

JS en la Web. JQuery.

Entorno: el navegador. Librerías

DOM. Eventos

Interfaz de usuario: Formularios. Imágenes

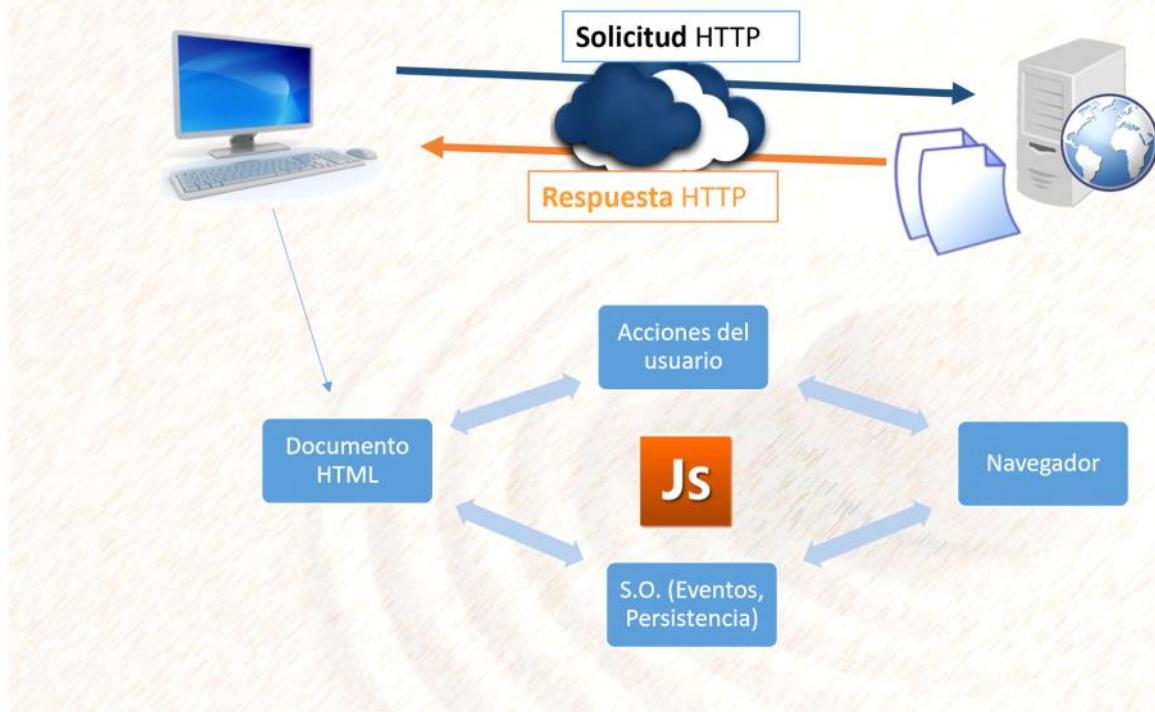
Entorno Web: Navegadores Librerías: JQuery

Navegadores. Plugins. Librerías JavaScript

Diferencias entre navegadores. Polyfills

Navegadores y JS: BOM

Modelo cliente-servidor en la Web



Funcionalidades básicas

Js

Interactuar con el **navegador (BOM)**.

Seleccionar, utilizar y modificar sus propiedades.

Interactuar con el **documento (DOM)**.

Seleccionar, añadir, modificar y borrar nodos.

Seleccionar conjuntos de nodos y aplicarles estilos CSS.

Generar nuevo contenido.

Interactuar con el usuario y el SO mediante el sistema de **eventos**: capturar acciones de ratón y de teclado, junto con el paso del tiempo y procesarlos adecuadamente.

Otras funcionalidades

Js

un sistema unificado de comunicación con el servidor de manera asíncrona (**AJAX**),

un sistema para poder trabajar con **formularios** de datos (un modo de interacción con el DOM)

mecanismos de **almacenamiento persistente** de la información con el nivel necesario de seguridad

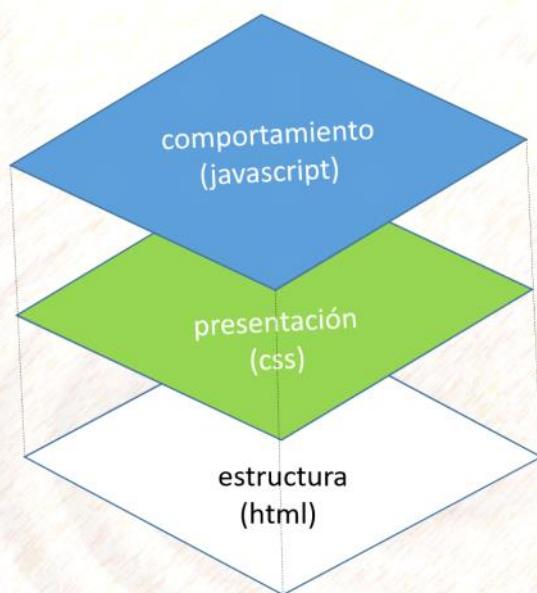
una especificación de componentes de software de tipo **widget** que nos permitan expandir las funcionalidades propias del navegador.

AC1

Mejora progresiva (progressive enhancement)

Modelo de diseño y
desarrollo de los sitios
Web

Completa
diferenciación
en tres capas



Learning JavaScript: A Hands-On Guide to the Fundamentals of Modern JavaScript
Tim Wright. Addison-Wesley, 2012

AC1

AC23

Mejora progresiva: 1^a etapa

Como técnica
de desarrollo

consiste en **empezar** por generar un
código genérico que funcione en todos
los navegadores,

Se recupera la idea inicial de la
Web, tal como fue concebida
por Tim Berners-Lee.,

el marcado HTML refleja
jerárquicamente la
estructura de los
elementos

Se recogen los nuevos
planteamientos de HTML5

El código acumula el
mayor valor semántico
posible

Se pone además un énfasis
prioritario en el tema de la
accesibilidad

Iniciativas como WAI-ARIA

CSS3 y Javascript Avanzado
Jordi Collell Puig. Universitat Oberta de Catalunya, 2013

Learning JavaScript: A Hands-On Guide to the Fundamentals of Modern JavaScript
Tim Wright. Addison-Wesley, 2012

Mejora progresiva: presentación

Segundo paso en la “mejora progresiva”: ir **introduciendo mejoras en la presentación válidas para** para navegadores más modernos

- Para ello se **emplea CSS**: gracias a su diseño en cascada, los navegadores antiguos simplemente ignoraran las propiedades que no conocen quedándose con propiedades más antiguas o incluso con el diseño inicial.

Logramos un control total óptimo del aspecto, puesto que a mejores prestaciones del navegador, mejor visualización.

En este modelo, los estilos CSS se aplican **embebidos** en la cabecera o en **ficheros externos** referenciados mediante la etiqueta <link>
No se recomienda nunca el uso de estilos “en línea”

Mejora progresiva: comportamiento

- Una vez que todo funciona sin **JavaScript**, se pueden añadir mediante el código **mejoras en la funcionalidad** del sitio Web

Nuevamente se puede incorporar el código de tres formas

- JavaScript en línea
- JavaScript embebido
- **JavaScript embebido en <head>**
- JavaScript en ficheros externos

Sólo se recomienda el uso de 2 de estas formas.

Facilitan lo que se conoce como **JS no obstructivo o no intrusivo** (*Unobtrusive JavaScript*)

- Separación entre las tres capas, aislando la funcionalidad JavaScript
- Uso de buenas prácticas para evitar los problemas de incompatibilidad

JavaScript: Formas poco recomendables

Js

JavaScript en-línea en atributos “Evento” en cualquier etiqueta HTML

```
<p onclick="alert('Un mensaje de prueba')">párrafo</p>
```

JavaScript embebido en cualquier parte del documento

```
<script type="text/javascript"> Código JavaScript </script>
```

```
<p>
  <script>
    alert('Un mensaje de prueba')
  </script>
</p>
```

Ejemplos de JS intrusivo, que se deben evitar

JS en un documento HTML: objetivos

Js

- Situar todo el código en un bloque, exportable a un fichero externo.
- Recoger las acciones del usuario sobre cualquier elemento del HTML (sin que en él haya código)
- Afectar cualquier parte del documento que nos interese (sin que en ella haya código)

Para ello es necesario emplear diversas técnicas

- Uso del DOM para poder acceder a los elementos de la estructura
- La definición de manejadores de eventos para poder responder a estos
- La agrupación del código en funciones
- La espera hasta la carga completa de la estructura antes de comenzar la ejecución de los scripts

Eventos: formas de gestionarlos (1)

Js

Manejadores de eventos (Event handlers)

Para responder al onclick de un elemento cuyo id es “clicAqui”

- creamos la función funcionOnclck ()
- asociamos la función manejadora al evento en el elemento a nivel del DOM

element.onevent=functionName

```
document.getElementById("clicAqui ").onclick = funcionOnclck;
```

sin paréntesis, queremos la función, no su resultado

Eventos: *formas de gestionarlos (1)*

Js

Manejadores de eventos (Event handlers)

Para responder al onclick de un elemento cuyo id es “clicAqui”

- creamos la función funcionOnclick ()
- asociamos la función manejadora al evento en el elemento a nivel del DOM

element.onevent=functionName

```
document.getElementById("clicAqui ").onclick = funcionOnclick;
```

sin paréntesis, queremos la función, no su resultado

Eventos: formas de gestionarlos (2)

Js

Escucha de eventos (Event listeners)

podemos tener uno o más manejadores “a la escucha”
(listeners) de un evento determinado:

asociar y desasociar manejadores de
eventos en DOM 2 estándar

→ addEventListener()
removeEventListener()

```
element.addEventListener(event,functionName, false)
```

```
document.getElementById("clicAqui ").  
addEventListener('click', funcionOnclck,false);
```

El evento “carga de la página” (1)

Para que el código pueda incluir referencias a los elementos de la estructura (por medio del DOM) es necesario que estos existan antes de que se procese el script

Este procesamiento queda “en suspenso” hasta que se produce el evento **window.onload**, que corresponde a la carga completa de la estructura definida mediante HTML

```
window.onload = function() {  
    document.getElementById("valorId").onclick  
        = miFuncion;  
}
```

Asocio al evento onload una función anónima, en la que incluyo aqueyos procesos que deben llevarse a cabo una vez que se ha cargado la página

El evento “carga de la página” (2)

La segunda forma de que el script espere a que la página se cargue por completo es utilizar la escucha del evento correspondiente, denominado DOMContentLoaded

```
function inicioScript () {  
  
    document.getElementById("btn_1").addEventListener(  
        "click", btnEventListener, false);  
}  
  
document.addEventListener("DOMContentLoaded",  
    inicioScript, false);
```

