

`pathname`: ruta de la página (sin el hostname)

`host`: nombre del host y puerto

`origin`: protocolo, nombre del host y puerto

`hash`: Parte de URL desde el carácter #

`search`: Parte de la URL desde el carácter ?

Métodos de `window.location`

`assign()`: Carga una nueva URL

`reload()`: Carga de nuevo la misma página

`replace()`: Carga una nueva URL sustituyendo la actual

(es decir, sin almacenar la actual en el historial del navegador)

BOM: El objeto window
URL del navegador

location:	http://localhost/tw/js/dom2.html#punto
assign:	assign()
hash:	#punto
host:	localhost
hostname:	localhost
href:	http://localhost/tw/js/dom2.html#punto
origin:	http://localhost
pathname:	/tw/js/dom2.html
port:	
protocol:	http:
reload:	reload()
replace:	replace()
search:	"
toJSON:	undefined
toString:	toString()
valueOf:	valueOf()

BOM: El objeto window
Información del navegador

La propiedad window.navigator

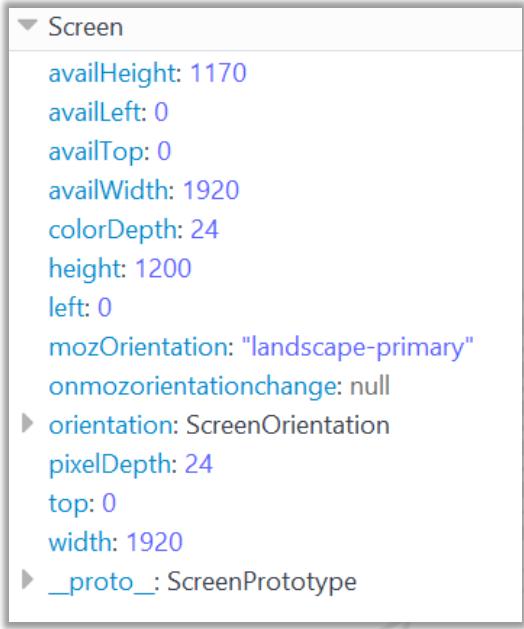
- Tiene propiedades con información relativa al navegador
- Algunas de ellas pueden variar según navegador

Propiedad	Valor (Chromium/Linux)	Valor (Firefox/Windows)
appCodeName	Mozilla	Mozilla
appName	Netscape	Netscape
appVersion	5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)	5.0 (Windows) buildID: 20161104212021
cookieEnabled	true	true
credentials	CredentialsContainer	unspecified
doNotTrack	null	unspecified
geolocation	Geolocation	Geolocation
hardwareConcurrency	12	2
language	es	es-ES
languages	[es]	[es-ES]
maxTouchPoints	0	0
mediaDevices	MediaDevices	MediaDevices
mimeTypeArray	MimeTypeArray	MimeTypeArray
onLine	true	true
permissions	Permissions	Permissions
platform	Linux x86_64	Win32
plugins	PluginArray	PluginArray
presentation	Presentation	Presentation
product	Gecko	Gecko
productSub	20030107	20100101
serviceWorker	ServiceWorkerContainer	ServiceWorkerContainer
userAgent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)	Mozilla/5.0 (Windows NT 6.2; rv:50.0) Gecko/20100101 Firefox/50.0
vendor	Google Inc.	unspecified
vendorSub	""	""
webkitPersistentStorage	DeprecatedStorageQuota	unspecified
webkitTemporaryStorage	DeprecatedStorageQuota	unspecified
__proto__	Navigator	Navigator

BOM: El objeto window
Información de la pantalla

La propiedad `window.screen`

Tiene propiedades con información relativa a la resolución y aspecto de la pantalla



```

Screen
availHeight: 1170
availLeft: 0
availTop: 0
availWidth: 1920
colorDepth: 24
height: 1200
left: 0
mozOrientation: "landscape-primary"
onmozorientationchange: null
orientation: ScreenOrientation
pixelDepth: 24
top: 0
width: 1920
__proto__: ScreenPrototype
  
```

BOM: El objeto window
Historial de navegación

La propiedad `window.history`

Tiene propiedades con información relativa al historial de navegación de la pestaña o ventana

<code>window.history.length</code>	Número de elementos del historial
<code>window.history.back()</code>	Avanzar
<code>window.history.forward()</code>	Retroceder
<code>window.history.go(N)</code>	Avanzar N elementos en el historial (Puede ser negativo)

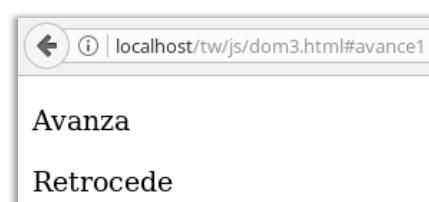
```

<!DOCTYPE html> <html> <head> <meta charset="utf-8">
<title>Ejemplo DOM</title> <script>
function avanzar() { window.history.forward(); console.log("avanza"); }
function retroceder() { window.history.back(); console.log("retrocede"); }
</script> </head>
<body> <p id="avanza" onclick="avanzar()">Avanza</p>
      <p id="retro" onclick="retroceder()">Retrocede</p> </body> </html>
  
```



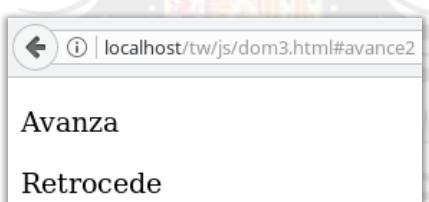
Avanza

Retrocede



Avanza

Retrocede



Avanza

Retrocede



UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

- 1. El lenguaje JavaScript
- 2. Client-side JavaScript
 - 1. Conceptos básicos
 - 2. El objeto window
 - 3. DOM
 - 4. Eventos
 - 5. Validación de formularios
 - 6. AJAX

»»»

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada © Javier Martínez Baena

DOM: Document Object Model

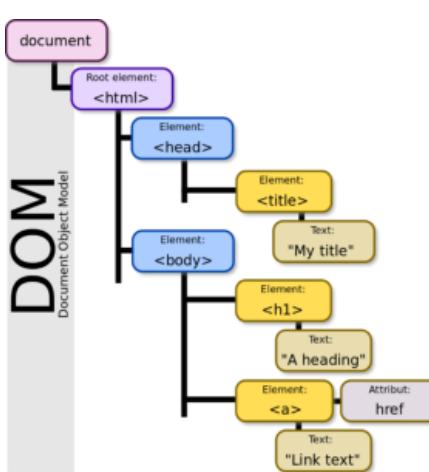
Document Object Model (DOM)

DOM (Document Object Model)

Es una **representación** del documento mostrado en el navegador que incluye una **API** para manipularlo dinámicamente.

Es independiente del lenguaje (existe en muchos lenguajes)

W3C se encarga de mantener el estándar del DOM
<https://www.w3.org/DOM/DOMTR>



- Recorrer la estructura jerárquica
- Añadir nodos
- Modificar nodos
- Borrar nodos
- Buscar nodos

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



DOM: Document Object Model

Nodos del DOM

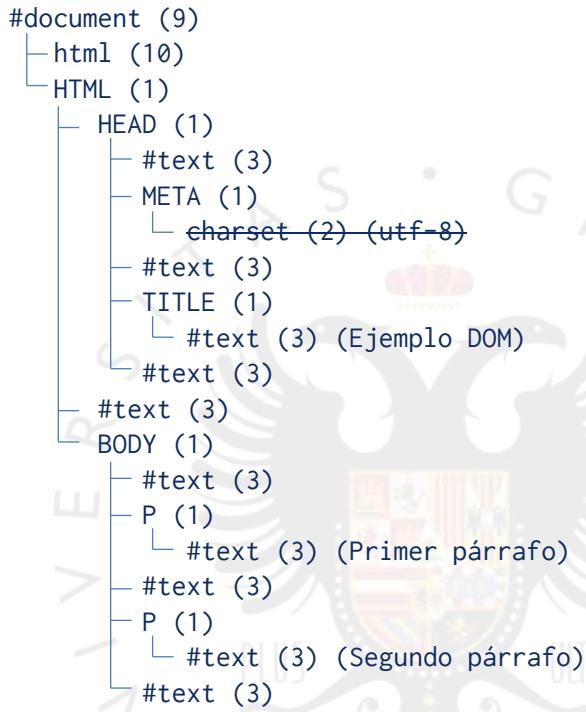
DOM (Document Object Model)

Todos los elementos de la jerarquía del DOM son de tipo Node:

- El documento completo
- Cada elemento HTML
- El texto de los elementos HTML
- Los atributos HTML
- Los comentarios

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Ejemplo DOM</title>
  </head>
  <body>
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
  </body>
</html>
```

1: ELEMENT_NODE 2: ATTRIBUTE_NODE
 3: TEXT_NODE 9: DOCUMENT_NODE
 10: DOCUMENT_TYPE_NODE



nodeName (nodeType) (nodeValue)

© Javier Martínez Baena

21

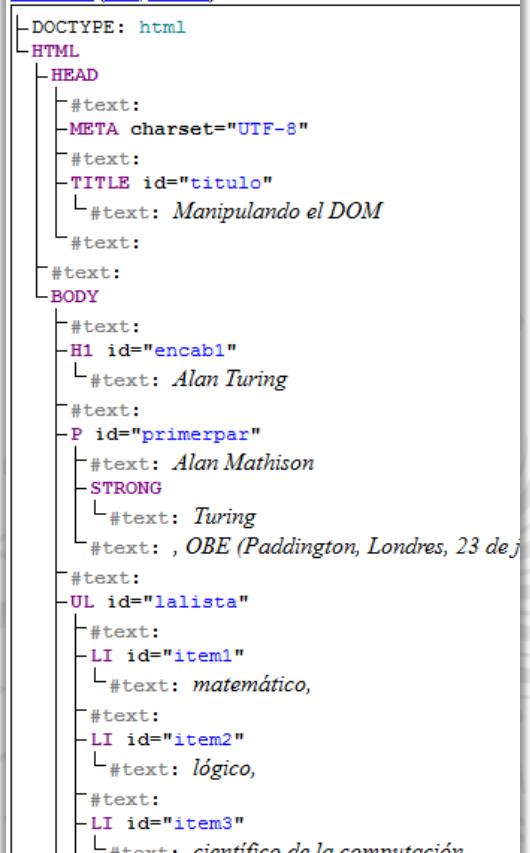


DOM: Document Object Model

Tipos de nodos del DOM

DOM view (hide, refresh):

```
<body id="fondo">
<h1 id="encab1">Alan Turing</h1>
<p id="primerpar">Alan Mathison
<strong>Turing</strong>, OBE (Paddington, Londres,
23 de junio de 1912-Wilmslow, Cheshire, 7 de junio
de 1954), fue un:</p>
<ul id="lalista">
  <li id="item1">matemático,</li>
  <li id="item2">lógico,</li>
  <li id="item3">científico de la computación,</li>
  <li id="item4">criptógrafo,</li>
  <li id="item5">filósofo</li>
</ul>
```



Visualización on-line del DOM:

<https://software.hixie.ch/utilities/js/live-dom-viewer>



DOM: Document Object Model

Tipos de nodos del DOM

```
<body id="fondo"><h1 id="encab1">Alan Turing</h1><p id="primerpar">Alan Mathison Turing</strong>, OBE (Paddington, Londres, 23 de junio de 1912-Wilmslow, Cheshire, 7 de junio de 1954), fue un:</p><ul id="lalista"><li id="item1">matemático,</li><li id="item2">lógico,</li><li id="item3">científico de la computación,</li><li id="item4">criptógrafo,</li><li id="item5">filósofo</li></ul>
```

- La estructura jerárquica de elementos es la misma
- Los nodos de texto pueden cambiar

DOM view (hide, refresh):

```
- DOCTYPE: html
- HTML
  - HEAD
    - #text:
    - META charset="UTF-8"
    - #text:
    - TITLE id="titulo"
      - #text: Manipulando el DOM
    - #text:
    - #text:
  - BODY
    - H1 id="encab1"
      - #text: Alan Turing
    - P id="primerpar"
      - #text: Alan Mathison
    - STRONG
      - #text: Turing
      - #text: , OBE (Paddington, Londres, 23 d
    - UL id="lalista"
      - LI id="item1"
        - #text: matemático,
      - LI id="item2"
        - #text: lógico,
      - LI id="item3"
        - #text: científico de la computación,
      - LI id="item4"
        - #text: criptógrafo,
      - LI id="item5"
```

DOM: Document Object Model

DOM: objetos y jerarquía de clases



Interface de Node

Atributos:	nodeType parentNode lastChild attributes	nodeName childNodes previousSibling textContent	nodeValue firstChild nextSibling removeChild() cloneNode()
Métodos:	insertBefore() appendChild() ...	replaceChild() cloneNode()	hasChildNodes()
...			

1: ELEMENT_NODE	5: ENTITY_REFERENCE_NODE	9: DOCUMENT_NODE
2: ATTRIBUTE_NODE	6: ENTITY_NODE	10: DOCUMENT_TYPE_NODE
3: TEXT_NODE	7: PROCESSING_INSTRUCTION_NODE	11: DOCUMENT_FRAGMENT_NODE
4: CDATA_SECTION_NODE	8: COMMENT_NODE	12: NOTATION_NODE

Si tipo es ELEMENT_NODE, nodeName es el TAG name de HTML

Si es de tipo ATTRIBUTE_NODE, nodeValue es el valor del TAG HTML

Si es de tipo TEXT_NODE, nodeValue es el valor del nodo (texto)

...

attributes es un array asociativo con los atributos del nodo
 textContent es una cadena con la concatenación del contenido de texto del nodo y todos sus descendientes

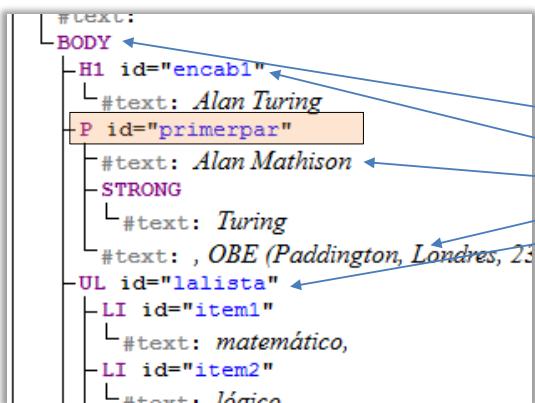
DOM: Document Object Model

Recorriendo el DOM



Propiedades de parentesco de los nodos del DOM

parentNode	elemento padre del nodo actual
childNodes	lista de elementos hijos
firstChild	primer hijo
lastChild	último hijo
nextSibling	siguiente hermano
previousSibling	hermano anterior



Si nodo es el que hay marcado:

nodo.parentNode;
nodo.previousSibling;
nodo.firstChild;
nodo.lastChild;
nodo.nextSibling;

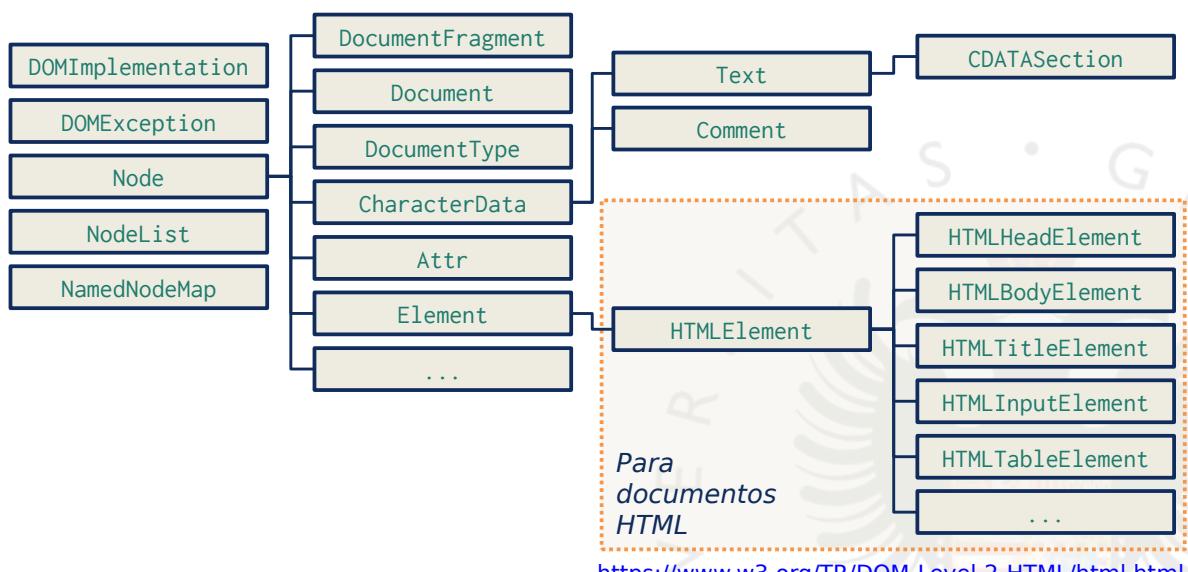


DOM: Document Object Model

Jerarquía de tipos para los nodos del DOM

Tipos de nodos

Cada tipo de nodo implementa interfaces especializadas



En todos los nodos se puede usar la API de Node y, además, otras API específicas del tipo de nodo de que se trate.

DOM: Document Object Model
Recorriendo el DOM

```

    BODY
    |
    +-- #text: 
    |   +-- H1 id="encab1"
    |       +-- #text: Alan Turing
    |
    +-- #text: 
    |   +-- P id="primerpar"
    |       +-- #text: Alan Mathison
    |       +-- STRONG id="tur"
    |           +-- #text: Turing
    |           +-- #text: , OBE (Paddington,
    |
    +-- #text: 
    |   +-- UL id="lalista"
    |       +-- #text:
  
```

HTML Code:

```

<body>
<h1 id="encab1">Alan Turing</h1>
<p id="primerpar">Alan Mathison <strong id="tur">Turing</strong>, OBE (Paddington, Londres, 23 de junio de 1912-Wilmslow, Cheshire, 7 de junio de 1954), fue un:</p>
<ul id="lalista">
  <li id="item1">matemático,</li>
  <li id="item2">lógico,</li>
  ...
  ...

var nodo=document.getElementById("primerpar").firstChild;
while (nodo!=null) {
  if (nodo.nodeType==Node.ELEMENT_NODE) // tipo 1
    console.log("Elemento ID: "+nodo.id);
  else if (nodo.nodeType==Node.TEXT_NODE) // tipo 3
    console.log("Nodo de texto: "+nodo.nodeValue);
  else
    console.log("Tipo de nodo desconocido");
  nodo=nodo.nextSibling;
}
  
```

Console Output:

```

Nodo de texto: Alan Mathison
Elemento ID: tur
Nodo de texto: , OBE (Paddington, Londres, 23 de junio de 1912-Wilmslow, Cheshire, 7 de junio de 1954), fue un:
  
```

DOM: Document Object Model
Accediendo a elementos por tipo o clase

getElementsByTagName

Accede a los elementos HTML del DOM de un determinado tipo
Devuelve un HTMLCollection (tratamiento similar a un array)

Code:

```

var ps=document.getElementsByTagName("p");
console.log(ps);
for (var i=0; i<ps.length; i++)
  console.log(ps[i].id);
  
```

Output:

```

HTMLCollection [ <p#primerpar> , <p#segundopar> ]
  primerpar
  segundopar
  
```

getElementsByClassName

Disponible en DOM 4

- Accede a los elementos del DOM de una determinada clase
- Se pueden especificar varias clases en la llamada
- Devuelve un elemento de tipo HTMLCollection



Listas "live" de nodos

Las listas de elementos devueltas por `getElementsByTag`, `getElementsByClass` se dicen "live" ya que se actualizan de forma dinámica acorde al contenido del documento HTML

```
<script>
function f() {
    var elems=document.getElementsByTagName("li");

    console.log("Lista de items:");
    for (var i=0; i<elems.length; i++)
        console.log(elems[i].id);

    // Borrar un item
    var uno=document.getElementById("item3");
    uno.parentNode.removeChild(uno);

    console.log("Lista de items:")
    for (var i=0; i<elems.length; i++)
        console.log(elems[i].id);
}
onload=f;
</script>
```

Lista de items:

item1

item2

item3

item4

item5

Lista de items:

item1

item2

item4

item5



Interface de element

Atributos:

- `tagName`
- ...

Métodos:

- | | |
|---------------------------------------|--|
| • <code>getAttribute()</code> | Devuelve el valor de un atributo |
| • <code>getAttributeNode()</code> | Devuelve el nodo de un atributo |
| • <code>getElementsByTagName()</code> | Devuelve descendientes con un TAG name |
| • <code>hasAttribute()</code> | Devuelve si tiene o no un atributo |
| • <code>removeAttribute()</code> | Elimina un atributo |
| • <code>setAttribute()</code> | Añade un atributo |
| • <code>setAttributeNode()</code> | Añade un nuevo nodo atributo |
| • ... | |

```
<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>Ejemplo DOM</title>
<script> onload = function() {
    document.getElementById("parrafo1").setAttribute("style","color:red");
    document.getElementById("parrafo2").textContent =
        document.getElementById("parrafo1").getAttribute("style"); } </script>
</head> <body>
<p id="parrafo1">Primer párrafo</p>
<p id="parrafo2">Segundo párrafo</p>
</body> </html>
```

Primer párrafo

color:red



Interface de text

Atributos:

- isElementContentWhitespace
Devuelve si el contenido es un espacio no visualizable
- wholeText
Devuelve el texto de este y todos sus nodos “lógicamente-adyacentes” como una cadena

Métodos:

- replaceWholeText()
Reemplaza este nodo y sus “lógicamente-adyacentes” por un nodo con un determinado texto

```
p1 = document.getElementById("parrafo1");
p2 = document.getElementById("parrafo2");
p1.childNodes[0].textContent = "Hola";
t1 = document.createTextNode("Pepe");
t2 = document.createTextNode("Juan");
p1.appendChild(t1);
p1.appendChild(t2);
console.log(t1.wholeText); // HolaPepeJuan
console.log(t1.textContent); // Pepe
p1.normalize();
console.log(p1.childNodes[0].textContent); // HolaPepeJuan
console.log(p2.childNodes[1].textContent); // ERROR, no existe
```

```
<body>
  <p id="parrafo1">Primer párrafo</p>
  <p id="parrafo2">Segundo párrafo</p>
</body>
```



Interface de attr

Los atributos son nodos (pueden usar su interface) aunque no se representan en el DOM tree (especificación DOM 3)

Atributos:

- isID Indica si es de tipo ID
- name Nombre del atributo
- value Valor del atributo

Uso desaconsejado. El interface de element ya incluye getAttribute y setAttribute para manipular los atributos. En DOM 4 (próximo estándar), attr no hereda de node

```
document.getElementById("parrafo1").getATTRIBUTE("style").value="color:green";
document.getElementById("parrafo2").textContent =
  document.getElementById("parrafo1").getATTRIBUTE("style").value;
```

```
<body>
<p id="parrafo1" style="color:red">Primer párrafo</p>
<p id="parrafo2">Segundo párrafo</p>
</body>
```

Primer párrafo
color:green



Interfaces especializadas de elementos HTML

Para facilitar la manipulación, proveen atributos con acceso directo a los atributos definidos en HTML

```
p1 = document.getElementById("parrafo1");
console.log(p1.id);
console.log(p1.className);
console.log(p1.align)
b = document.getElementById("body");
b.bgColor = "LightGreen";
p1.style="color:red";