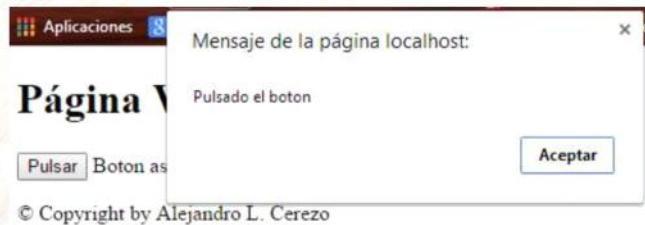
La “escucha” asocia el evento DOMContentLoaded con una función (o con una función anónima), en la que, cuando se haya cargado la página, se definirán las escuchas de todos los eventos de la página

Ejercicio 1

jueves, 25 de mayo de 2017 20:01

Ejercicio 1

Creamos una página en la que un botón permita lanzar un mensaje emergente mediante Alert()



Utilizamos eventos y funciones de modo que se cumplan los principios de JS no intrusivo, independizando todo el código JS. Reproducimos el proceso mediante event handler y event listener

AC24

Aplicaciones Web frente a páginas

Evolución de los navegadores como S.O

S.O. de escritorio	Navegador
Interfaz; Organizar iconos para acceder a las aplicaciones	Interfaz; Organizar los marcadores
Múltiples aplicaciones en ventanas	Múltiples pestañas
APIS de bajo nivel: red, gráficos, ficheros	Nuevas APIS HTML5: red, gráficos, ficheros

La nueva
funcionalidad de
los navegadores

Aplicaciones Web
como evolución de
las páginas Web

Aplicaciones Web

Aplicación
Web

- parte de una página web
- utiliza JS para acceder a todos los servicios avanzados que puede proporcionar el navegador

- Se asume la necesidad de los más recientes navegadores
- No existen requisitos de retro compatibilidad
- Desaparece cualquier limitación a utilizar los más recientes recursos del lenguaje, como las últimas APIs de HTML5

- red: XMLHttpRequest Level 2
- gráficos: Canvas / SVG
- ficheros: Web Storage

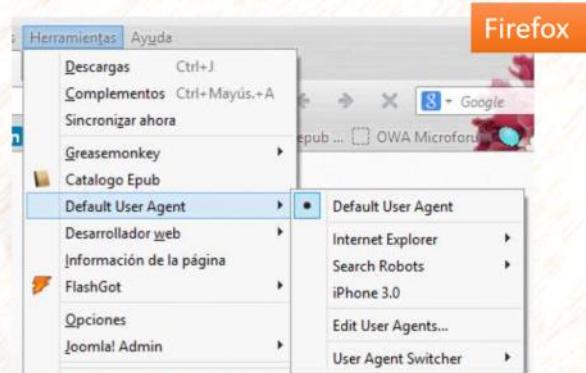
JavaScript: The Definitive Guide, 6th ed. David Flanagan. O'Reilly Media, 2011 pg 310

Navegadores y agentes de usuario Js

Agente de usuario: aplicación que funciona como cliente en el protocolo de la Web (http)

Un servidor Web puede responder específicamente según el agente y a su vez los agentes pueden identificarse como otro agente de usuario, lo que se conoce como *user-agent spoofing*.

- navegadores web “estándar”
- web crawler y buscadores
- navegadores de dispositivos móviles
- lectores de pantalla
- navegadores en Braille



Particularidades y navegadores

Comentarios condicionales en IE. Funcionan desde IE5 hasta IE9 (incluidos), permitiendo insertar estilos de aplicación exclusiva en estas versiones de IE. Estas instrucciones sólo son válidas en ficheros HTML, no pudiendo incorporarse en los ficheros independientes de estilos (CSS).

<http://www.quirksmode.org/css/condcom.html>

```
<!--[if IE 6]>
Special instructions for IE 6 here
<![endif]-->
```

Comprobación de la versión del navegador. Dentro del código JavaScript existen varias técnicas posibles.

- "browser sniffing"
- "object detection"

Identificación del agente de usuario

Js

"*browser sniffing*"

consultar los atributos appName y appVersion del **objeto navigator**

```
document.write("You are running " + navigator.appName+"<br>");  
document.write("Its version is " + navigator.appVersion);
```

En concreto, la aplicación más concreta de este método es reconocer la presencia de MSIE, especialmente su versión 6

```
// Detectar si el navegador es Internet Explorer  
var ie = navigator.userAgent.toLowerCase().indexOf('msie')!=-1;  
// Detectar si el navegador es Internet Explorer 6  
var ie = navigator.appName.indexOf('Internet Explorer') != -1  
&& browserVersion.indexOf('6') != -1;
```

Identificación de las capacidades del agente de usuario

Js

"*object detection*"

se renuncia a determinar cual es el agente usuario
basta comprobar si soporta determinado objeto

```
if(document.all)
{
    // MSIE DOM
}
else if (document.getElementById)
{
    // W3C DOM
    // modern browser: MOZ, Chrome, Safari, Opera or IE");
}
```

[Desarrollos\JavaScript\Extras\Detectar_Browser.html](#)

Identificar características: Modernizr

The screenshot shows the official website for Modernizr. At the top, there's a navigation bar with links for DOWNLOAD, DOCUMENTATION, RESOURCES, and NEWS. Below the navigation, a quote from Bruce Bowman is displayed: "An indispensable tool." — Bruce Bowman, sr. product manager, Edge Tools & Services. The main content area features a large orange box containing text about Modernizr's purpose and capabilities. To the right of this box is a section titled "Download Modernizr 2.7.1" with options for "View documentation", "DEVELOPMENT" (Uncompressed, 42 kB), and "PRODUCTION" (CompressYourData). Below these sections is a "Get started with Modernizr" section with a note about best practices and progressive enhancement.

librería de JavaScript que detecta cuáles características soporta un explorador. Actualmente revisa 18 características de CSS3 y más de 40 relacionadas con HTML5, examinando cómo responde el explorador a una serie de pruebas.

How it works

Download Modernizr 2.7.1

View documentation

DEVELOPMENT Uncompressed, 42 kB

PRODUCTION CompressYourData

Get started with Modernizr

While Modernizr gives you finer control over the experience through JavaScript-driven feature detection, it is important to continue to use best practices throughout your development process. Use progressive enhancement wherever you can, and don't sacrifice accessibility for convenience or performance.

<http://modernizr.com/>

Compensar diferencias: Polyfills

Código descargable que proporciona una serie de facilidades que no forman parte del diseño de un determinado navegador

e.g. las características de HTML5 no soportadas en IE en sus versiones 8 o 9

- html5shiv
- -prefix-free
- Selectivizr
- Flexie
- CSS3 PIE
- JSON 2
- es5-shim
- FlashCanvas
- MediaElement.js
- Webshims Lib

Polyfills: HTML5 shiv

Creamos los elementos nuevos con Javascript:

```
document.createElement('nav');  
document.createElement('header');  
document.createElement('footer');  
document.createElement('article');  
document.createElement('section');
```

Y... ¡magia! Nuestra flamante página ya se puede ver en todos los navegadores :-)

No hace falta que los creemos cada vez, solo hay que cargar este script de Remy Sharp (o Modernizr)

```
<!--[if lt IE 9]>  
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>  
<![endif]-->
```

polyfills que emulan características relacionadas a
HTML5 y tecnologías asociadas,

Bibliotecas o frameworks

Js

- **jQuery**
- **Underscore.js**
- **React.js**
- **MooTools**
- Prototype
- YUI
- **AngularJS**
- **BackboneJS**
- **Ember.js**
- SAP - OpenUI5
- **Google Web Toolkit** (de Java a JS)
- AccDC
- Ample SDK
- Atoms.js
- DHTMLX
- Dojo
- **Echo3**
- Enyo
- **Ext JS**
- **Handlebars**
- **Kendo UI**
- **Knockout..js**
- **Meteor**
- PhoneJS
- Pyjamas
- qooxdoo
- **Rialto**
- SmartClient & SmartGWT
- **Socket.IO**
- SproutCore
- Wakanda
- ZK
- Webix
- **D3.js** - Kinetic.js

<http://www.infoq.com/research/javascript-frameworks-2015>

Bibliotecas

Js

Orientadas al interfaz de las páginas web

jQuery, creada por John Resig, la más utilizada, casi un estándar “de facto”



prototype, de las primeras bibliotecas utilizadas, entre otros por Apple
<http://prototypejs.org/>



Mootools
<http://mootools.net/>



Su objetivo es conseguir una **API** (*application programming interface*) **común a los diferentes navegadores**.

Frameworks

Js

YUI, escrita para Yahoo
<http://yuilibrary.com/>



BackboneJS
<http://backbonejs.org/>



AngularJS
<https://angularjs.org/>



Ember.js
<http://emberjs.com/>



Backbone JS



- Framework código abierto (*open source*) en JS
- Orientado a la creación de aplicaciones (interface y datos complejos)
- Basado en el patrón MVC (Modelo-Vista-Controlador)
- Muy flexible y muy ligero
- Muy bien documentado
- Buen rendimiento
- Muy abstracto. No manipula el DOM y delega los aspectos del interfaz (e.g. en otros *frameworks* como **JQuery**)
- En operaciones rutinarias emplea la utilidades agrupadas en la librería **UnderscoreJS**

EmberJS – muy rígido. Muchas funcionalidades en pocas líneas pero sin apenas flexibilidad

AngularJS – flexible, muchas posibilidades. Más difícil de aprender y con menos rendimiento. Trabajo directo sobre entidades del DOM

Librería (Biblioteca) JQuery



creada inicialmente por John Resig y presentada el 14 de enero de 2006; es software libre y de código abierto, posee una doble licencia, la del MIT y la Licencia Pública General de GNU v2,



<http://jquery.com/>

- simplificar la manera de interactuar con los documentos HTML y con los estilos CSS
- homogenizar el funcionamiento de los distintos navegadores
- manipular el árbol DOM,
- manejar eventos,
- desarrollar animaciones y
- agregar interacción con la técnica AJAX

Instalación



Se trata de un fichero externo, con un conjunto de funciones: se enlaza como cualquier otro script

desde una **red de distribución de contenido** (CDN, *content distribution network*), como Microsoft, jQuery o Google:

- <script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.6.3.min.js">
 </script>
- <script src="http://code.jquery.com/jquery-1.6.3.min.js">
 </script>
- <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.3/jquery.min.js">
 </script>

desde el servidor **web de la página** que utiliza la librería

- <script src="jquery.xxx.js"></script>

Inicio del script (1)



```
<script src="js/jquery-1.6.3.min.js"></script>

<script>
$(document).ready(function() {
// conjunto de instrucciones en JavaScript y JQuery
});
</script>
```

`$(document).ready(` 

función propia que espera
hasta que se carga el HTML
de la página

realmente el método ready() del
objeto \$(document)

`function() { }`

función anónima en la que incluimos todo
el código a ejecutar

Inicio del script (1)



```
<script src="js/jquery-1.6.3.min.js"></script>

<script>
$(function() {
// conjunto de instrucciones en JavaScript y JQuery
});
</script>
```

\$(function() { })

atajo a la expresión anterior

nuevamente utilizamos, esta vez de forma implícita, el método ready() del objeto \$(document) para incluir una función anónima en la que a su vez incluimos todo el código a ejecutar

Respuesta a eventos



```
$(`selector`).evento(función());
```

La respuesta puede ser una de las funciones predefinidas, como las que conocemos, o una respuesta más compleja, en forma de función anónima

```
$(`selector`).evento(function() {
  //código de la función anónima
}); // fin del manejador de eventos
```

```
$('#menu').mouseover(function() {
  $('#submenu').show();
}); // end mouseover
```

La función on() [bind]



1.7

La función .on() permite otra forma de asignación más flexible de un determinado evento a un elemento (i.e. selector)

```
$( 'selector' ).on( evento(s) ,  
                    [datos] ,  
                    función);
```

```
$( 'selector' ).on('click',respuextaClick(oEvento));
```

La función .off() permite deshacer la asignación de un determinado evento a un elemento (i.e. selector)

```
$( 'selector' ).off('click');
```

JQuery: procedimiento general



Los procesos en JQuery incluyen tres etapas

- seleccionar un **elemento HTML**
- asociarle un **evento** (opcionalmente)
- ejecutar una **acción** que afecta al elemento seleccionado, en su caso en respuesta al evento correspondiente

Los acciones posibles incluyen:

- cambiar una propiedad
- añadir nuevo contenido
- eliminar un elemento
- extraer información de un elemento
- añadir o eliminar atributos

AC1

Navegadores: Comportamiento Js

Browser Object Model (BOM)

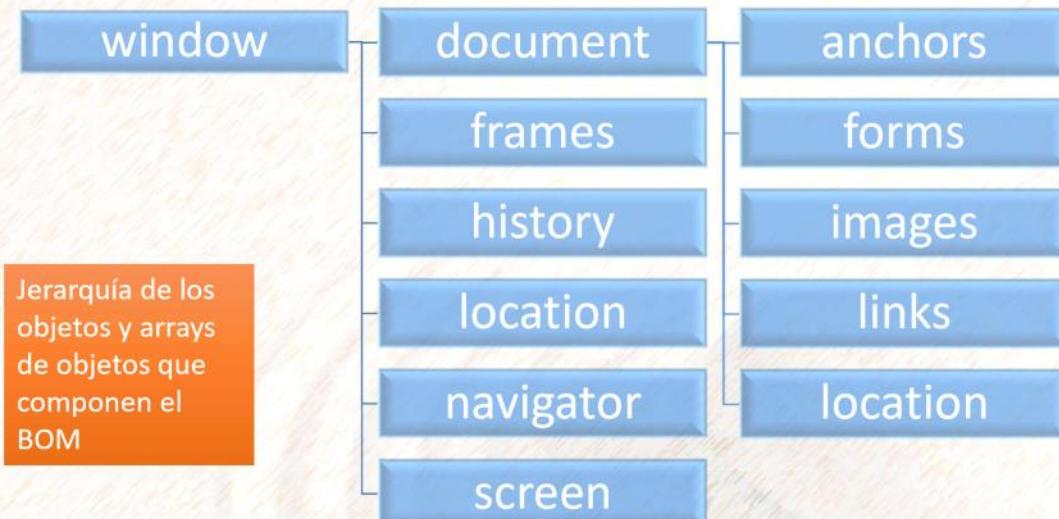
- acceder y modificar las propiedades de las ventanas del propio navegador.
- introducido en las versiones 3.0 de los navegadores Internet Explorer y Netscape Navigator
- no existe ninguna estandarización “oficial”

Algunos de los elementos que forman el BOM son:

- Crear, mover, redimensionar y cerrar ventanas de navegador.
- Obtener información sobre el propio navegador.
- Propiedades de la página actual y de la pantalla del usuario.
- Gestión de cookies.
- Objetos ActiveX en Internet Explorer

Objetos del BOM

Js



Seguridad Web (1)

Same Origin Policy (SOP)

un programa JavaScript sólo puede acceder elementos con el mismo origen que aquel en el que ha sido ejecutado él

(restricción de pertenecer al “mismo origen”)



Esto hace que un iFrame se ejecute en una Sand-Box respecto a la página que lo incluye, a la que no tiene acceso

Protege del Cross-Site Scripting (XSS) empleado para robar *cookies* de sesión, inutilizar navegadores, injectar código que haga peticiones a un determinado servidor

Seguridad Web (2)

Same Origin Policy (SOP)

NO IMPLICA

ninguna dificultad al **usar scripts / css procedentes de otro dominio.**

Estos se invocan desde nuestra página Web, de la que heredan su origen de cara a la SOP

Es decir, la política del mismo origen

- se aplica al contenido del navegador web, al espacio de datos del cliente web, y
- no a las peticiones que se hagan desde él.

Recursos cross-site

- Código javascript con <script src="..."></script>
- CSS con <link rel="stylesheet" href="...">
- Imágenes con
- ficheros multimedia con <video> o <audio>
- fonts con @font-face
- plug-ins con <object>, <embed> o <applet>
- cualquier contenido con <iframe>
- las peticiones dinámicas (AJAX)

Categorías en
las Interacciones

- escritura (generalmente permitida)
- embebimiento (generalmente permitido)
- lectura (generalmente no permitida o filtrada por el elemento embebido)

AC3

Js

Objeto window

representa la ventana completa del navegador.

- permite mover, redimensionar y manipular la ventana actual del navegador,
- abrir y cerrar nuevas ventanas de navegador.

Métodos

alert()	open()	moveBy()	setInterval()
confirm()	close()	moveTo()	setTimeout()
prompt()	createPopup()	resizeBy()	clearInterval()
print()	focus()	resizeTo()	clearTimeout()
	blur()	scrollBy()	
	stop()	scrollTo()	

Objeto window: nuevas ventanas

 JS

open() permite abrir una nueva ventana de navegador, devolviendo un objeto window, que permite manejárla

```
var oVentana= window.open("URL","destino (e.g. _blank=",lista de opciones);
```

ejemplo: "toolbar=yes, scrollbars=yes,
resizable=yes, top=500, left=500,
width=400, height=400"

createPopup() permite abrir una ventana en IE

close() permite cerrar las ventanas de navegador abiertas previamente, al tener referencia al objeto correspondiente

```
oVentana.close()
```

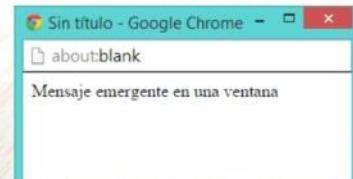
Ejercicio 3

jueves, 25 de mayo de 2017 20:11

Ejercicio 3

Utilizando distintos botones para lanzarlas, creamos distintas funciones que permitan abrir y cerrar una ventana.

Abrir/Cerrar una ventana vacía



Comprobamos el uso de los métodos open() y close() del objeto window

objeto window: manipulación

focus() blur()	entrega / quita el foco a una ventana
moveBy(x, y) moveTo(x, y) resizeBy(x, y) resizeTo(x, y)	BY indica desplazamiento / incremento TO indica valores finales de coordenadas / tamaño
stop()	detiene la carga de una

- navegadores son cada vez menos permisivos con estos métodos
- una aplicación nunca debe suponer que este tipo de funciones están disponibles

Objeto window: obtener datos

Js

	WK, MOZ, OP, IE(9-10)	IE (5-8)
posición de la ventana	window.screenX window.screenY	window.screenLeft window.screenTop
tamaño de <i>viewport</i> o zona visible de la ventana (sin barra de estado ni menús)	window.innerWidth window.innerHeight	document.body.offsetWidth document.body.offsetHeight
tamaño total de la ventana	window.outerWidth window.outerHeight	(no disponible)

AC5

Js

Objeto screen: propiedades

información sobre la pantalla del usuario

availHeight	Altura de pantalla disponible para las ventanas
availWidth	Anchura de pantalla disponible para las ventanas
colorDepth	Profundidad de color de la pantalla (32 bits normalmente)
height	Altura total de la pantalla en píxel
width	Anchura total de la pantalla en píxel

La altura/anchura de pantalla disponible para las ventanas es menor que la altura/anchura total de la pantalla, ya que se tiene en cuenta el tamaño de los elementos del sistema operativo como por ejemplo la barra de tareas y los bordes de las ventanas del navegador.

Ejercicio (consola 1)

`screen`.

Mediante la consola de JavaScript comprobamos como podemos consultar las propiedades de la pantalla a través de `screen`.

Objetivo: conocer el contenido del objeto `screen` del BOM

Ejercicio 4

jueves, 25 de mayo de 2017 20:11

Ejercicio 4

Utilizando distintos botones para lanzarlas, creamos distintas funciones que permitan abrir una ventana con una imagen y redimensionarla al tamaño de la pantalla o al de la propia imagen

Abrir/Cerrar una ventana con una imagen

Ajustar el tamaño de la ventana con la imagen



Comprobamos el uso de los métodos y propiedades de los objetos window y screen

AC4

Objeto window: gestión del tiempo

Js

setTimeout()

ejecutar una función al transcurrir un determinado periodo de tiempo; i.e. programar eventos temporizados
setTimeout(nombre_función, milisegundos)

```
función muestraMensaje() {  
    alert("Han transcurrido 3 segundos desde que me  
    programaron");}  
var id = setTimeout(muestraMensaje, 3000)
```

setInterval()

permite establecer la ejecución periódica y repetitiva de una función; i.e. programar eventos periódicos
setInterval(nombre_función, milisegundos)

clearTimeout() clearInterval()

ambas devuelven un identificador: hace posible interrumpir la cuenta atrás o la repetición
clearTimeout(id) / clearInterval(id)

Ejercicio 5

jueves, 25 de mayo de 2017 20:09

Ejercicio 5a

Cronómetro.

Creamos un cronómetro asociado a 2 botones:
- Iniciar/ parar
- Inicializar a 0

3.3 segundos

[arrancar/parar](#) [inicializar](#)

Objetivo: conocer el uso de los métodos del objeto Window relacionados con el tiempo

Nota: comentar el uso de textContent / innerHTML.

Ejercicio 5b

Cronómetro.

Recuperamos el cronómetro asociado a 2 botones:
- Iniciar/ parar
- Inicializar a 0

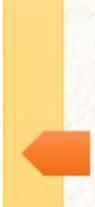
3.3 segundos

[arrancar/parar](#) [inicializar](#)

Lo implementamos empleando OOP completa (incluyendo prototipos), creamos 2 cronómetros (2 objetos distintos) que funcionen independientemente con sus respectivos botones

Objetivo: conocer el uso de los métodos del objeto Window relacionados con el tiempo

Nota: comentar el uso de textContent / innerHTML.



relacionados con el tiempo

Nota: comentar el uso de textContent / innerHTML.