<body id="body">
  <p id="parrafo1" class="pclas" align="center">Primer párrafo</p>
</body>
```

Interfaces especializadas para tratar el estilo CSS

El atributo style proviene de una interfaz especializada para manipular estilos inline (ElementCSSInlineStyle)

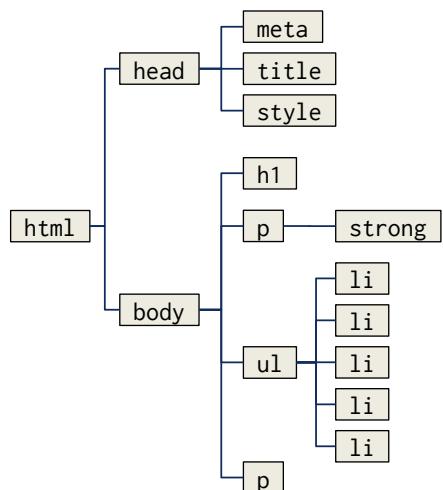
<https://www.w3.org/TR/DOM-Level-2-HTML/>

<https://www.w3.org/TR/DOM-Level-2-Style/>



Interfaces especializadas de elementos HTML

Los atributos de los elementos de HTML son accesibles mediante propiedades de los objetos representados en el DOM



```
<script>
function f() {
  var elem1=document.getElementById("lalista");
  elem1.type = "1";
  var elem2=document.getElementById("encab1");
  elem2.align = "center";
}
onload=f;
</script>
```

Alan Turing

Alan Mathison Turing, OBE (Paddington, Londres, 23 de junio de 1912-Wilmslow, Cheshire, 7 de junio de 1954), fue un:

1. matemático,
2. lógico,
3. científico de la computación,
4. criptógrafo,
5. filósofo

Es considerado uno de los padres de la ciencia de la computación y precursor de la informática moderna.

DOM: Document Object Model
Resumen de API

node	element	document
nodeType nodeName nodeValue parentNode childNodes firstChild lastChild previousSibling nextSibling Attributes textContent insertBefore() replaceChild() removeChild() appendChild() cloneNode() hasChildNodes() normalize()	tagName getAttribute() getAttributeNode() getElementsByName() hasAttribute() removeAttribute() setAttribute() setAttributeNode()	documentElement Doctype inputEncoding documentURI createAttribute() createDocumentFragment() createElement() createTextNode() getElementById() getElementsByName()
text		attr
isElementContentWhitespace wholeText replaceWholeText()		isID name value

https://www.w3.org/TR/#tr_DOM

DOM: Document Object Model
Interface de serialización

Interface de serialización

Se usa para parsear y serializar nodos del DOM

- Obtener una estructura DOM a partir de una cadena HTML
- Obtener una cadena HTML a partir de un DOM

Extensión a la interface de element:

- **innerHTML**: sustituir el contenido de un nodo por el árbol que resulta de la cadena HTML indicada
- **outerHTML**: ídem pero incluyendo al nodo en la sustitución

```

<head>
  <meta charset="UTF-8">
  <title id="titulo">Manipulando el DOM</title>
  <script>
    function f() {
      var elem=document.getElementById("segundopar");
      elem.innerHTML = "blablabla";
    }
    onload=f;
  </script>
</head>

```

- científico de la computación,
- criptógrafo,
- filósofo

blablabla

<https://www.w3.org/TR/DOM-Parsing/>



```
list1 = document.getElementById("lista1");
lista1.innerHTML = "<ul><li>Otro elemento</li><ul>";
```

```
list2 = document.getElementById("lista2");
lista2.outerHTML = "<ul><li>Otro elemento</li><ul>";
```

```
<p>Primera lista:</p>
<ol id="lista1">
  <li>Elemento 1</li>
  <li>Elemento 2</li>
</ol>
<p>Segunda lista:</p>
<ol id="lista2">
  <li>Elemento 1</li>
  <li>Elemento 2</li>
</ol>
```

Primera lista:

- Otro elemento

Segunda lista:

- Otro elemento



Interface NodeSelector

API para seleccionar nodos del DOM que emparejan con selectores CSS. Se puede usar en elementos de tipo: Document, documentFragment, Element

Métodos:

- | | |
|----------------------|---|
| • querySelector() | Devuelve el primer elemento que empareja con el selector |
| • querySelectorAll() | Devuelve una lista de nodos (NodeList) que emparejan con el selector. La lista no es "live" |

```
<body id="fondo">
<h1 id="encab1">Alan Turing</h1>
<p id="primerpar">Alan Mathison <strong class="clasedes">Turing</strong>, OBE
(Paddington, <strong>Londres</strong>, 23 de junio de 1912-Wilmslow, Cheshire, 7
de junio de 1954), fue un:</p>
<ul id="lalista">
  <li id="item1">matemático,</li>
  <li id="item2" class="clasedes">lógico,</li>
  <li id="item3">científico de la computación,</li>
  <li id="item4" class="clasedes"><strong>criptógrafo</strong>,</li>
  <li id="item5">filósofo</li>
</ul>
</body>
```

<https://dev.w3.org/2006/webapi/selectors-api2/>

DOM: Document Object Model
Accediendo a los elementos por selectores (CSS)

```

console.log("Selector: .clasedes");
var nodos = document.querySelectorAll(".clasedes");
for (var n=0; n<nodos.length; n++)
    console.log(nodos[n].nodeName+ ' / '+nodos[n].textContent);

console.log("Selector: li.clasedes");
var nodos = document.querySelectorAll("li.clasedes");
for (var n=0; n<nodos.length; n++)
    console.log(nodos[n].nodeName+ ' / '+nodos[n].textContent);

console.log("Selector: li:nth-child(odd)");
var nodos = document.querySelectorAll("li:nth-child(odd)");
for (var n=0; n<nodos.length; n++)
    console.log(nodos[n].nodeName+ ' / '+nodos[n].textContent);

console.log("Selector: li strong");
var nodos = document.querySelectorAll("li strong");
for (var n=0; n<nodos.length; n++)
    console.log(nodos[n].nodeName+ ' / '+nodos[n].textContent);

```

Selector: .clasedes	STRONG / Turing
LI / lógico,	LI / criptógrafo,
Selector: li.clasedes	LI / lógico,
LI / criptógrafo,	LI / matemático,
Selector: li:nth-child(odd)	LI / científico de la computación,
LI / filósofo	LI / filósofo
Selector: li strong	STRONG / criptógrafo

<https://dev.w3.org/2006/webapi/selectors-api2/>

DOM: Document Object Model
Interfaces para manipular reglas de estilo CSS

Interfaces para controlar las reglas de estilo CSS
StyleSheet, StyleSheetList, MediaList
CSSStyleSheet, CSSRuleList, CSSRule, CSSStyleRule, ...

<https://www.w3.org/TR/DOM-Level-2-Style/>


UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

- 1. El lenguaje JavaScript
- 2. Client-side JavaScript
 - 1. Conceptos básicos
 - 2. El objeto window
 - 3. DOM
 - 4. Eventos
 - 5. Validación de formularios
 - 6. AJAX

»»»

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada © Javier Martínez Baena

Eventos en JavaScript

Introducción



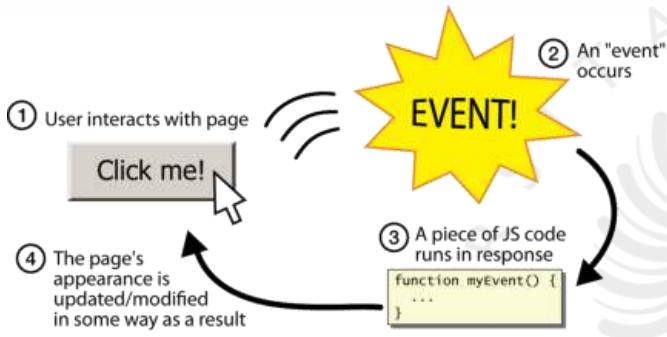
La programación JS se basa en eventos (asíncronos)

En JS:

- Se implementan los manejadores de eventos (funciones)
- Se registran los manejadores de eventos (se asocian a un evento)

El navegador:

- Captura eventos (de usuario, de red, ...)
- Ejecuta el manejador correspondiente



Recomendation: <https://www.w3.org/TR/DOM-Level-2-Events/events.html>
 Working Draft: <https://www.w3.org/TR/uievents/>
<https://developer.mozilla.org/en-US/docs/Web/Events>

<https://javascript.info/events>

<http://www.webstepbook.com/supplements-2ed/slides/chapter08-javascript.shtml#slide11>



Tipos de eventos: Interface UIEvent

Eventos generados por el user-agent y asociados a la interfaz de usuario.

- load Generado cuando acaba de cargarse un documento
- unload Generado cuando se elimina un documento del entorno
- abort Generado al abortar la carga de un recurso
- error Generado al producirse un error de carga/interpretación
- select Generado cuando un usuario selecciona un texto

Tipos de eventos: Interface FocusEvent

Eventos relacionados con el control del foco.

- focus Generado al ganar el foco
- focusin Generado al ganar el foco él o cualquier descendiente
- blur Generado al perderse el foco
- focusout Generado al perder el foco él o cualquier descendiente



Programando eventos en HTML

Se asigna un valor a un atributo de un elemento HTML

- El atributo se llama "on" seguido del nombre del evento
- El valor del atributo es el cuerpo del manejador (JS)
- Esta forma de programar eventos **no está recomendada**.
- Provoca alto acoplamiento entre HTML y JS

```
<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>T</title> </head>
<body>
<section onfocus="console.log('section focus');"
         onfocusin="console.log('section focusin');"
         onblur="console.log('section blur');"
         onfocusout="console.log('section focusout');">
<input type="text" onfocus="console.log('input focus');"
       onfocusin="console.log('input focusin');"
       onblur="console.log('input blur');"
       onfocusout="console.log('input focusout');">
</section>
</body> </html>
```

input focus
input focusin
section focusin
input blur
input focusout
section focusout



Tipos de eventos: Interface MouseEvent

screenX, screenY Coordenadas de pantalla absolutas
 clientX, clientY Coordenadas relativas a la ventana del navegador
 ctrlKey, shiftKey, altKey, metaKey Si están o no pulsadas
 button Botón pulsado (0=primario, 1=medio, 2=secundario)
 buttons Bitmask con estado de botones (1 bit para estado de cada botón)

Eventos relacionados con el ratón.

- click Generado al hacer click (pulsa y suelta)
- dblclick Generado con doble click
- mousedown Generado al pulsar un botón
- mouseup Generado al soltar un botón
- mouseenter Generado al entrar el cursor del ratón en el elemento o en alguno de sus descendientes
- mouseover Generado al entrar el cursor del ratón en el elemento
- mouseleave Generado al salir el cursor del ratón del elemento o de sus descendientes
- mouseout Generado al salir el cursor del ratón del elemento
- mousemove Generado al moverse el ratón

Tipos de eventos: Interface WheelEvent

wheel Se genera al rotar la rueda



Ejemplo

Recuerde: esta forma de programar eventos no se recomienda

```

<body id="fondo">
  <div id="div1" onmousedown=
    "var obj = document.createElement('p');
     obj.textContent='Pulsado sobre div1';
     document.getElementsByTagName('body')[0].appendChild(obj);" >
    <p>Texto de primer nivel</p>
    <div id="div2">
      <p>Texto interior</p>
    </div>
  </div>
</body>
  
```

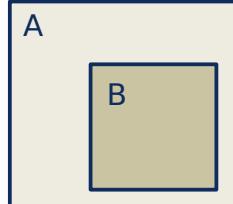


Eventos en JavaScript

Eventos de ratón

Secuencia habitual de eventos de ratón al entrar a un elemento A

Event Type	Element
1 <u>mousemove</u>	
2 <u>mouseover</u>	A
3 <u>mouseenter</u>	A
4 <u>mousemove</u>	A
5 <u>mouseout</u>	A
6 <u>mouseleave</u>	A



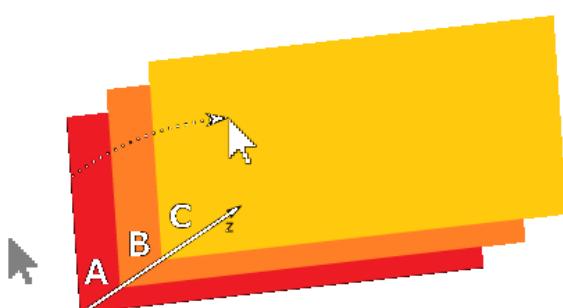
Elementos anidados

Event Type Element

1	<u>mousemove</u>	
2	<u>mouseover</u>	A
3	<u>mouseenter</u>	A
4	<u>mousemove</u>	A
5	<u>mouseout</u>	A
6	<u>mouseover</u>	B
7	<u>mouseenter</u>	B
8	<u>mousemove</u>	B
9	<u>mouseout</u>	B
10	<u>mouseleave</u>	B
11	<u>mouseover</u>	A
12	<u>mousemove</u>	A
13	<u>mouseout</u>	A
14	<u>mouseleave</u>	A

<https://www.w3.org/TR/uievents/>

Elementos superpuestos



Eventos en JavaScript

Eventos de ratón

Event Type Element

1	<u>mousemove</u>	
2	<u>mouseover</u>	C
3	<u>mouseenter</u>	A
4	<u>mouseenter</u>	B
5	<u>mouseenter</u>	C
6	<u>mousemove</u>	C
7	<u>mouseout</u>	C
8	<u>mouseleave</u>	C
9	<u>mouseleave</u>	B
10	<u>mouseleave</u>	A

<https://www.w3.org/TR/uievents/>

Eventos en JavaScript
Eventos de cambio de contenidos

Tipos de eventos: Interface InputEvent

Eventos que se activan cuando el DOM es modificado (se usa en formularios al cambiar valores de campos editables)

- beforeinput Generado justo antes de modificar un elemento editable
- input Generado al cambiar contenido editable