AC5

Objeto navigator: propiedades (1)

Js

- permite obtener información sobre el propio navegador.
- de los primeros objetos que incluyó el BOM y de los menos estandarizados
- En IE, el objeto navigator también se puede acceder a través del objeto clientInformation.

	Tipo	
appCodeName	String	nombre del navegador (normalmente es Mozilla)
appName	String	nombre oficial del navegador
appVersion	String	versión del navegador
appMinorVersion	String	versión del navegador (sólo en IE)
userAgent	String	identificación del navegador ante los servidores
product	String	nombre del producto (normalmente, es Gecko)
productSub	String	información adicional sobre el producto

Objeto navigator: propiedades (2)

Js

	Tipo	
browserLanguage	String	el idioma del navegador
language	String	el idioma del navegador
systemLanguage	String	idioma del sistema operativo (sólo IE)
userLanguage	String	idioma del sistema operativo (sólo IE)
cpuClass	String	tipo de CPU del usuario ("x86", "68K", "PPC", "Alpha", "Other") (sólo IE)
oscpu	String	el sistema operativo o la CPU (sólo Firefox)
platform	String	plataforma sobre la que se ejecuta el navegador
userProfile	String	perfil del usuario (sólo IE)
securityPolicy	String	solo Firefox

Objeto navigator: propiedades (3)

Js

	Tipo	
mimeTypes	Array	tipos MIME registrados por el navegador
plugins	Array	lista de plugins instalados en el navegador
cookieEnabled	Bool	indica si las cookies están habilitadas
javaEnabled()	Bool	indica si JavaScript está habilitado
onLine	Bool	indica si el navegador está conectado a Internet
preference()		Método empleado para establecer preferencias en el navegador (sólo Firefox)

Ejercicio (consola 2)

navigator.

Mediante la consola de JavaScript comprobamos como podemos consultar las propiedades del navegador a través de *navigator*.

Objetivo: conocer el contenido del objeto *navigator* del BOM

AC5

Objeto document

Js

pertenece tanto al DOM como al BOM, donde proporciona información sobre la propia página HTML.

lastModified	La fecha de la última modificación de la página
referrer	La URL desde la que se accedió a la página (es decir, la página anterior en el array history)
title	El texto de la etiqueta <title>
URL	La URL de la página actual del navegador

Objeto document: contenido

Js

contiene varios arrays con información sobre algunos elementos de la página::

Array	
anchors	"anclas" de la página (los enlaces de tipo <code></code> , ya en desuso)
applets	applets (componentes de aplicación, generalmente Java, que se ejecuta en el contexto del navegador)
embeds	objetos embebidos en la página mediante la etiqueta <code><embed></code>
forms	formularios de la página
images	imágenes de la página
links	enlaces de la página (los elementos de tipo <code></code>)

Objeto document: acceso



para acceder a los elementos de los arrays incluidos en el objeto *document* puede utilizarse tanto el índice como el nombre del elemento correspondiente (indicado con el atributo name)

para ``
usaríamos

`document.images[0]` o
`document.images["logotipo"]`

estos mismos elementos pueden ser accedidos por cualquiera de los métodos disponibles en el DOM

`document.getElementsByTagName("img")`
`document.getElementById("logotipo")`

Ejercicio (consola 3)

document.

Mediante la consola de JavaScript comprobamos como podemos consultar las propiedades del documento y acceder a los elementos referenciados a través de *document*.

Objetivo: conocer el contenido del objeto *document* del BOM

AC5

Objeto location: propiedades

Js

Debido a la falta de estandarización, *location* es una propiedad tanto del objeto *window* como del objeto *document*.

Representa la URL de la página HTML que se muestra en el navegador y proporciona varias propiedades útiles para el manejo de la URL:



port (vacio cuando no se indica y se utiliza el 80)

search: contenido que se encuentra tras el símbolo ?

Objeto location: métodos



assign() Equivalente a location.href = "http://www.ejemplo.com"

```
location.assign("http://www.ejemplo.com");
```

replace() Similar a assign(), salvo que se borra la página actual del array history del navegador

```
location.replace("http://www.ejemplo.com")
```

reload() Recarga la página.
Si el argumento es true, se carga la página desde el servidor. Si es false, se carga desde la cache del navegador

```
location.reload(true);
```

Ejercicio (consola 4)

location.

Mediante la consola de JavaScript comprobamos como podemos consultar las propiedades de la URL y acceder a una nueva a través de *location*.

Objetivo: conocer el contenido del objeto *location* del BOM

Cookies



fragmento de información que un navegador web almacena en el disco duro del visitante a una página

La información se almacena a petición del servidor web

directamente desde la propia página web con **JavaScript**

desde el servidor web mediante las **cabeceras HTTP**, que pueden ser generadas desde un lenguaje de web scripting como **PHP**.

Las cookie correspondientes a un servidor web se almacenan como una única cadena

Nombre
Valor
Validez

Cookies: funcionalidad



La información almacenada en una cookie
puede ser recuperada por el servidor web
en posteriores visitas a la misma página

compensa el carácter
"sin estado" (stateless)
del protocolo HTTP

Cookies en JS

`document.cookie`

propiedad cookie del
objeto document

accede a todas las cookies de una
página web, que son devueltas como
una única cadena

Al crear una nueva, se
añade respetando el
contenido anterior

Para acceder a una cookie
concreta, es necesario
analizar la cadena para
localizar su valor

Lo habitual es **crear las funciones** setCookie() y getCookie() para
facilitar estas dos operaciones

Operaciones con las cookies

Creación o modificación

```
document.cookie = "<nombre>=<valor>;  
[expires=<time UTC>] [path=/]"
```

```
document.cookie += "username=John Doe;  
expires=Thu, 18 Dec 2015 12:00:00 UTC; path=/;"
```

Si la cookie indicada no existe, la añade automáticamente a las existentes

Lectura

```
var x = document.cookie;
```

Se obtiene “cookie1=value; cookie2=value; cookie3=value...”

Eliminación

```
document.cookie += "<nombre>=;  
[expires=<fecha pasada UTC>];"
```

setCookie(): ejemplo

```
function setCookie(cname, cvalue, exdays) {  
    var d = new Date();  
    d.setTime(d.getTime() + (exdays*24*60*60*1000));  
    var expires = "expires=" + d.toUTCString();  
    document.cookie += cname + "=" + cvalue + "; " + expires;  
}
```

Se transforma el número de días en una fecha válida, a partir de la fecha en que se ejecuta la función

Se incorpora a `document.cookie` el conjunto de valores de la nueva cookie

getCookie() : ejemplo

```
function getCookie(cname) {  
    var name = cname + "=";  
    var ca = document.cookie.split(';' );  
    for(var i=0; i<ca.length; i++) {  
        var c = ca[i];  
        while (c.charAt(0)==' ') c = c.substring(1);  
        if (c.indexOf(name) != -1) return  
            c.substring(name.length,c.length);  
    }  
    return "";  
}
```

Se crea un array a partir de la cadena de cookies
Se recorre buscando el nombre de la cookie
Si se encuentra, se devuelve su valor

Ejercicio 6

Utilizando distintos botones para lanzarlas, creamos distintas funciones que permitan iniciar / borrar un contador del número de visitas a una página, mostrarlo y recargar la página, para incrementarlo.

Inicia/Borrar el
contador de visitas

Iniciar contador

Vaciar contador

Mostrar número de
visitas a la página

Mostrar

Recargar la página

Recargar

Comprobamos el uso de las cookies para almacenar información, disponibles en sucesivas sesiones del navegador

AC10

Storage APIs: Web Storage

Js

En el caso de *Web Storage* y su atributo *sessionStorage*, esta incorporación también incrementa el nivel de control y la eficiencia de las aplicaciones web.

Web Storage contiene dos importantes atributos que son a veces considerados APIs por sí mismos:



Offline & Storage

- *sessionStorage*
- *localStorage*

Ambos reemplazan la anterior función del sistema de **cookies** y fueron creados para superar sus limitaciones

Web Storage

JS

sessionStorage es responsable por mantener consistencia sobre la duración de la sesión de una página web y preservar información temporal como el contenido de un carro de compras, asegurando los datos en caso de accidente o mal uso (cuando la aplicación es abierta en una segunda ventana, por ejemplo).

localStorage nos permite grabar contenidos extensos de información en el ordenador del usuario. La información almacenada es persistente y no expira, excepto por razones de seguridad.

Establecer un valor: `localStorage.setItem('mi_clave', 'mi_valor');`
Recuperar un valor: `localStorage.getItem('mi_clave');`
Eliminar un valor: `localStorage.removeItem('mi_clave');`
Limpiar el Storage: `localStorage.clear();`

Ejercicio (1a)

Contador de visitas.

Replanteamos el ejercicio realizado con cookies : utilizando distintos botones para lanzarlas, creamos distintas funciones que permitan iniciar / borrar un contador del número de visitas a una página, mostrarlo y recargar la página, para incrementarlo.

Inicia/Borrar el
contador de visitas

Iniciar contador

Vaciar contador

Mostrar número de
visitas a la página

Mostrar

Recargar la página

Recargar

Comprobamos el uso de localStorage para almacenar información,
disponibles en sucesivas sesiones del navegador

DOM

(Document Object Model)

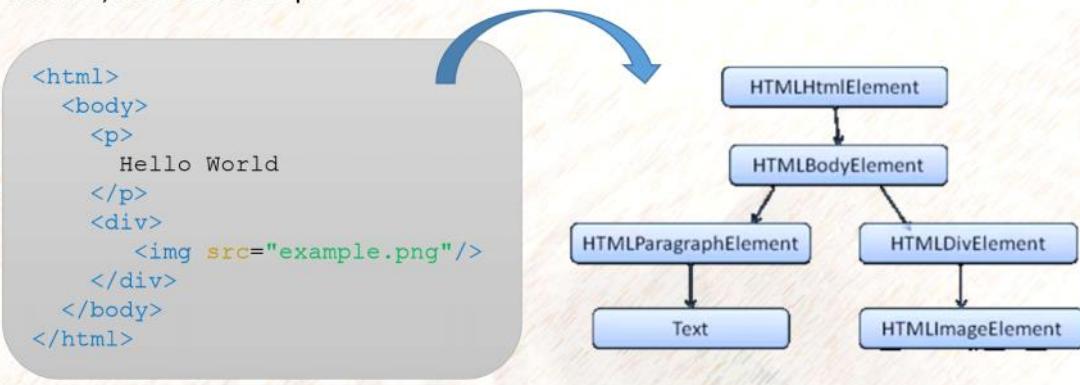
Árbol de nodos. Objetos Nodo

Niveles 1 y 2: Formas de acceso y manipulación

Modelo de objetos JQuery

DOM (Document Object Model)

DOM (modelo de objetos del documento) es la presentación estándar de los objetos del documento HTML y la interfaz de los elementos HTML para el mundo exterior, como JavaScript

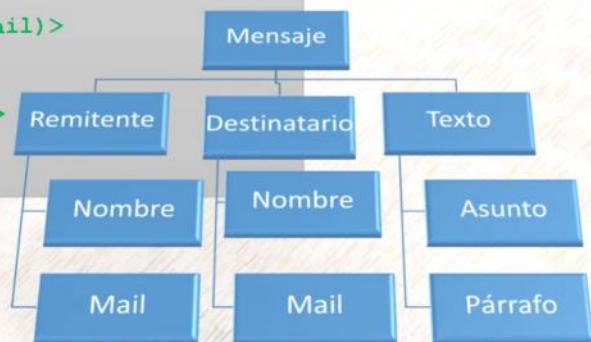


El árbol de salida ("árbol de análisis") de un motor de renderización está formado por elementos DOM y nodos de atributo.

Su origen: El DTD en XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Este es el DTD de Edit_Mensaje --&gt;

&lt;!ELEMENT Mensaje (Remitente, Destinatario, Texto)*&gt;
&lt;!ELEMENT Remitente (Nombre, Mail)&gt;
&lt;!ELEMENT Nombre (#PCDATA)&gt;
&lt;!ELEMENT Mail (#PCDATA)&gt;
&lt;!ELEMENT Destinatario (Nombre, Mail)&gt;
&lt;!ELEMENT Nombre (#PCDATA)&gt;
&lt;!ELEMENT Mail (#PCDATA)&gt;
&lt;!ELEMENT Texto (Asunto, Parrafo)&gt;
&lt;!ELEMENT Asunto (#PCDATA)&gt;
&lt;!ELEMENT Parrafo (#PCDATA)&gt;</pre>
```



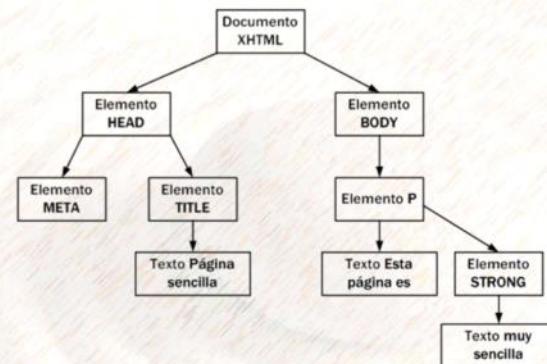
DTD: **document type definition**
(definición de tipo de documento)

Nodos y árbol de nodos

Js

Todos los navegadores realizan de forma automática, como parte del proceso de renderización, la transformación los documentos XHTML en un conjunto de elementos llamados **nodos**, que constituyen el DOM.

- La conversión de etiquetas en nodos se realiza de forma **jerárquica**, dando lugar al árbol de nodos
- Las páginas HTML habituales producen árboles con miles de nodos, pero el proceso de transformación es rápido y automático



Navegadores y DOM

Js

Desde su primera implementación, se han desarrollado sucesivamente tres conjuntos de especificaciones

- nivel 1
- • nivel 2
- nivel 3

El uso de DOM siempre está limitado por las posibilidades que ofrece **el motor de renderizado de cada navegador**.

- Los motores Gecko (**Firefox**), WebKit (**Safari, Chrome**) y Presto (**Opera**) implementan DOM de nivel 1 y 2 (y parte del 3),
- El motor Trident de **Internet Explorer** (versión 7 y anteriores) ni siquiera es capaz de ofrecer una implementación completa de DOM nivel 1.

DOM: Herramientas de desarrollo

JS

Firebug, en Firefox.
Pestaña DOM.



Herramientas para
desarrolladores,
en Chrome.
Pestaña Elements

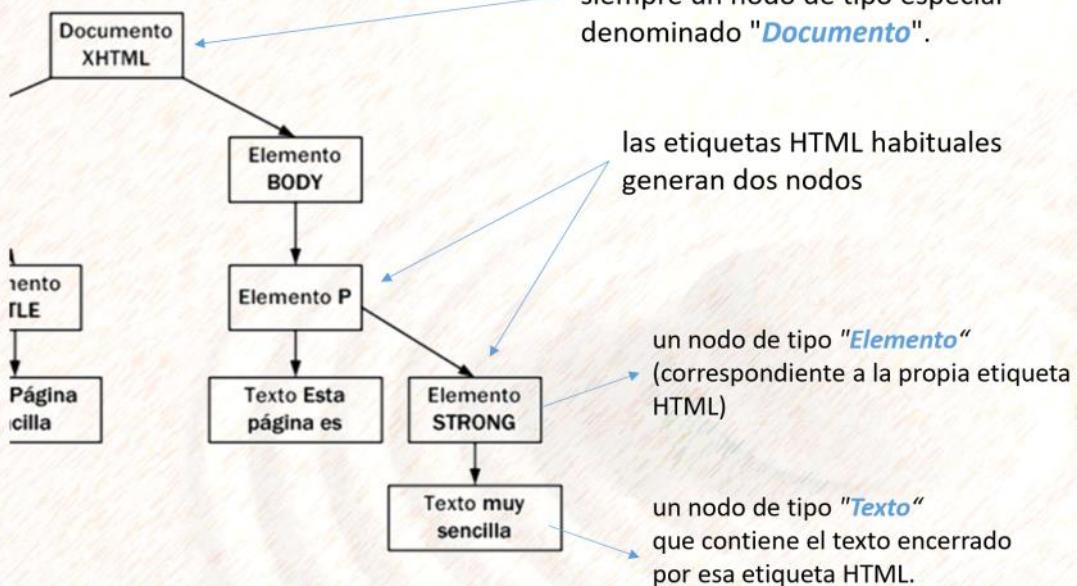


Del HTML al árbol de nodos

Js

```
<p>Esta página es <strong>muy  
sencilla</strong></p>
```

La raíz del árbol de nodos es siempre un nodo de tipo especial denominado "*Documento*".



Tipos de nodos XML y HTML

Js

Document, nodo raíz del que derivan todos los demás nodos del árbol.

DocumentType, es el nodo que contiene la representación del DTD empleado en la página (indicado mediante el DOCTYPE).

Element, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.

Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.

Text, nodo que contiene el texto encerrado por una etiqueta XHTML.

Comment, representa los comentarios incluidos en la página XHTML.

Notation

Entity

CDataSection

EntityReference

DocumentFragment

ProcessingInstruction

El objeto Node

Js

JavaScript crea el objeto Node para definir las propiedades y métodos necesarios para procesar y manipular los documentos

En primer lugar, el objeto Node define las siguientes constantes para la identificación de los distintos tipos de nodos:

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

Propiedades y métodos de Node (1)

JS

Propiedad / Método	Valor devuelto	Descripción
nodeName	String	El nombre del nodo En los de tipo 1 es el nombre de la etiqueta HTML. En algunos tipos es <i>null</i>
nodeValue	String	El valor del nodo (Es <i>null</i> para algunos tipos de nodo, e.g. para los de tipo 1)
nodeType	Number	Una de las 12 constantes definidas anteriormente

Los nodos de tipo *Element* (1) incluyen propiedades que reproducen el valor de sus nodos hijos de tipo *Text* (2) y *Attribute* (2)

Propiedad / Método	Valor devuelto	Descripción
textContent	String	El contenido del nodo de tipo Text correspondiente
attributes	NamedNodeMap	Se emplea con nodos de tipo Element. Contiene objetos de tipo Attr que definen todos los atributos del elemento

Propiedades y métodos de Node (2)

JS

Diversas propiedades sólo lectura reflejan las relaciones de un nodo

- con los adyacentes o "hermanos" (*sibling*) y
- con los que derivan de él o hijos (*child*)

Propiedad / Método	Valor devuelto	Descripción
ownerDocument	Document	Referencia del documento al que pertenece el nodo
childNodes	NodeList	Lista de todos los nodos hijos del nodo actual
children	HTMLCollection	Lista de todos los nodos tipo Element hijos del nodo actual
firstChild	Node	Referencia del primer nodo de la lista childNodes
lastChild	Node	Referencia del último nodo de la lista childNodes
previousSibling	Node	Referencia del nodo hermano anterior o null si este nodo es el primer hermano
nextSibling	Node	Referencia del nodo hermano siguiente o null si este nodo es el último hermano
hasChildNodes()	Boolean	Devuelve true si el nodo actual tiene uno o más nodos hijo

Propiedades y métodos de Node (3)

JS

Los métodos del objeto Nodo permiten añadir eliminar o reemplazar nodos hijos o insertar un nodo en el mismo nivel que el que ejecuta el método

Propiedad/Método	Valor devuelto	Descripción
appendChild(nodo)	Node	Añade un nuevo nodo al final de la lista childNodes
removeChild(nodo)	Node	Elimina un nodo de la lista childNodes
replaceChild(nuevoNodo, viejoNodo)	Node	Reemplaza un nodo hijo (viejoNodo) por el nodo nuevoNodo
insertBefore(nuevoNodo, anteriorNodo)	Node	Inserta el nodo hijo nuevoNodo antes que la posición del nodo anteriorNodo dentro de la lista childNodes (si no se indica el segundo parámetro, se añade al final)
cloneNode(deep)	Nodo	Devuelve una copia del nodo, incluyendo su descendencia si el valor Deep es true

Colecciones de nodos (1)

Js

Cuando se seleccionan simultáneamente varios nodos, o en la propiedad `childNodes / children`, se genera una **colección**, un objeto similar a un Array (*array-like*) cuyo prototipo puede ser **NodeList** o **HTMLCollection** (que no incluyen los métodos de la clase Array)

Si sacamos por consola el valor de la propiedad `childNodes` del elemento `<body>` de una página sencilla, vemos un ejemplo de esta estructura NodeList

```
dir(document.body.childNodes)
```

- Una **NodeList** puede contener nodos de cualquier tipo.
- Una **HTMLCollection** sólo contiene nodos de tipo *Element* e incorpora un método más, `namedItem()`