```
<body id='cuerpo'>
<form action='' id='formulario'>
    Escribe algo:
    <input type='text' id='caja'
        oninput=
            'document.getElementById("cuerpo").appendChild(document.createTextNode("."));'/>
</form>
</body>
```

Escribe algo:

Escribe algo: H

.

Escribe algo: Ho

..

Escribe algo: Hol

...

Escribe algo: Hola I

....

Eventos en JavaScript
Eventos de teclado

Tipos de eventos: Interface KeyboardEvent

key	Valor de la tecla pulsada. Si es una tecla especial contiene una cadena indicándolo https://www.w3.org/TR/uievents-key/
code	Código (cadena) que representa la tecla pulsada https://www.w3.org/TR/uievents-code/
location	Situación en el teclado (standard, left, right, numpad)
ctrlKey, shiftKey, altKey, metaKey	Estado de estas teclas
repeat	El evento es consecuencia de mantener pulsada una tecla

Eventos:

- keydown Generado al pulsar una tecla
- keyup Generado al soltar una tecla

Tecla	Key	Code
a	a	KeyA
Shift+a	A	KeyA
Shift	Shift	ShiftLeft / ShiftRight
0	0	Digit0
= (Shift+0)	=	Digit0

keydown
beforeinput
input
keyup

keydown
beforeinput
input



Eventos en JavaScript

Eventos de teclado

KeyboardEvent . key <pre>"Shift", "Control", "Alt", "Meta" "ArrowDown", "ArrowLeft", "ArrowRight", "ArrowUp" "End", "Home", "PageDown", "PageUp" "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ".", "Enter", "+", "-", "*", "/"</pre>	Valid location values <pre>DOM_KEY_LOCATION_LEFT, DOM_KEY_LOCATION_RIGHT DOM_KEY_LOCATION_STANDARD, DOM_KEY_LOCATION_NUMPAD DOM_KEY_LOCATION_STANDARD, DOM_KEY_LOCATION_NUMPAD DOM_KEY_LOCATION_STANDARD, DOM_KEY_LOCATION_NUMPAD</pre>
--	---

<https://www.w3.org/TR/uievents/>



Eventos en JavaScript

El manejador de eventos (event handler)

El manejador de eventos

Asignar código JS a un atributo en el código HTML es mala idea
Manejador de eventos: función que se ejecuta al dispararse un evento

```
function h() {  
    var obj = document.createElement("p");  
    obj.textContent="Pulsado sobre div1";  
    document.getElementsByTagName("body")[0].appendChild(obj);  
}  
  
function f() {  
    var div1 = document.getElementById("div1");  
    div1.onmousedown = h;  
}  
  
onload=f;
```

Texto de primer nivel

Texto interior

<body id="fondo">
 <div id="div1">
 <p>Texto de primer nivel</p>
 <div id="div2">
 <p>Texto interior</p>
 </div>
 </div>
</body>

- Limitación: un único manejador por tipo de evento
- Conceptualmente: equivalente a asignar propiedad en HTML pero ...
... desacopla JS y HTML

Para cancelar el manejador de eventos:

```
div1.onmousedown = null;
```



Conceptos básicos sobre eventos

- Event target: objeto en el que ha ocurrido el evento
- Event handler/listener: función que se ejecuta cuando ocurre el evento
- Event object: objeto con información sobre el evento

```
<body>
  <p id="p1">Primer párrafo</p>
</body>
```

```
function h(e) {
  var obj = document.createElement("p");
  obj.textContent="Pulsado sobre "+e.target.id+" en ("+e.clientX+","+e.clientY+")";
  document.getElementsByTagName("body")[0].appendChild(obj);
}

function f() {
  var p = document.getElementById("p1");
  p.onmousedown = h;
}
window.onload=f;
```

h: event handler
e: event object
e.target: event target

Primer párrafo
Pulsado sobre p1 en (87,27)
Pulsado sobre p1 en (27,22)
Pulsado sobre p1 en (127,32)

El navegador le pasa al manejador de eventos, como primer argumento, el "event object"

Cómo se registra un manejador de eventos (**método preferido**)

Llamando a un método específico para ello:

`addEventListener` / `attachEvent` (en el caso de IE<=8)

```
function h(e) {
  var obj = document.createElement("p");
  obj.textContent="Pulsado sobre div1";
  document.getElementsByTagName("body")[0].appendChild(obj);
}

function f() {
  var div1 = document.getElementById("div1");
  div1.addEventListener("mousedown",h);
}
```

Ventajas:

- múltiples manejadores por evento
- bajo acoplamiento HTML/JS

Orden de ejecución de los manejadores de eventos

1. Manejadores registrados asignando una propiedad de un objeto o un atributo HTML
2. Manejadores registrados con `attachEventListener`.
 - DOM2: no se garantiza ningún orden concreto.
 - DOM3: orden en que han sido registrados

Eventos en JavaScript
Propagación de eventos

Propagación de eventos

¿Qué ocurre cuando hay elementos HTML anidados?

```
<body id="fondo">
  <div id="div1">
    <p id="pext">Texto de primer nivel</p>
    <div id="div2">
      <p id="pint">Texto interior</p>
    </div>
  </div>
</body>
```

Texto de primer nivel
Texto interior

body
div: div1
p: pext
div: div2
p: pint

div1
pext
div2
pint

Eventos en JavaScript
Propagación de eventos

Propagación de eventos

¿Qué ocurre cuando hay elementos HTML anidados?

```
function h1(e) {
  console.log("Pulsado sobre div1.");
}

function h2(e) {
  console.log("Pulsado sobre div2.");
}

onload = function () {
  document.getElementById("div1").onclick = h1;
  document.getElementById("div2").onclick = h2;
}
```

Pulsado sobre div2.
Pulsado sobre div1.

Texto de primer nivel
Texto interior

div1
pext
div2
pint

Bubbling:
Los eventos se propagan ascendiendo por el árbol DOM



Eventos en JavaScript

Propagación de eventos

Fases en la propagación de eventos

Cuando se dispara un evento, este desencadena una secuencia de eventos en el DOM

Fase 1: capture

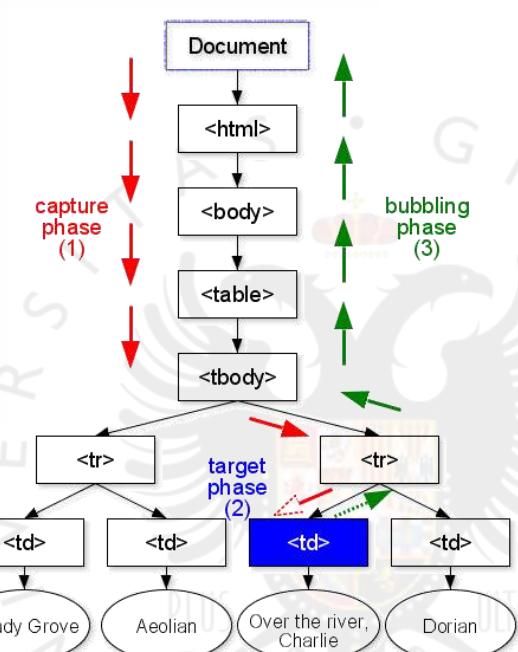
El evento se propaga desde la raíz hacia las hojas

Fase 2: target

El evento se dispara en el nodo más interno (hoja), el target object

Fase 3: bubbling

El evento se propaga desde las hojas hacia la raíz



<https://www.w3.org/TR/2006/WD-DOM-Level-3-Events-20060413/events.html>

Eventos en JavaScript

Propagación de eventos

Fases en la propagación de eventos

Casi todos los eventos tienen fase de bubbling ...
... aunque hay unos pocos que no (focus, blur, load, ...)

```
<!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>T</title> </head>
<body>
<section onfocus="console.log('section focus');"
         onfocusin="console.log('section focusin');"
         onblur="console.log('section blur');"
         onfocusout="console.log('section focusout');">
<input type="text" onfocus="console.log('input focus');"
       onfocusin="console.log('input focusin');"
       onblur="console.log('input blur');"
       onfocusout="console.log('input focusout');">
</section>
</body> </html>
```

I
input focus
input focusin
section focusin
input blur
input focusout
section focusout

Evento focus: no tiene fase bubbling
Evento focusin: sí tiene fase bubbling

Eventos en JavaScript
Atributos de los objetos de tipo evento

Interface event

Aporta información sobre el evento en curso al manejador, que lo recibe como primer argumento.

Atributos:

bubbles	Si el evento es de tipo bubbling o no
cancelable	Si se puede o no cancelar la acción por defecto
target	Target que ha originado el evento
currentTarget	Target que actualmente está ejecutando sus manejadores en el proceso de capturing/bubbling
eventPhase	En qué fase se encuentra actualmente el proceso CAPTURING_PHASE, AT_TARGET, BUBBLING_PHASE
type	Nombre del evento

Métodos:

preventDefault()	Cancelar la acción por defecto (si cancelable)
stopPropagation()	Cancelar la propagación a ancestros/descendientes

Eventos en JavaScript
Propagación de eventos

Propagación de eventos

Quien es quién ...

```

function h1(e) {
    console.log("Pulsado sobre div1.");
    console.log(" Type : " + e.type);
    console.log(" Target : " + e.target.id);
    console.log(" Current target : " + e.currentTarget.id);
}

function h2(e) {
    console.log("Pulsado sobre div2.");
    console.log(" Type : " + e.type);
    console.log(" Target : " + e.target.id);
    console.log(" Current target : " + e.currentTarget.id);
}

 onload = function () {
    document.getElementById("div1").addEventListener("click",h1);
    document.getElementById("div2").addEventListener("click",h2);
}

```

Texto de primer nivel

Texto interior

Pulsado sobre div2.
Type : click
Target : pint
Current target : div2

Pulsado sobre div1.
Type : click
Target : pint
Current target : div1

Eventos en JavaScript
Propagación de eventos

Propagación de eventos

Cambiando la fase de disparo del evento

```

function h1(e) {
    console.log("Pulsado sobre div1.");
    console.log(" Type : " + e.type);
    console.log(" Target : " + e.target.id);
    console.log(" Current target : " + e.currentTarget.id);
}

function h2(e) {
    console.log("Pulsado sobre div2.");
    console.log(" Type : " + e.type);
    console.log(" Target : " + e.target.id);
    console.log(" Current target : " + e.currentTarget.id);
}

onload = function () {
    document.getElementById("div1").addEventListener("click",h1,true);
    document.getElementById("div2").addEventListener("click",h2,true);
}

```

The diagram illustrates event propagation. A red box highlights the word 'true' in the call to addEventListener for the first div. A mouse cursor points to the text 'Texto interior' (inner text) inside a blue box. To the right, two boxes show log outputs for click events on different divs. The top box for div1 shows 'Target : pint' (pint is likely a typo for 'pintado') and 'Current target : div1'. The bottom box for div2 shows 'Target : pint' and 'Current target : div2'.

Eventos en JavaScript
Eventos y el objeto this

El objeto this en el manejador de eventos

- Está referido al elemento que contiene el manejador