```
▼ NodeList[13] □
▶ 0: text
▶ 1: comment
▶ 2: text
▶ 3: header
▶ 4: text
▶ 5: aside
▶ 6: text
▶ 7: div
▶ 8: text
▶ 9: article
▶ 10: text
▶ 11: footer
▶ 12: text
▶ length: 13
▶ __proto__: NodeList
```

Colecciones de nodos (2)

Js

Aunque no es estrictamente un *array*, cualquier se recorre como si lo fuera, empleando un **bucle for**

Un **NodeList**
puede ser

- "vivo" (*live*) refleja cualquier cambio posterior en el DOM. Es el resultado normal cuando se crea un NodeList
- **estático**, NO refleja los cambios en el DOM posteriores a la creación del NodeList

Una
HTMLCollection

incorpora el método *namedItem(<key>)* que devuelve el primer elemento que tenga el valor indicado como atributo *id* o *name*

DOM - niveles 1 y 2

Js

Acceso relativo a los nodos

- document.childNodes[]

Acceso directo a
los nodos

- document.getElementsByTagName()
- document.getElementsByName()
- document.getElementById()

Crear nodos

- document.createElement(etiqueta)
- document.createTextNode(contenido)
- document.createAttribute(nombre)

Crear, modificar y
eliminar nodos

- nodo.appendChild(nodoHijo)
- nodo.removeChild()
- replaceChild()
- insertBefore()

añadir y modificar Atributos HTML y propiedades CSS

Nodos hijos

Js

Una vez accedido a un nodo (padre) → podemos acceder sus nodos hijos empleando 2 propiedades

oNode.childNodes → Devuelve una colección de todos los nodos hijo,
incluyendo los de tipo *text* y *comment*.
El formato devuelto es **NodeList**

oNodo.children → Devuelve una colección de todos los nodos hijo,
excluyendo los de tipo *text* y *comment*.
El formato devuelto es **HTMLCollection**

Secuencia de acceso a los nodos

Js

Recorrer la estructura
de nodos de una
página:

acceder al nodo raíz de la página y después a
sus nodos hijos y a los nodos hijos de esos
hijos y así sucesivamente

Nodo inicial:

document

Nodos “hijos”

childNodes

- nodo Document (tipo 10)
- HTML (tipo 1)

Nodos Element “hijos”

children

- HTML (tipo 1)

Nodo inicial:

document.HTML

- head (tipo 1)
- text (tipo 2)
- body (tipo 1)

Nodos “hijos”

childNodes

Nodos Element “hijos”

children

- head (tipo 1)
- body (tipo 1)

Acceso relativo a los nodos

Js

Acceder a un nodo a través de sus nodos padre:

acceder al nodo raíz de la página y después a uno de sus nodos hijos y de uno de los nodos hijos de esos hijos y así sucesivamente

Acceso a un nodo "hijo"

document
.childNodes[i]
.childNodes[i]

nodeType
nodeName
nodeValue
item

```
document.childNodes[1].childNodes[2].childNodes[7]...  
.nodeValue
```

Un ejemplo práctico en un fichero html : otra forma más dinámica de presentar la información del curso : JS21_DOM.html

Accesos transversales

Js

Acceso a un nodo “padre” es posible ser más específico que obtener la lista de hijos

- firstChild
- lastChild

A partir de a un nodo determinado también es posible moverse transversalmente (no solo de padres a hijos):

- previousSibling
- nextSibling

La Se puede consultar si un elemento tiene nodos "hijos"

propiedad
hasChildNodes()

Level 3 Transversales

Js

El tratamiento de los espacios en blanco (para algunos navegadores un nodo tipo texto) lleva a ciertas inconsistencias que tratan de ser corregidas por el **API Transversal de Elementos** incluida en el DON *Level 3*

- childElementCount
- firstElementChild
- lastElementChild
- previousElementSibling
- nextElementSibling

trabajan siempre con
nodos tipo *Element*, hijos
o hermanos, (excluyendo
los nodos de texto y de
comentario)

Todas siguen el mismo patrón que la propiedad *children*, ya introducida en el Level 1 como alternativa a *childNodes*

Ejercicio (7a)

El DOM desde la consola de JavaScript.

Tenemos una hoja HTLM que vamos a utilizar en la presentación de el **acceso relativo** a los nodos. Para seguir dinámicamente la presentación, comprobaremos la posibilidad de ir reproduciendo los comandos directamente en la consola.

[desarrollos\javascript\JS21_DOM_Secuencial.html](#)

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo accediendo a los nodos de forma secuencial directamente desde la consola de Javascript

Ejercicio (7b)

Modificación de una hoja HTLM.

Tenemos una hoja HTLM que hemos utilizado en la presentación de el **acceso relativo** a los nodos. Empleando esta técnica, modificaremos dinámicamente uno de los items de la lista inicial, de forma que al pulsar con el ratón se sustituya Flickr por Flickr (Fotos)

Desarrollos\javascript\JS21_DOM_Secuencial.html

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo accediendo a los nodos de forma secuencial.

Acceso directo a los nodos

domingo, 4 de junio de 2017 19:31

```
getElementsByName("nombreEtiqueta")
getElementsByName("nombre")
getElementById('id')
```

```
getElementsByClassName('valor')
querySelector('Selector')
querySelectorAll ('Selector')
```

Acceso directo a los nodos (1)

Js

getElementsByName(nombreEtiqueta) obtiene todos los elementos de la página HTML cuya etiqueta sea igual que el parámetro que se le pasa a la función

```
var parrafos = document.getElementsByTagName("p");
```

Array de nodos DOM con todos los párrafos, a los que podemos acceder individual o secuencialmente

nodo a partir del cual se realiza la búsqueda de los elementos. Si se utiliza el valor document, se obtendrán todos los párrafos de la página,

```
for(var i=0; i<parrafos.length; i++) {
    var parrafo = parrafos[i];
    alert(parrafo.textContent)
    ...
}
```

La función **getElementsByName()** se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función.

Acceso directo a los nodos (2)

Js

`getElementsByName("nombre")` obtiene todos los elementos de la página HTML cuyo atributo *name* sea igual que el parámetro que se le pasa a la función

```
var parrafoEspecial = document.getElementsByName("especial");
```

el método devuelve un [objeto de tipo NodeList](#), con los nodos que cumplen la condición, a los que se accede directamente con el correspondiente índice en el array de nodos.

Normalmente el atributo *name*, que se usa principalmente en formularios, es único, y solo existe un elemento [0]; caso diferente son los **radiobuttons**, que se agrupan precisamente porque comparten un mismo *name*

Acceso directo a los nodos (3)

Js

getElementById('id') obtiene el elemento de la página HTML cuyo atributo id sea igual que el parámetro que se le pasa a la función

```
var cabecera = document.getElementById("cabecera");
```

Como el atributo id es necesariamente único, el método devuelve un único objeto de tipo Nodo, con el que se accede directamente al nodo deseado. Es la más utilizada de estas funciones

getElementsByClassName('valor') retorna una *NodeList* con todos los elementos que incluyen en la propiedad **class** el valor o los valores indicado



```
var clase_1 = document.getElementsByClassName("clase_1");
```

Ejercicio (8a)

El DOM desde la consola de JavaScript.

Tenemos una hoja HTLM que vamos a utilizar en la presentación de el **acceso absoluto** a los nodos. Para seguir dinámicamente la presentación, comprobaremos la posibilidad de ir reproduciendo los comandos directamente en la consola.

[Desarrollos\javascript\JS22_DOM_Directo.html](#)

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo accediendo a los nodos de forma secuencial directamente desde la consola de Javascript

Ejercicio (8b)

Modificación de una hoja HTLM.

Tenemos una hoja HTLM que hemos utilizado en la presentación de el **acceso absoluto** a los nodos. Empleando esta técnica, añadiremos la posibilidad de eliminar cada párrafo una vez leído, igual que ocurría cuando empleábamos directamente los comandos en la consola.

[Desarrollos\javascript\JS22_DOM_Directo.html](#)

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo accediendo a los nodos de forma secuencial directamente desde la consola de Javascript

Acceso directo a los nodos : API de selectores

Js

querySelector('Selector')

retorna el primer elemento que concuerda con el grupo de selectores declarándolos usando comillas y con la sintaxis de CSS,

```
document.querySelector("#principal p:firstchild")
```

querySelectorAll ('Selector')

retorna una instancia estática de una *NodeList* con todos los elementos que concuerda con el selector, nuevamente declarado entre comillas y con la sintaxis CSS

HTML5: Extensiones del DOM

Js



- Relativo a "clases"
método `getElementsByName('valor')`
propiedad `classList`
- Gestión del foco
propiedad `activeElement`
método `hasFocus()`
- Cambios en `HTMLDocument`
- Set de caracteres
propiedades `charset` y `defaultCharset`
- Atributos data- personalizados
propiedad `dataset`
- Inserción de HTML
propiedades `innerHTML` y `outerHTML`
método `insertAdjacentHTML()`
- Método `scrollIntoView()`

Ejercicio (9a)

Una lista de enlaces cambiantes:

Codificar una lista de enlaces que cambian de texto cuando se hace clic sobre ellos, como los que aparecen en el ejemplo.

Opción emplear OOP y crear una función constructora (e.g.
Lista) de la que se instancia un objeto

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo.

Ejercicio (09b)

Una lista de enlaces cambiantes:

Repetir la lista de enlaces que cambian de texto cuando se hace clic sobre ellos, utilizando en este caso el API de **selectores**. y partiendo de una estructura HTML como la que se muestra.

```
<ul>
  <li class="sprite">
    <a id="twitter" href="#" ...>Twitter</a>
  </li>
  ...
</ul>
```

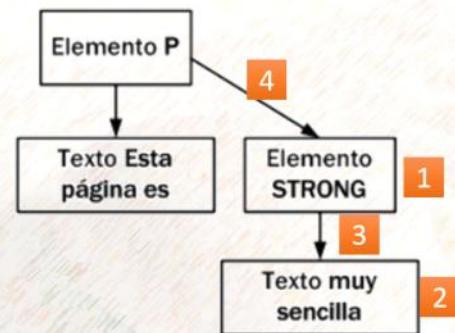
Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo.

Crear nodos

Js

Añadir un nodo como hijo de otro nodo es un proceso que involucra una serie de pasos sucesivos

1. Crear nodo de tipo Element
2. Crear el correspondiente nodo de tipo Text, con el contenido que nos interesa
3. Añadir el nodo Text como hijo del nodo Element
4. Finalmente, añadir el nodo Element como hijo de algún nodo de la página; previamente lo tendremos localizado, e.g. con la función getElementsById(id) que ya conocemos.



Crear nodos: métodos usados

Js

- **createElement(etiqueta)**: crea un nodo de tipo Element que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
- **createTextNode(contenido)**: crea un nodo de tipo Text que almacena el contenido textual de los elementos XHTML.
- **nodoPadre.appendChild(nodoHijo)**: añade un nodo como hijo de otro nodo. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo Text como hijo del nodo Element y a continuación se añade el nodo Element como hijo de algún nodo de la página.

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola
Mundo!");
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);
// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

Crear nodos: otros métodos

Js

createAttribute(nombre) Crea un nodo de tipo atributo con el nombre indicado

createCDataSection(texto) Crea una sección CDATA con un nodo hijo de tipo texto que contiene el valor indicado

createComment(texto) Crea un nodo de tipo comentario que contiene el valor indicado

createDocumentFragment() Crea un nodo de tipo DocumentFragment

createEntityReference(nombre) Crea un nodo de tipo EntityReference

createProcessingInstruction(objetivo, datos) Crea un nodo de tipo ProcessingInstruction

Eliminar nodos

Js

- **removeChild()**: permite eliminar un nodo del árbol DOM de la página, pasándole como parámetro el nodo que se va a eliminar.

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
```

Esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad `nodoHijo.parentNode`

Copiar (clonar) nodos

Js

- **cloneNode()**: permite copiar un nodo del árbol DOM almacenándolo como una variable. El clon puede ser modificado y añadido como hijo en cualquier posición

```
var parrafo = document.getElementById("provisional");
var copia = parrafo.cloneNode();
-----
oElement.appendChild(copia);
```

El único parámetro posible (deep) indica la profundidad de la copia:

- false (por defecto) copia el nodo y sus atributos
- true copia también todos sus descendientes

Modificar nodos

Js

La propiedad **textContent**

- en modo lectura (sin darle valor) recoge el texto contenido en un elemento
- en modo escritura permite sustituir ese texto por un nuevo valor, o añadir este último, empleando += “Añadir texto”. En cualquier caso sin incluir marcadores HTML, que serían interpretados como literales, no como marcas

```
var parrafo = document.getElementById("provisional");
parrafo.textContent = "Nuevo texto";
```

En esta propiedad se reproduce, en caso de que exista, el **nodo de tipo texto** (3) contenido en un nodo de tipo elemento (1)

Modificar nodos

Js

La propiedad **innerHTML** se incorpora como estándar en HTML5:



- en modo lectura (sin parámetros) devuelve el conjunto de todos los nodos hijos de un elemento
- en modo escritura permite añadir (+=) / sobrescribir el contenido completo de un nodo

```
var div1= document.getElementById("provisional");
div.innerHTML = "<p>Nuevo texto</p>";
```

También se incorpora la propiedad **outerHTML**:



- en modo lectura (sin parámetros) devuelve un nodo con todo los elementos que contiene
- en modo escritura permite sobrescribir un nodo completo, incluido el propio elemento

Ejercicio (10)

Aumentar una lista.

Crear una página con un lista y un botón o un enlace, de modo que cada vez que lo pulsa, el usuario pueda añadir un nuevo ítem a la lista.

Detalle: La acción de pinchar sobre un enlace o botón corresponde al ya conocido evento onclick que ejecutará un función; junto con los demás "Eventos" de JavaScript, se verán a continuación con profundidad.

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo.

Ejercicio (11)

Una lista dinámica de los reyes de España.

A partir del ejercicio anterior y el realizado previamente, en el que se comprobaba contra un array si un nombre dado correspondía a los usados por algún rey de España, completar una página html que, a modo de concurso permita crear desde cero y dinámicamente una lista con los reyes de España.

Objetivo: Comprobar como se aplica el uso del DOM y sus funciones en una posible situación real, como es validar las entradas de una lista construida por el usuario frente a un array de posibles valores validos.

Acceso a los atributos HTML

Js

Los **atributos** HTML de los elementos de la página se representan en el DOM de diversas maneras

- son **nodos "hijo"** de tipo *attribute* (2) del elemento al que se aplican
- se mapean en la **propiedad *attributes*** del elemento, en forma de `namedNodeMap`, un colección viva, equivalente a un `nodeList`
- pueden ser leídos y manipulados por las **propiedades específicas** `getAttribute()`, `setAttribute()`, and `removeAttribute()` del elemento al que se aplican. Estas propiedades permiten también manipular atributos definidos por el usuario
- los atributos estándar se transforman automáticamente en propiedades de los nodos, de forma que pueden ser **accedidos directamente**.

Acceso directo a los atributos HTML

JS

Los **atributos** HTML de los elementos de la página se transforman automáticamente en **propiedades de los nodos**.

Para acceder a su valor, simplemente se indica el nombre del atributo HTML detrás del nombre del nodo

```
<a id="enlace" href="http://www...com">Enlace</a>
```



```
var enlace = document.getElementById("enlace");
alert(enlace.href); // muestra http://www...com
```

"*class*"
cambia de nombre → Dado que *class* es una palabra reservada, el atributo class se transforma en la propiedad **className**

Modificar las clases

Js

El **atributo class** almacena con frecuencia **múltiples valores**, pudiendo ser necesario modificar o eliminar alguno de ellos

Una vez recogido el valor, directamente con la propiedad **className** o mediante el método `getAttribute("class")`, se dispone de una cadena de caracteres que hay que procesar

HTML5 incorpora la propiedad **classList** para facilitar este proceso. En ella se recoge una colección del tipo `DOMTokenList` que puede recorrerse como un array de solo lectura y que incorpora varias funciones para poder manipularlo:

- `classList.add(clase1, clase2,...)`
- `classList.remove(clase1, clase2,...)`
- `classList.toggle(clase1)`
- `classList.contains(clase1)`



Acceso directo a las propiedades CSS

Js

Los **estilos** CSS de los elementos de la página definidos “en linea” se incorporan automáticamente a la **propiedades style de los nodos**.

Para acceder a su valor, simplemente se indica style el nombre del estilo CSS detrás del nombre del nodo

```

```



```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin); // muestra 0
```

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre de acuerdo con la camelNotation:

Estilos CSS y clases

Js

Para mantener la división entre conducta, estilo u estructura deben evitarse los estilos “en línea” y por tanto **no se recomienda** utilizar la propiedad **style** de los nodos del DOM

La alternativa es definir **clases** con los estilos necesarios y aplicar esas clases a los nodos del DOM, e.g. empleando la propiedad **classList** (HTML5)

Otra opción es modificar los estilos definidos en la cabecera del documento , como se ve en el ejemplo

```
var style = document.createElement("style");
style.type = "text/css";
style.appendChild(document.createTextNode("body{background-
color:red}"));
var head = document.getElementsByTagName("head")[0];
head.appendChild(style);
```

Ejercicio (12)

Mostrar la parte oculta de un texto:

Al pulsar sobre “ver mas” o similar se nos muestra completo el texto del que estábamos viendo el principio

Objetivo: Familiarizarnos con el uso del DOM y las funciones que permiten manipularlo, concretamente getElementById(id) y los cambios de los atributos de una etiqueta.

Eventos

jueves, 1 de junio de 2017 16:34

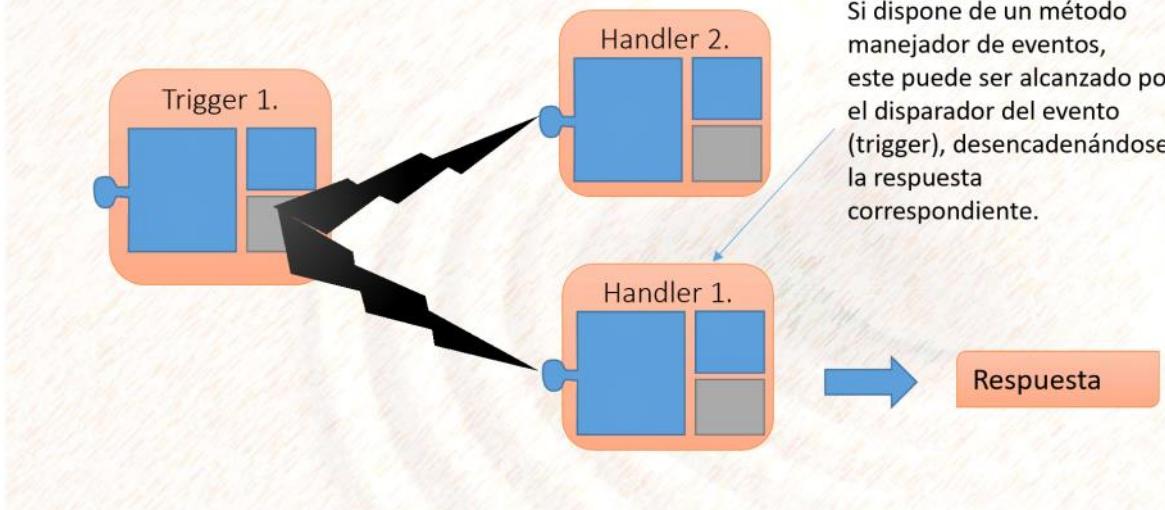
Eventos



Ligado a la OOP está el modelo de programación basada en eventos

Los **objetos** pueden anunciar sus **cambios de estado** disparando uno de sus eventos

Cuando una clase está a la escucha:



Event trigger (Desencadenante)



Definición del evento entre los atributos públicos de la clase

```
CLASS c1 DEFINITION.  
  PUBLIC SECTION.  
    EVENTS e1 EXPORTING VALUE(p1)  
           TYPE i.  
  METHODS m1.  
  
  PRIVATE SECTION.  
    DATA a1 TYPE i.  
  
  ENDCLASS.
```

Uno de los métodos de la clase desencadena (“dispara”) el evento

```
CLASS c1 IMPLEMENTATION.  
  METHOD m1.  
    a1 = ...  
    RAISE EVENT e1  
      EXPORTING p1 = a1.  
  ENDMETHOD.  
  ENDCLASS.
```

Event handler (*Manejador*)



Un método concreto de una segunda clase se define como responsable de responder a un determinado evento ("manejarlo").

```
CLASS c2 DEFINITION.  
  PUBLIC SECTION.  
    METHODS m2 FOR EVENT e1 OF c1  
      IMPORTING p1.
```

```
  PRIVATE SECTION.  
    DATA a2 TYPE i.
```

```
ENDCLASS.
```

La implementación de ese método será la respuesta al evento

```
CLASS C2 IMPLEMENTATION.  
  METHOD m2.  
    a2 = p1.  
    ...  
  ENDMETHOD.  
ENDCLASS.
```

Eventos en el navegador

Eventos son las acciones reflejo del comportamiento de los usuarios y del propio navegador

A muchos de ellos responde de forma predeterminada el navegador

Eventos del ratón

- click
- dblclick
- mousedown
- mouseup
- mouseover
- mouseout
- mousemove

Eventos del documento / la ventana

- load
- resize
- scroll
- unload

Eventos de formulario

- submit
- reset
- change
- focus
- blur
(lost focus)

Eventos del teclado

- keypress
- keydown
- keyup

Eventos del sistema (1)

Evento	Descripción	Elementos para los que está definido
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>, <textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>

Eventos del sistema (2)

Evento	Descripción	Elementos para los que está definido
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Nuevos atributos “Evento” en HTML5

Window

onafterprint
onbeforeprint
onbeforeunload
onerror
onhaschange
onmessage
onoffline
ononline
onpagehide
onpageshow
onpopstate
onredo
onresize
onstorage
onundo

Form

oncontextmenu
onformchange
onforminput
oninput
oninvalid

Mouse

ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart
ondrop
onmousewheel
onscroll

Media Events

oncanplay
oncanplaythrough
ondurationchange
onemptied
onended
onerror
onloadeddata
onloadedmetadata
onloadstart
onpause
onplay
onplaying
onprogress

onratechange
onreadystatechange
onseeked
onseeking
onstalled
onsuspend
ontimeupdate
onvolumechange
onwaiting

Respuesta a los eventos

Js

Para cada evento, HTML define
un atributo manejador de eventos → **on<evento>**

Así a cada evento se le puede asignar una función que
actuará como **manejador de eventos** (*event handler*).