```

<body>
    <section onclick="console.log(this.nodeName);">
        <p onclick="console.log(this.nodeName);>Párrafo 1</p>
        <p>Párrafo 2</p>
    </section>
</body>

```

The diagram shows two stages of event propagation. In the first stage, a red arrow points from the text 'Párrafo 1' to a box labeled 'P SECTION'. In the second stage, a red arrow points from the text 'Párrafo 2' to a box labeled 'SECTION'. This illustrates how 'this' refers to the current element in the event handler.

Cuando se propagan los eventos, this se corresponde con el currentTarget

Eventos en JavaScript

Cancelación de eventos



Cómo se cancela un manejador de eventos

Llamando a un método específico para ello:

`removeEventListener`
`detachEvent` (en el caso de IE<=8)

```
function h(e) {
    var obj = document.createElement("p");
    obj.textContent="Pulsado sobre div1";
    document.getElementsByTagName("body")[0].appendChild(obj);
    document.getElementById("div1").removeEventListener("mousedown",h);
}

function f() {
    var div1 = document.getElementById("div1");
    div1.addEventListener("mousedown",h);
}
```

No se pueden cancelar eventos programados con funciones anónimas

Eventos en JavaScript

El manejador de eventos



Valor devuelto por el manejador de eventos

Cuando el manejador se ha registrado asignando una propiedad/atributo

Devolver `false` le indica al navegador que no haga la acción asignada por defecto al evento.

Ejemplo: El botón “submit” de un formulario, por defecto, envía el formulario. Si se registra un manejador asociado a ese evento y devuelve `false`, al pulsar no se envía el formulario.

Valor devuelto por el manejador de eventos

Cuando el manejador se ha registrado con `attachEventListener`

No ha de devolver nada.

En caso de que el evento sea cancelable, se puede ejecutar `preventDefault()`. De esa forma se cancela la acción por defecto que debería realizar.



Eventos en JavaScript

Eventos cancelables

onclick: evento cancelable

```

<body>
  <a id='enlace' href="http://google.es">Ir a Google</a>
</body>

// Manejador de evento para asignar a través de un atributo
function g1() {
  alert('Ejecutando función g1()');
  return false; // Cancela acción por defecto (ir al enlace)
  return true; // Continuará con la acción por defecto (ir al enlace)
}

onload = function() {
  document.getElementById('enlace').onclick = g2;
}

// Manejador de evento para asignar con addEventListener
function g2(e) {
  alert('Ejecutando función g2()');
  e.preventDefault(); // Cancela acción por defecto (ir al enlace)
  return false; // Sin efecto ?
  return true; // Sin efecto ?
}

onload = function() {
  document.getElementById('enlace').addEventListener('click',g2);
}

```

Eventos en JavaScript

Temporizadores (timers)



Temporizadores

Son manejadores de eventos que se disparan transcurrido un cierto tiempo

setTimeout	Activa un temporizador Devuelve un número que representa el temporizador Puede llevar parámetros extra para pasarlo al manejador
clearInterval	Desactiva un temporizador
setInterval	Activa un temporizador y lo repite de forma periódica hasta que sea desactivado Puede llevar parámetros extra para pasarlo al manejador

```

function h() {
  console.log("Ejecutando h");
}

```

```

onload=function() {
  setTimeout(h,3000);
}

```

A los 3 segundos de haber cargado la página se ejecuta h()

Eventos en JavaScript
Temporizadores (timers)

```

var temporizador; // Variable global
function h() {
    console.log("Ejecutando h");
}
function cancela() {
    console.log("Cancelando timer");
    clearInterval(temporizador);
}
onload=function() {
    temporizador = setTimeout(h,5000);
    document.getElementById("par").onclick=cancela;
}

function h() {
    console.log("Ejecutando h");
}
function cancela(t) {
    console.log("Cancelando timer");
    clearInterval(t);
}
onload=function() {
    var temporizador = setInterval(h,2000);
    document.getElementById("par").onclick=function() {cancela(temporizador);};
}

```

A los 5 segundos de cargar la página muestra el mensaje salvo que pulsemos el texto

Ejecución de un evento periódico
Uso de setInterval
Sin variables globales

Tecnologías Web
3º Grado en Ingeniería Informática

UNIVERSIDAD DE GRANADA

Programación en el lado del cliente – Client-side JS

1. El lenguaje JavaScript
2. Client-side JavaScript

- 1. Conceptos básicos**
- 2. El objeto window**
- 3. DOM**
- 4. Eventos**

» 1. Ejemplo: 3 en raya

5. Validación de formularios
6. AJAX

DECSAI

Eventos en JavaScript

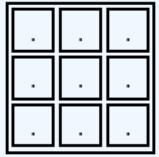
Ejemplo

Ejemplo: 3 en raya

Código HTML

```
<body>
  <div id="contenedor">
    <h1>3 en raya</h1>
    <table id="tablero">
      <tr> <td id="e00"></td> <td id="e01"></td> <td id="e02"></td> </tr>
      <tr> <td id="e10"></td> <td id="e11"></td> <td id="e12"></td> </tr>
      <tr> <td id="e20"></td> <td id="e21"></td> <td id="e22"></td> </tr>
    </table>
    <p>Turno: <span id="turno">X</span></p>
    <p>Información: <span id="info"></span></p>
  </div>
</body>
```

3 en raya



Turno: X

Información:

DECSAI

Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Código JS

```
// Registra los manejadores de eventos de cada casilla
function activarEventos() {
  document.getElementById("e00").addEventListener("click", function(){pulsaCelda(0,0);});
  document.getElementById("e01").addEventListener("click", function(){pulsaCelda(0,1);});
  document.getElementById("e02").addEventListener("click", function(){pulsaCelda(0,2);});
  document.getElementById("e10").addEventListener("click", function(){pulsaCelda(1,0);});
  document.getElementById("e11").addEventListener("click", function(){pulsaCelda(1,1);});
  document.getElementById("e12").addEventListener("click", function(){pulsaCelda(1,2);});
  document.getElementById("e20").addEventListener("click", function(){pulsaCelda(2,0);});
  document.getElementById("e21").addEventListener("click", function(){pulsaCelda(2,1);});
  document.getElementById("e22").addEventListener("click", function(){pulsaCelda(2,2);});
}

// Preparar juego al terminar de cargarse la página
onload = function() {
  activarEventos();
}
```

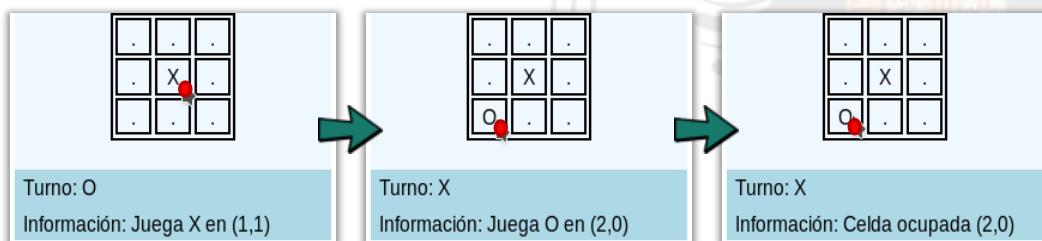
Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Código JS

```
// Acción a ejecutar al pulsar sobre una casilla del tablero
function pulsaCelda(f,c) {
    var celda = document.getElementById("e"+f+"_"+c);
    var turno = document.getElementById("turno");
    var info = document.getElementById("info");
    if (celda.textContent==".") { // Celda vacía aún
        // poner ficha, informar al usuario, pasar turno
        celda.textContent=turno.textContent;
        info.textContent = "Juega "+turno.textContent+ " en (" +f+", "+c+")";
        turno.textContent = turno.textContent=='X' ? 'O' : 'X';
    } else // Celda ocupada
        info.textContent = "Celda ocupada (" +f+", "+c+")";
    checkStatus(); // Comprobar si hay ganador o tablas
}
```



Eventos en JavaScript

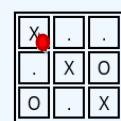
Ejemplo

Ejemplo: 3 en raya

Código JS

```
function checkStatus(){
    var e00 = document.getElementById("e00").textContent; var e01 = document.getElementById("e01").textContent;
    var e02 = document.getElementById("e02").textContent; var e10 = document.getElementById("e10").textContent;
    var e11 = document.getElementById("e11").textContent; var e12 = document.getElementById("e12").textContent;
    var e20 = document.getElementById("e20").textContent; var e21 = document.getElementById("e21").textContent;
    var e22 = document.getElementById("e22").textContent;
    var ganador = '';
    if ((e00!='.' && e00==e01 && e00==e02) || (e10!='.' && e10==e11 && e10==e12) ||
        (e20!='.' && e20==e21 && e20==e22) || (e00!='.' && e00==e10 && e00==e20) ||
        (e01!='.' && e01==e11 && e01==e21) || (e02!='.' && e02==e12 && e02==e22) ||
        (e00!='.' && e00==e11 && e00==e22) || (e02!='.' && e02==e11 && e02==e20)) {
        ganador = document.getElementById("turno").textContent=='X' ? 'O' : 'X';
    } else {
        if (e00!='.' && e01!='.' && e02!='.' && e10!='.' && e11!='.' && e12!='.' &&
            e20!='.' && e21!='.' && e22!='.')
            ganador = 'E';
    }
    if (ganador!='') {
        var info = document.getElementById("info");
        if (ganador=='E') info.textContent = "EMPATE";
        else info.textContent = "GANADOR: " +ganador;
    }
}
```

3 en raya



Turno: O
Información: GANADOR: X

Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Mejora: generalización usando API de HTMLTableElement

```
<body>
  <div id="contenedor">
    <h1>3 en raya</h1>
    <table id="tablero">
      <tr> <td></td> <td></td> <td></td> </tr>
      <tr> <td></td> <td></td> <td></td> </tr>
      <tr> <td></td> <td></td> <td></td> </tr>
    </table>
    <p>Turno: <span id="turno">X</span></p>
    <p>Información: <span id="info"></span></p>
  </div>
</body>
```

// Registra los manejadores de eventos de cada casilla

```
function activarEventos() {
  var tab = document.getElementById("tablero");
  for (var f=0; f<tab.rows.length; f++)
    for (var c=0; c<tab.rows[f].cells.length; c++)
      tab.rows[f].cells[c].addEventListener("click",function(){pulsaCelda(f,c);});
}
```

<https://developer.mozilla.org/es/docs/Web/API/HTMLTableElement>



Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Creación de una clausura para mantener el contexto del manejador

```
// Registra los manejadores de eventos de cada casilla
function activarEventos() {
  var tab = document.getElementById("tablero");
  for (var f=0; f<tab.rows.length; f++)
    for (var c=0; c<tab.rows[f].cells.length; c++)
      tab.rows[f].cells[c].addEventListener("click",function(){pulsaCelda(f,c);});
```



```
function activarEventos() {
  var tab = document.getElementById("tablero");
  for (var f=0; f<tab.rows.length; f++)
    for (var c=0; c<tab.rows[f].cells.length; c++) {
      ( function () {
        var mif=f;
        var mic=c;
        tab.rows[f].cells[c].addEventListener("click",
          function(){pulsaCelda(mif,mic);});
```

```
      }
    )(); // IIFE
  }
```

Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Cancelación de manejadores de eventos al finalizar la partida
 Los eventos programados con funciones anónimas no se pueden cancelar con removeEventHandler

```
function activarEventos() {
    var tab = document.getElementById("tablero");
    for (var f=0; f<tab.rows.length; f++)
        for (var c=0; c<tab.rows[f].cells.length; c++)
            ( function () {
                var mif=f;
                var mic=c;
                var lis = function() { pulsaCelda(mif,mic); };
                tab.rows[f].cells[c].addEventListener("click",lis);
                listeners.push({f:mif,c:mic,h:lis});
            })
    }()
}
```

Eventos en JavaScript

Ejemplo

Ejemplo: 3 en raya

Cancelación de manejadores de eventos al finalizar la partida
 Los eventos programados con funciones anónimas no se pueden cancelar con removeEventHandler

```
function checkStatus(){
    ...
    if (ganador!=='') {
        cancelarEventos(); // Cancelar eventos del tablero
        ...
    }

    var listeners = [];
    function cancelarEventos() {
        var tab = document.getElementById("tablero");
        while (listeners.length>0) {
            var ele = listeners.pop();
            tab.rows[ele.f].cells[ele.c].removeEventListener("click",ele.h);
        }
    }
}
```

Propuesta: implementa el juego del N en raya, donde N es el tamaño del tablero y es configurable



UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

- 1. El lenguaje JavaScript
- 2. Client-side JavaScript
 - 1. Conceptos básicos
 - 2. El objeto window
 - 3. DOM
 - 4. Eventos
 - 5. Validación de formularios**
 - 6. AJAX

»»»

El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

La opción no recomendada

Incrustar la programación del evento en el atributo HTML

```
<div class='formulario'>
<form name='misdatos' action='{$$_SERVER['SCRIPT_NAME']}' method='POST'
      onsubmit='return validarFormulario(this)'>

  <div class='frm_input'>
    <label for='nombre'>Nombre:</label>
    <input type='text' name='nombre' /> </div>
  <div class='frm_input'>
    <label for='email'>Email:</label>
    <input id='email' type='text' name='email' onblur='changeEmail(this)' /> </div>
  <div class='frm_input'>
    <label for='telefono'>Teléfono:</label>
    <input id='tele' type='text' name='telefono' onchange='changeTelefono(this)' />
  </div>
  <div class='frm_submit'>
    <input type='submit' name='accion' value='Enviar' />
  </div>
</form>
</div>
```

this: referido al elemento que provoca el evento (control del formulario)

Alto acoplamiento HTML/JS

Nombre:	<input type="text"/>
Email:	<input type="text"/>
Teléfono:	<input type="text"/>
Submit	



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

La opción **no recomendada**

Incrustar la programación del evento en el atributo HTML

```
function validarEmail(email) {
    // https://www.w3schools.com/jsref/prop_email_pattern.asp
    // Esto es lo mismo que comprueba el input type='email'
    return email.match(/^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,3}$/);
}

function validarTelefono(tel) {
    return tel.match(/^\((\([0-9]{2}\))?\s*[0-9]{3}\)\s*[0-9]{6}\$/);
}

function changeEmail(e) {
    if (!validarEmail(e.value))
        alert("Email incorrecto");
}

function changeTelefono(e) {
    if (!validarTelefono(e.value))
        alert("Número de teléfono no válido");
}
```

```
function validarFormulario(e) {
    var err = "";
    if (!validarTelefono(e.telefono.value))
        err += "El teléfono es incorrecto\n";
    if (!validarEmail(e.email.value))
        err += "El email es incorrecto\n";
    if (err!="")
        alert(err);
    return (err=="");
}
```



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

Acceso a formularios desde propiedad del DOM

El objeto `document` incluye algunas colecciones (`HTMLCollection`):

- `images`, `links`, `anchors`, ...
- `forms`: colección con todos los formularios. Array indexado y asociativo por el atributo `name` del formulario

```
<div class='formulario'>
    <form name='misdatos' action='{$$_SERVER['SCRIPT_NAME']}' method='POST'
          onsubmit='return validarFormulario(this)'>
        <div class='frm_input'>
            <label for='nombre'>Nombre:</label>
            <input type='text' name='nombre' /> </div>
        <div class='frm_input'>
            <label for='email'>Email:</label>
            <input id='email' type='text' name='email' onblur='changeEmail(this)' /> </div>
        <div class='frm_input'>
            <label for='telefono'>Teléfono:</label>
            <input id='tele' type='text' name='telefono' onchange='changeTelefono(this)' />
        </div>
        <div class='frm_submit'>
            <input type='submit' name='accion' value='Submit' /> </div>
    </form>
</div>
```



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

Acceso a formularios desde propiedad del DOM

El objeto `document` incluye algunas colecciones (`HTMLCollection`):

- `images`, `links`, `anchors`, ...
- `forms`: colección con todos los formularios. Array indexado y asociativo por el atributo `name` del formulario

```
onload = function() {
    document.forms['misdatos']['tele'].onchange = changeTelefono;
    document.forms.misdatos.email.onblur = function() {
        changeEmail(document.forms.misdatos.email);
    }
    document.forms.misdatos.onsubmit = validarFormulario;
}

function changeTelefono() {
    if (!validarTelefono(document.forms.misdatos.tele.value))
        alert("Número de teléfono no válido");
}

function validarFormulario() {
    var err = "";
    if (!validarTelefono(document.forms.misdatos.tele.value))
        err += "El teléfono es incorrecto\n";
    if (!validarEmail(document.forms.misdatos.email.value))
        err += "El email es incorrecto\n";
    ...
}
```



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

La forma recomendada

Mediante `addEventListener`

```
function validarFormulario(e) {
    var err = "";
    if (!validarTelefono(e.target.tele.value))
        err += "El teléfono es incorrecto\n";
    if (!validarEmail(e.target.email.value))
        err += "El email es incorrecto\n";
    if (err != '') {
        alert(err);
        e.preventDefault(); // IMPORTANTE
        return false;
    } else
        return true;
}
```

```
onload = function() {
    document.getElementById('tele').addEventListener('change',changeTelefono);
    document.getElementById('email').addEventListener('blur',changeEmail);
    document.getElementById('fmisdatos').addEventListener('submit',validarFormulario);
}
```

```
function changeEmail(e) {
    if (!validarEmail(e.target.value))
        alert("Email incorrecto");
}
function changeTelefono(e) {
    // this es el target del evento
    if (!validarTelefono(this.value))
        alert("Número de teléfono no válido");
}
```

En este caso, el argumento es un event object

El lenguaje JavaScript: Client-Side JS
Validación de formularios en el cliente

Avisos menos intrusivos
En lugar de alert: incorporar información junto al control validado

```
<div class='formulario'>
  <form id='fmisdatos' name='misdatos' action='{$$_SERVER['SCRIPT_NAME']}' method='POST'>
    <div class='frm_input'> <label for='nombre'>Nombre:</label>
      <input type='text' name='nombre' /> </div>
    <div class='frm_input'> <label for='email'>Email:</label>
      <input id='email' type='text' name='email' />
      <div class='info' id='emailinfo'></div>
    </div>
    <div class='frm_input'> <label for='telefono'>Teléfono:</label>
      <input id='tele' type='text' name='telefono' />
    </div>
    <div class='frm_submit'><input type='submit' name='accion' value='Submit' /></div>
  </form>
</div>
```

El lenguaje JavaScript: Client-Side JS
Validación de formularios en el cliente

Avisos menos intrusivos
En lugar de alert: incorporar información junto al control validado

```
<div class='frm_input'> <label for='email'>Email:</label>
  <input id='email' type='text' name='email' />
  <div class='info' id='emailinfo'></div>
</div>

function changeEmail(e) {
  if (!validarEmail(e.target.value))
    document.getElementById('emailinfo').textContent='Email no válido';
  else
    document.getElementById('emailinfo').textContent='';
}
```

Nombre:	<input type="text"/>
Email:	<input type="text"/> no se ninguno
	Email no válido
Teléfono:	<input type="text"/> muchos números
	Teléfono no válido
<input type="button" value="Submit"/>	



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

Avisos menos intrusivos

En lugar de alert: incorporar información junto al control validado

```
<div class='frm_input'> <label for='telefono'>Teléfono:</label>
  <input id='tele' type='text' name='telefono' />
</div>
```

```
function changeTelefono(e) {
  if (!validarTelefono(this.value)) {
    var mensaje = document.createElement('div');
    mensaje.textContent = 'Teléfono no válido';
    mensaje.className = 'info';
    mensaje.id = 'teleinfo';
    e.target.parentNode.insertBefore(mensaje, e.target.nextSibling);
  } else {
    var mensaje = document.getElementById('teleinfo');
    if (mensaje)
      mensaje.parentNode.removeChild(mensaje);
  }
}
```



El lenguaje JavaScript: Client-Side JS

Validación de formularios en el cliente

Avisos menos intrusivos

En lugar de alert: incorporar información junto al control validado

```
<div class='frm_input'> <label for='email'>Email:</label>
  <input id='email' type='text' name='email' />
  <div class='info' id='emailinfo'>Email no válido</div>
</div>
```

Nombre:	<input type="text"/>
Email:	<input type="text"/> no sé ninguno Email no válido
Teléfono:	<input type="text"/>
<input type="button" value="Submit"/>	

```
.formulario .info {
  margin-left: 100px;
  margin-top: -8px;
  margin-bottom: 10px;
  font-size: 80%;
  color: red;
}

#emailinfo {
  display: none;
}
```

```
function changeEmail(e) {
  if (!validarEmail(e.target.value))
    document.getElementById('emailinfo').style.display = 'block';
  else
    document.getElementById('emailinfo').style.display = 'none';
}
```



UNIVERSIDAD
DE GRANADA

Tecnologías Web

3º Grado en Ingeniería Informática

Programación en el lado del cliente – Client-side JS

1. El lenguaje JavaScript
2. Client-side JavaScript
 1. Conceptos básicos
 2. El objeto window
 3. DOM
 4. Eventos
 5. Validación de formularios
 6. AJAX

»»»

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada © Javier Martínez Baena



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Historia

2002. Microsoft crea objeto XMLHttpRequest para comunicación asíncrona
2005. James Garrett acuña el término AJAX
2006. Estandarizado por W3C
2012. XMLHttpRequest Level 2 (estándar actual)

En la actualidad el estándar lo mantiene WHATWG (Web Hypertext Application Technology Working Group) (<https://whatwg.org>)

En los 90: cada petición al servidor devuelve una página HTML completa

- Usabilidad deficiente
- Ineficiente

AJAX permite:

- Hacer peticiones HTTP asíncronas a un servidor desde la página web
- Con los datos recibidos se puede modificar la página (DOM)

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada © Javier Martínez Baena

El lenguaje JavaScript: Client-Side JS

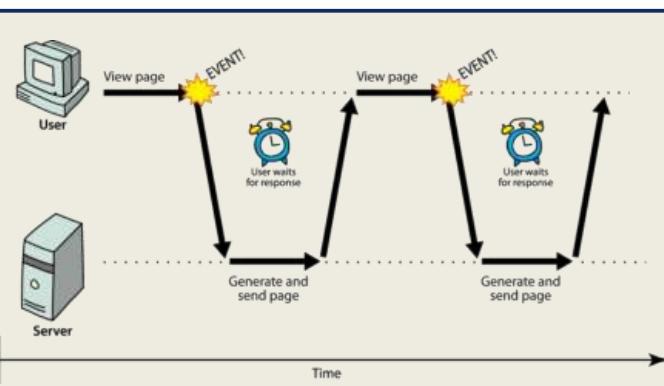
Ajax



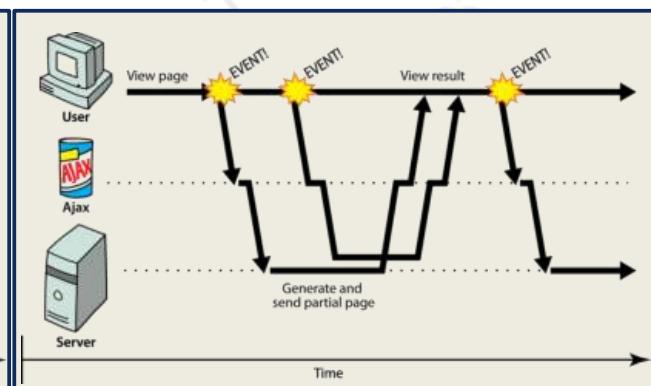
Ajax

Es una combinación de varias tecnologías que permiten crear páginas web interactivas:

- JavaScript / DOM
 - XML (originalmente) / JSON / ... para intercambio de datos
 - Objeto XMLHttpRequest
- Comunicación asíncrona con el servidor HTTP



Comunicación síncrona



Comunicación asíncrona

http://www.webstepbook.com/supplements/slides/ch10-ajax_xml.shtml

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada

© Javier Martínez Baena

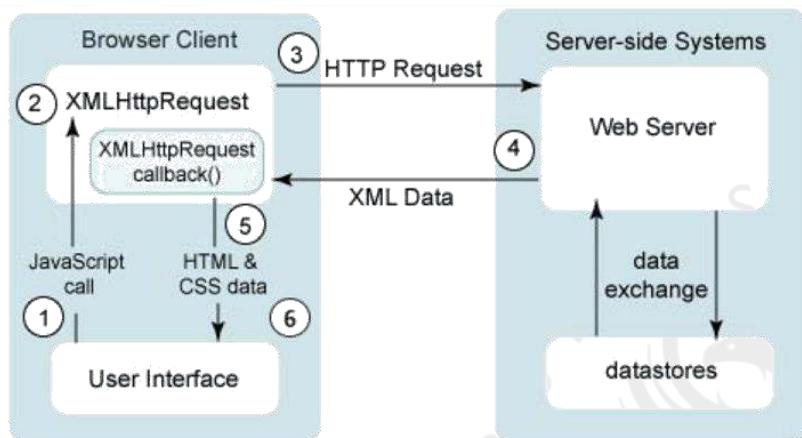
92

El lenguaje JavaScript: Client-Side JS

Ajax



La capa de abstracción de Ajax se gestiona mediante un objeto XMLHttpRequest



1. Se genera necesidad de datos del servidor
2. Se crea un objeto XMLHttpRequest
3. Se hace una petición HTTP al servidor
4. El servidor procesa la petición y devuelve datos
5. El objeto XMLHttpRequest dispara un evento y se ejecuta un manejador
6. El código JS procesa los datos recibidos

<http://www.c-sharpcorner.com/UploadFile/manas1/implementing-ajax-concept-in-Asp-Net-using-xmlhttprequest/>

Departamento de Ciencias de la Computación e Inteligencia Artificial - Universidad de Granada

© Javier Martínez Baena

93



```

onload=function() {
    console.log("Preparando petición");
    var obj = new XMLHttpRequest();
    obj.open("GET", "ajax_resp.php");
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            console.log("Recepción finalizada");
            document.getElementById("par").innerHTML=obj.responseText;
        }
    }
    obj.send(); // Hacer la petición
    console.log("Petición enviada");
}

<body>
<p>Texto recibido:</p>
<p id="par"></p>
</body>

<?php      ajax_resp.php
    echo "Hola, estoy aquí";
?>

```

Texto recibido:
Hola, estoy aquí

▶ GET <http://localhost/tw/js/ajax1.html>
Preparando petición
Petición enviada
▶ GET [XHR http://localhost/tw/js/ajax_resp.php](http://localhost/tw/js/ajax_resp.php)
Recepción finalizada



Interface XMLHttpRequest: realizando la petición

Método para crear la petición:

open(met,url,async,user,pwd) Establece los parámetros de la petición
met: método (GET, POST)
url: URL a la que se hace la petición
async: true/false (defecto: true). Indica si la petición es asíncrona.
user, pwd: Credenciales si fuesen necesarias

Método para añadir cabeceras a la petición HTTP:

setRequestHeader(nombre,valor)

Método para enviar la petición:

send(cuerpo) Envía la petición

El parámetro cuerpo es necesario en el caso de peticiones POST y

contiene el cuerpo de la petición HTTP

Si la petición es síncrona, este método es bloqueante hasta haber recibido la respuesta. En caso contrario finaliza en cuando los datos de la petición han sido enviados



Interface XMLHttpRequest: manejadores de eventos

Manejadores de eventos heredados de XMLHttpRequestEventTarget:

loadstart	Se dispara cuando se comienzan a cargar datos
progress	Se dispara periódicamente mientras dura la petición
abort	Se dispara si la petición es cancelada
error	Se dispara si se produce un error en la petición
load	Se dispara cuando finaliza con éxito la petición HTTP
timeout	Se dispara en caso de timeout
loadend	Se dispara cuando se completa la carga

Manejador de eventos propio:

onreadystatechange	Manejador de eventos. Se ejecuta cuando cambia el valor del atributo readyState
--------------------	---

Atributo readyState Contiene el estado de la petición

0 = UNSENT	Aún no se ha enviado la petición
1 = OPENED	Se ha llamado a open()
2 = HEADERS_RECEIVED	Se han recibido las cabeceras de la respuesta
3 = LOADING	Se está recibiendo la respuesta
4 = DONE	Se ha completado la recepción

<https://xhr.spec.whatwg.org/>



Interface XMLHttpRequest: procesando la respuesta

Atributos:

status	Estado de la respuesta (código HTTP)
statusText	Texto de respuesta HTTP
response	Respuesta (puede ser texto, blob, array, ...)
responseType	Tipo de dato devuelto en la respuesta
responseText	Respuesta en forma de texto (cadena)
responseXML	Document con la respuesta HTML/XML
responseURL	URL a la que se ha hecho la petición

responseType	response
""	Cadena/texto
"arraybuffer"	ArrayBuffer (array de datos binarios raw)
"blob"	Blob (objeto que representa un fichero raw)
"json"	Objeto con datos JSON
"text"	Cadena/texto
"document"	Objeto de tipo Document

<https://xhr.spec.whatwg.org/>

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Interface XMLHttpRequest: procesando la respuesta

Métodos:

`getResponseHeader(nombre)`

Devuelve el valor de la cabecera HTTP indicada

`getAllResponseHeaders()`

Devuelve todas las cabeceras HTTP

 nombre: valor

 nombre: valor

...

`overrideMimeType()`

Independientemente del mimetype enviado por el servidor, con este método se puede forzar

<https://xhr.spec.whatwg.org/>

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

GeoWeb: geografía política

Una web con datos geográficos

Listado | Listado Paginado | Listado Paginado (botones) | Listado Paginado (

Ciudad	Comunidad	
A Coruña (La Coruña)	Galicia	
Albacete	Kastilia-La Mancha	
Alcalá de Henares	Madrid	
Alcorcón	Madrid	
Algeciras	Andalusia	
Almería	Andalusia	
Badajoz	Extremadura	136613
Barakaldo	Baskimaaaaaa	98212
Barcelona	Katalonia	1503451
Bilbao	Baskimaa	357589

La tabla se crea de forma dinámica
Al cargar la página no existe

```
function FORM_listadoCiudades () {
echo <<< HTML
<div id='tabladata' class='listado'>
</div>
HTML;
}
```

PHP

Primera Anterior Siguiente Última

El menú de paginación funcionará
con tecnología AJAX



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Funciones para manipular la tabla

Usamos API de `HTMLTable`:

```
t.rows    t.insertRow()    t.deleteRow()    f.insertCell()
```

```
// Crear la tabla (vacía)
function tblCrearTabla() {
    if (!document.getElementById("tabladatastbl")) {
        var t = document.createElement("TABLE");
        t.id = "tabladatastbl";
        document.getElementById("tablados").appendChild(t);
    }
}

// Eliminar la tabla por completo
function tblBorrarTabla() {
    var tabla = document.getElementById("tabladatastbl");
    if (tabla)
        tabla.parentNode.removeChild(tabla);
}

// Borrar todas las filas de datos (pero no borrar la tabla)
function tblVaciarDatos() {
    var tabla = document.getElementById("tabladatastbl");
    if (tabla)
        while (tabla.rows.length>1)
            tabla.deleteRow(-1); // Borrar la última fila
}
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

```
// Añadir el encabezado de la tabla (y crearla si no existe)
function tblAddHeader() {
    if (!document.getElementById("tabladatashdr")) {
        tblCrearTabla();
        var tabla = document.getElementById("tabladatastbl");
        var fila = tabla.insertRow(0);
        fila.id = "tabladatashdr";
        var c1 = fila.insertCell(0);    c1.innerHTML = "Ciudad";
        var c2 = fila.insertCell(1);    c2.innerHTML = "Comunidad";
        var c3 = fila.insertCell(2);    c3.innerHTML = "Población";
    }
}

// Añadir una nueva fila de datos
// (y crear la tabla si no existe)
function tblAddRow(d1,d2,d3) {
    if (!document.getElementById("tabladatashdr"))
        tblAddHeader();
    var tabla = document.getElementById("tabladatastbl");
    var fila = tabla.insertRow(-1);
    fila.className = "tabladosrow";
    var c1 = fila.insertCell(0);    c1.innerHTML = d1;
    var c2 = fila.insertCell(1);    c2.innerHTML = d2;
    var c3 = fila.insertCell(2);    c3.innerHTML = d3;
}
```

Se insertan atributos para:

- Dar formato con CSS
- Manipular el DOM



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Barra de paginación

Con PHP:

- El rango de ciudades listadas se controlaba con variables GET
- Cada pulsación es una nueva página (completa)

Ahora, con JavaScript:

- Cada pulsación genera un evento (no se carga ninguna página nueva)
- El programa JS debe llevar en cuenta el rango de ciudades



Necesitamos conocer el número total de ciudades para el paginador

```
// Variables globales
var ini = 0;
var items = 10;
var numciudades = -1;

onload=function() {
    getNumCiudades();
}
```

```
function getNumCiudades() {
    var obj = new XMLHttpRequest();
    obj.open("GET", "masDatos.php?maxitems=1");
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            numciudades = ... // Recibir datos
            activarPaginador(); // Activar pulsaciones
            cargarDatos(ini,items); // Rellenar tabla
        }
    }
    obj.send();
}
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Script PHP para responder a peticiones AJAX

- Para devolver el número de ciudades (JSON)
- Para devolver una lista de ciudades (JSON)
- En caso de error devuelve una cadena con un mensaje

```
<?php
require('db.php');
if (isset($_GET['maxitems'])) // Comprobar parámetros de la petición
    $nitems = true;
else if (isset($_GET['ini'])) && is_numeric($_GET['ini']) && $_GET['ini']>=0 &&
    isset($_GET['lon']) && is_numeric($_GET['lon']) && $_GET['lon']>0) {
    $ini = $_GET['ini'];
    $lon = $_GET['lon'];
}
// Procesar petición
if (isset($nitems)) {
} else if (isset($ini)) {
} else
    $error = "Error en parámetros
// Devolver datos ...
if (isset($error))
    echo $error;
else
    echo json_encode($resultado,JSON_PRETTY_PRINT);
?>
```

```
if (isset($nitems)) {
    $db=DB_conexion();
    if ($db) {
        $numciudades=DB_getNumCiudades($db);
        DB_desconexion($db);
        $resultado['numciudades'] = $numciudades;
    } else
        $error = "Error de conexión a la BBDD";
} else if (isset($ini)) {
```



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Script PHP para responder a peticiones AJAX

- Para devolver el número de ciudades (JSON)
- Para devolver una lista de ciudades (JSON)
- En caso de error devuelve una cadena con mensaje

```

<?php
require('db.php');
if (isset($_GET['maxitems'])) // Comprobar parámetros de la petición
    $nitems = true;
else if (isset($_GET['ini'])) && is_numeric($_GET['ini']) && $_GET['ini']>=0 &&
    isset($_GET['lon']) && is_numeric($_GET['lon']) && $_GET['lon']>0 {
    $ini = $_GET['ini'];
    $lon = $_GET['lon'];
}
// Procesar petición
if (isset($nitems)) {
} else if (isset($ini)) {
} else
    $error = "Error en parámetros
// Devolver datos ...
if (isset($error))
    echo $error;
else
    echo json_encode($resultado, JSON_PRETTY_PRINT);
?>
```

```

} else if (isset($ini)) {
    $db=DB_conexion();
    if ($db) {
        $resultado=DB_getCiudades($db,$ini,$lon);
        DB_desconexion($db);
        if ($resultado==false)
            $error = "Error al obtener listado";
    } else
        $error = "Error de conexión a la BBDD";
} else
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo



JSON: JavaScript Object Notation

- Formato de datos (pensado para intercambio)
- Atribución: 2006, Douglas Crockford (él dice que no lo ha inventado)
- Definición en RFC 4627, incluido en ECMAScript 5
- Idea similar a XML pero más sencillo

```
{
  "glossary": {
    "title": "example glossary",
    "GlossDiv": {
      "title": "S",
      "GlossList": {
        "GlossEntry": {
          "ID": "SGML",
          "SortAs": "SGML",
          "GlossTerm": "Standard Generalized Markup Language",
          "Acronym": "SGML",
          "Abbrev": "ISO 8879:1986",
          "GlossDef": {
            "para": "A meta-markup language, used to create ...",
            "GlossSeeAlso": ["GML", "XML"]
          },
          "GlossSee": "markup"
        }
      }
    }
  }
}
```

```

!DOCTYPE glossary PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<glossary><title>example glossary</title>
<GlossDiv><title>S</title>
<GlossList>
  <GlossEntry ID="SGML" SortAs="SGML">
    <GlossTerm>Standard Generalized Markup Language</GlossTerm>
    <Acronym>SGML</Acronym>
    <Abbrev>ISO 8879:1986</Abbrev>
    <GlossDef>
      <para>A meta-markup language, used to create ...</para>
      <GlossSeeAlso OtherTerm="GML">
      <GlossSeeAlso OtherTerm="XML">
    </GlossDef>
    <GlossSee OtherTerm="markup">
  </GlossEntry>
</GlossList>
</GlossDiv>
</glossary>
```

<http://json.org/example.html>



JSON: JavaScript Object Notation

Los datos se pueden estructurar como

- Colección de pares nombre/valor {}
- Lista ordenada de valores []

La notación proviene de JavaScript

```
{
  "nombre": "Menú del día",
  "primeros": ["Sopa", "Gazpacho", "Ensalada"],
  "segundos": [
    { "tipo": "carne", "nombre": "Solomillo", "salsa": ["pimienta", "alioli"]},
    { "tipo": "pescado", "nombre": "Lubina a la sal"}
  ],
  "postre": ["Flan", "Café", "Fruta"]
}
```



JSON: JavaScript Object Notation

Crear un objeto a partir de datos JSON:

- `JSON.parse()`

```
var cad = '{"nombre": "Menú del día",'+
  ' "primeros": ["Sopa", "Gazpacho", "Ensalada"],'+
  ' "segundos": ['+
    { "tipo": "carne", "nombre": "Solomillo", "salsa": ["pimienta", "alioli"]},'+
    { "tipo": "pescado", "nombre": "Lubina a la sal"}'+
  ],'+
  ' "postre": ["Flan", "Café", "Fruta"]'+
}';
```

```
var obj = JSON.parse(cad);
```

```
console.log(cad);
console.log(obj);
```

```
{"nombre": "Menú del día", "primeros": ["Sopa", "Gazpacho", "Ensalada"], "segundos": [ { "tipo": "carne", "nombre": "Solomillo", "salsa": ["pimienta", "alioli"]}, { "tipo": "pescado", "nombre": "Lubina a la sal"}], "postre": ["Flan", "Café", "Fruta"]}
Object { nombre: "Menú del día", primeros: Array[3], segundos: Array[2], postre: Array[3] }
```



JSON: JavaScript Object Notation

Crear una cadena JSON a partir de un objeto:

- `JSON.stringify()`

```
var obj = {
    "nombre": "Menú del día",
    "primeros": ["Sopa", "Gazpacho", "Ensalada"],
    "segundos": [ { "tipo": "carne", "nombre": "Solomillo", "salsa": ["pimienta", "alioli"]},
                  { "tipo": "pescado", "nombre": "Lubina a la sal"}],
    "postre": ["Flan", "Café", "Fruta"] };

var cad = JSON.stringify(obj);
```

```
console.log(cad);
console.log(obj);
```

```
{"nombre": "Menú del día", "primeros": ["Sopa", "Gazpacho", "Ensalada"], "segundos": [{"tipo": "carne", "nombre": "Solomillo", "salsa": ["pimienta", "alioli"]}, {"tipo": "pescado", "nombre": "Lubina a la sal"}], "postre": ["Flan", "Café", "Fruta"]}

Object { nombre: "Menú del día", primeros: Array[3], segundos: Array[2], postre: Array[3] }
```



JSON: JavaScript Object Notation

De forma análoga, en PHP:

`json_encode()`: codifica un objeto PHP en una cadena JSON
`json_decode()`: decodifica una cadena JSON en un objeto PHP

```
<?php
...
// Procesar petición
if (isset($nitems)) {

} else if (isset($ini)) {

} else
    $error = "Error en parámetros GET";

// Devolver datos ...
if (isset($error))
    echo $error;
else
    echo json_encode($resultado, JSON_PRETTY_PRINT);
?>
```

Incluye espacios en blanco para separar items

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo



Procesando datos de número de ciudades

- JSON con datos
- Cadena con mensaje de error

```
function getNumCiudades() {
    var obj = new XMLHttpRequest();
    obj.open("GET", "masDatos.php?maxitems=1");
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            try {
                numciudades = JSON.parse(obj.responseText).numciudades;
                activarPaginador();
                cargarDatos(ini,items);
            } catch (e) {
                // Si el texto devuelto no es JSON
                alert("Error: "+obj.responseText);
            }
        }
    }
    obj.send();
}
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo



Paginador (versión 1)

Programar eventos para cada botón

```
function fprimero() {
    ini = 0;
    items = 10;
    cargarDatos(ini,items);
}

function fultimo() {
    ini = (numciudades-1) - ( (numciudades-1) % items);
    cargarDatos(ini,items);
}

function fanterior() {
    ini = items>ini ? 0 : (ini-items);
    cargarDatos(ini,items);
}

function fsiguiente() {
    var ultima = (numciudades-1) - ( (numciudades-1) % items);
    ini = (ini + items) > numciudades ? ultima : (ini + items);
    cargarDatos(ini,items);
}

function activarPaginador() {
    document.getElementById('pag_pri').addEventListener('click',fprimero);
    document.getElementById('pag_sig').addEventListener('click',fsiguiente);
    document.getElementById('pag_ant').addEventListener('click',fanterior);
    document.getElementById('pag_ult').addEventListener('click',fultimo);
}
```



Paginador (versión 2)

Usando funciones anónimas

```
function activarPaginador() {
    document.getElementById('pag_pri').addEventListener('click', function() {
        ini=0;
        items=10;
        cargarDatos(ini,items);
    });
    document.getElementById('pag_sig').addEventListener('click', function() {
        ini=(ini+items)>numciudades ? ((numciudades-1)-((numciudades-1)%items)) : (ini+items);
        cargarDatos(ini,items);
    });
    document.getElementById('pag_ant').addEventListener('click', function() {
        ini=items>ini ? 0 : (ini-items);
        cargarDatos(ini,items);
    });
    document.getElementById('pag_ult').addEventListener('click', function() {
        ini=(numciudades-1)-((numciudades-1)%items);
        cargarDatos(ini,items);
    });
}
```



cargarDatos: de nuevo necesitamos una petición AJAX

En lugar de repetir el mismo esquema que con getNumCiudades() ...
... crear una función para usar en cualquier petición AJAX

```
function peticionAjax(url,callback) {
    var obj = new XMLHttpRequest();
    obj.open("GET",url);
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            try {
                callback(JSON.parse(obj.responseText));
            } catch (e) {
                alert("Error: "+obj.responseText);
            }
        }
    }
    obj.send();
}

onload=function() {
    peticionAjax("masDatos.php?maxItems=1", function(datos) {
        numciudades = datos.numciudades;
        activarPaginador();
        cargarDatos(ini,items);
    });
}
```



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

cargarDatos: de nuevo necesitamos una petición AJAX

En lugar de repetir el mismo esquema que con getNumCiudades() ...
... crear una función para usar en cualquier petición AJAX

```
function peticionAjax(url,callback) {
    var obj = new XMLHttpRequest();
    obj.open("GET",url);
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            try {
                callback(JSON.parse(obj.responseText));
            } catch (e) {
                alert("Error: "+obj.responseText);
            }
        }
    }
    obj.send();
}
```

```
function cargarDatos(primeros,numero) {
    peticionAjax("masDatos.php?ini="+primeros+"&lon="+numero,
        function(datos) {
            tblVaciarDatos();
            for (var f in datos)
                tblAddRow(datos[f].name, datos[f].district, datos[f].population);
        });
}
```



El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Eliminando las variables globales ...

Crear un objeto en la función y pasar (referencia) al resto de funciones

```
function peticionAjax(url,callback,variables) {
    var obj = new XMLHttpRequest();
    obj.open("GET",url);
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            try {
                callback(JSON.parse(obj.responseText),variables);
            } catch (e) {
                alert("Error: "+obj.responseText);
            }
        }
    };
    obj.send();
}

onload=function() {
    var limites = {ini:0, items: 10, numciudades:-1};
    peticionAjax("masDatos.php?maxitems=1", function(datos,lim) {
        lim.numciudades = datos.numciudades;
        activarPaginador(lim);
        cargarDatos(lim.ini,lim.items);
        },limites);
}
```



Eliminando las variables globales ...

Crear un objeto en la función y pasar (referencia) al resto de funciones

Paginador (Versión 1)

```
function fultimo(limites) {
    limites.ini = (limites.numciudades-1)-((limites.numciudades-1)%limites.items);
    cargarDatos(limites.ini,limites.items);
}

function activarPaginador(limites) {
    document.getElementById('pag_pri').addEventListener('click',
        function() { fprimero(limites); } );
    document.getElementById('pag_sig').addEventListener('click',
        function() { fsiguiente(limites); } );
    document.getElementById('pag_ant').addEventListener('click',
        function() { fanterior(limites); } );
    document.getElementById('pag_ult').addEventListener('click',
        function() { fultimo(limites); } );
}
```



Eliminando las variables globales ...

Crear un objeto en la función y pasar (referencia) al resto de funciones

Paginador (Versión 2)

```
function activarPaginador(limites) {
    document.getElementById('pag_pri').addEventListener('click',
        function() { limites.ini=0; limites.items=7;
                    cargarDatos(limites.ini,limites.items); } );
    document.getElementById('pag_sig').addEventListener('click',
        function() { limites.ini=(limites.ini+limites.items)>limites.numciudades ?
                    ((limites.numciudades-1)-((limites.numciudades-1)%limites.items)) :
                    (limites.ini+limites.items);
                    cargarDatos(limites.ini,limites.items); } );
    document.getElementById('pag_ant').addEventListener('click',
        function() { limites.ini=limites.items>limites.ini ? 0 : (limites.ini-limites.items);
                    cargarDatos(limites.ini,limites.items); } );
    document.getElementById('pag_ult').addEventListener('click',
        function() { limites.ini=(limites.numciudades-1)-((limites.numciudades-1)%limites.items);
                    cargarDatos(limites.ini,limites.items); } );
}
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Realizando la petición POST con Ajax ...

```
function peticionAjaxPOST(url,querystr,callback,variables) {
    var obj = new XMLHttpRequest();
    obj.open("POST",url);
    obj.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    obj.onreadystatechange = function() {
        if (obj.readyState==4 && obj.status==200) {
            try {
                callback(JSON.parse(obj.responseText),variables);
            } catch (e) {
                alert("Error: "+obj.responseText);
            }
        }
    }
    obj.send(querystr);
}
```

```
peticionAjaxPOST("masDatos_post.php", "ini="+primero+"&lon="+numero, function(datos) { ... });
```

El lenguaje JavaScript: Client-Side JS

Ajax: ejemplo

Realizando una petición síncrona con Ajax ...

El método send() es bloqueante

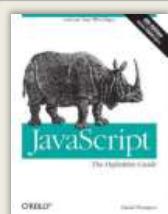
```
var request = new XMLHttpRequest();

request.open('GET', '/bar/foo.txt', false); // 'false' makes the request synchronous
request.send(null);

if (request.status === 200) {
    console.log(request.responseText);
}
```

El lenguaje JavaScript: Client-Side JS

Bibliografía



David Flanagan
JavaScript: The definitive guide (6th ed)
Addison Wesley. 2011

<http://effectivejs.com/>



Tim Wright
Learning JavaScript
Addison Wesley. 2013

<http://learningjsbook.com>



Ved Antani
Mastering JavaScript
Packt. 2016

<http://javascriptissexy.com/>

Artículos sobre cuestiones puntuales

<http://javascript.crockford.com/>

Web del autor. Varios artículos interesantes

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>