La forma en que se gestiona la asignación de funciones para manejar
los eventos depende del navegador, existiendo varias posibilidades

- funciones definidas directamente en HTML (JS embebido)
- funciones definidas a nivel del DOM:
se definen varios modelos de eventos

Manejadores de eventos HTML: atributos de los elementos

Js

El código se incluye en un atributo del propio elemento

```
<input type="button" value="Pínchame y verás"  
      onclick="alert('Gracias por pinchar');" />
```

↑
se aprovechan los atributos manejadores de
eventos definidos en HTML

Es posible agrupar todo el código JS en una función externa y llamarla
desde el elemento HTML.

```
function muestraMensaje() {  
    alert('Gracias por pinchar');}  
<input type="button" value="Pínchame y verás"  
      onclick="muestraMensaje()" />
```

Es el método menos recomendable porque no respeta la separación de capas

Manejadores de eventos HTML: this y event

Js

En el código JS embebido en HTML podemos referirnos al elemento como **this**. NO en una función asociada al manejador HTML

En ambos casos se crea la variable **event**, con el objeto **Event**

```
<div id="contenidos" style="border:thin solid silver"  
onmouseover="this.style.borderColor='black';"
```

Se crea automáticamente la variable **this**, referencia al elemento HTML que ha provocado el evento

```
<input type="button" value="Haz Clic" onclick="hacerClick()">  
...function hacerClic () {alert(this.value)"}
```

ERROR: la referencia **this** sería incorrecta

Modelos de Eventos

Js

DOM Level 0

Modelo básico de eventos. Se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Sus características son limitadas, pero lo soportan **todos los navegadores**.

DOM Level 2

Modelo de eventos estándar. Esta definido en las versiones más avanzadas del estándar DOM (DOM nivel 2) y es mucho más poderoso que el original. Lo incluyen todos los **navegadores modernos**, salvo Internet Explorer

Modelo de eventos de Internet Explorer. Es similar pero incompatible con el modelo estándar, y exclusivo de **Internet Explorer**.

Modelo básico de eventos

Js

- versión 4 del estándar HTML
- parte del nivel más básico de DOM (*Level 0*).
- características limitadas
- Soporte en **todos los navegadores**.

Cada elemento o etiqueta HTML define su propia lista de posibles eventos que se le pueden asignar



más utilizados

- **onload**: esperar a que se cargue la página por completo
- **onclick, onmouseover, onmouseout** : controlar el ratón
- **onsubmit**: controlar el envío de los formularios.

Manejadores de eventos "semánticos"

Js

Método más acorde con las buenas prácticas básicas en el diseño de páginas y aplicaciones web: la separación de los contenidos (HTML) y su aspecto o presentación (CSS).

En esa línea el código se debe mezclar lo menos posible con los elementos de HTML

Para responder al onclick de un elemento cuyo id es "clicAqui"

- creamos la función funcionOnclick ()
- la asociamos al evento en el elemento a nivel del DOM

`element.onevent=functionName`

`document.getElementById("clicAqui").onclick = funcionOnclick;`

sin paréntesis, queremos la función, no su resultado

Ejemplo de Manejadores "semánticos"

Js

```
<script>
function muestraMensaje() {
alert('Gracias por pinchar');
}
```

Función externa

```
document.getElementById("pinchable").onclick
= muestraMensaje;
```

Se asignar la función externa
al elemento

```
<input id="pinchable" type="button"
value="Pinchame y verás" />
```

En el cuerpo HTML solo hay que añadir el íd
único del elemento, nada de código JS

Carga completa de la página

Js

Para los manejadores
semánticos, es importante
recordar que

antes de que se puedan utilizar las
funciones DOM la página se debe
cargar completamente

Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

```
window.onload = function() {  
    document.getElementById("valorId").onclick  
    = miFuncion;  
}
```

Asocio al evento onload una función anónima, en la que incluyo aquellos procesos que deben llevarse a cabo una vez que se ha cargado la página

El objeto Event (1)

JS

información adicional
sobre el evento

→ objeto event
(grandes diferencias entre navegadores)

MSIE

el resto

objeto Event forma parte
del objeto Window

el único argumento que tienen las
funciones manejadoras de eventos.

```
var oEv = window.event
```

```
function manejadorEventos(oEv) {  
    var evento = oEv;  
}
```

Código válido en
ambos casos

```
function manejadorEventos(oEv) {  
    var evento = oEv || window.event;  
}
```

El objeto Event (2)

Js

propiedades

bubbles
cancelable
currentTarget
defaultPrevented
eventPhase
isTrusted
target: elemento que desencadena el evento
timeStamp
type: nombre del evento
view

metodos

initEvent()
preventDefault()
stopImmediatePropagation()
stopPropagation()

Información sobre el evento

Js

Propiedades del
objeto Event

type: nombre del evento
sin el prefijo on

```
function resalta(elEvento) {  
    var evento = elEvento || window.event;  
    switch(evento.type) {  
        case 'mouseover':  
            this.style.borderColor = 'black';  
            break;  
        case 'mouseout':  
            this.style.borderColor = 'silver';  
            break; }  
}
```

```
window.onload = function() {  
oElemento = document.getElementById("seccion")  
oElemento.onmouseover = resalta.bind(oElemento);  
oElemento.onmouseout = resalta.bind(oElemento);  
}
```

Ejercicio (16a)

Estilo dinámico empleando manejadores de eventos HTML y funciones:

Cambiar el color del borde que rodea a un elemento (i.e. párrafo, div...) según este el ratón sobre él o no. Hacer que suceda en dos elementos, utilizando en uno estilos y en otro clases.

Extra: Asegurarse de que el código sea válido en Firefox, Internet Explorer y Opera. Para ello hay que tener en cuenta como almacena la información sobre los colores cada navegador.

Objetivo: Familiarizarnos con el uso de los Eventos y las funciones manejadores de eventos. Utilizar **this** en funciones manejadoras de eventos vinculadas a atributos HTML

Comprobar las **inconsistencias de los navegadores** y usar switch...case para corregirlas.

Ejercicio (16b)

Estilo dinámico y manejadores semánticos :

Replantear el ejercicio en el que se cambia el color del borde que rodea a un elemento (i.e. párrafo, div...) según este el ratón sobre él o no. En este caso serán varios los elementos implicados y un solo manejador gestionará todos ellos

Utilizaremos la opción mediante clases y nos aseguraremos de que el código funcione si son varias las clases asignadas a un elemento.

Objetivo: Familiarizarnos con el uso de los Eventos y las funciones manejadores de eventos. Comprobar el uso del objeto Event para ampliar la funcionalidad de un manejador de eventos.

Modelos de Eventos: DOM 2

Js

DOM Level 0

Modelo básico de eventos. Se introdujo para la versión 4 del estándar HTML y se considera parte del nivel más básico de DOM. Sus características son limitadas, pero lo soportan **todos los navegadores**.

DOM Level 2

Modelo de eventos estándar. Esta definido en las versiones más avanzadas del estándar DOM (DOM nivel 2) y es mucho más poderoso que el original. Lo incluyen todos los **navegadores modernos**, salvo Internet Explorer

Modelo de eventos de Internet Explorer. Es similar pero incompatible con el modelo estándar, y exclusivo de **Internet Explorer**.

Registro de eventos



Se crean los objetos desencadenador y manejador con sus referencias correspondientes

Se declaran "a la escucha" los manejadores

Se desencadena el evento.

PROGRAM ...

DATA: trigger TYPE REF TO trigger,
handler_1 TYPE REF TO handler,
handler_2 TYPE REF TO handler.

CREATE OBJECT: trigger, handler_1, handler_2.

SET HANDLER

handler_1->handle_event
handler_2->handle_event
FOR trigger.

.....

CALL METHOD trigger->raise_event.

Automáticamente, el manejador detecta el evento y desencadena la respuesta

Flujo de eventos "propietarios"

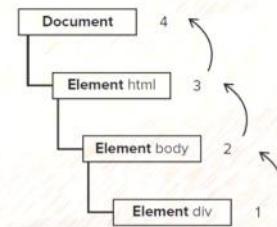
Js

Eventos "propietarios": navegadores 4.x

Internet Explorer

propagación de eventos
(*event bubbling*)

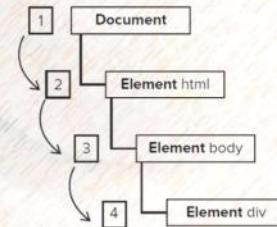
el evento empieza en el elemento más específico



Netscape Communicator

captura de eventos
(*event capturing*)

el evento empieza en el elemento más genérico



Flujo de eventos en el DOM 2

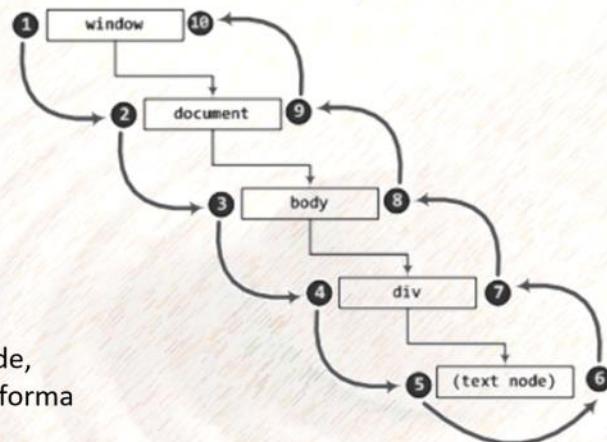
Js

El flujo de eventos (*event flow*) permite que varios elementos diferentes puedan responder a un mismo evento en cualquier navegador.

Eventos DOM

1. event capturing
2. event bubbling

El nodo más específico, text node, puede ejecutar dos eventos de forma consecutiva.



```
<html onclick="procesaEvento()">
<head><title>Ejemplo de flujo de eventos</title></head>
<body onclick="procesaEvento()">
<div onclick="procesaEvento()">Pincha aqui</div>
</body>
</html>
```

API EventTarget

Es el API **estándar** del DOM (DOM nivel 2) que permite definir como se reciben y se disparan eventos

asociar y desasociar
manejadores de eventos → addEventListener()
removeEventListener()

disparar un evento → dispatchEvent()

Internet Explorer define su propio API para gestionar los eventos

asociar y desasociar
manejadores de eventos → attachEvent()
detachEvent()

Escucha de eventos (*listeners*)

Js

estándar - asociar y desasociar manejadores de eventos

`addEventListener()` define uno o más manejadores “a la espera” de un evento determinado: escuchando (*listeners*)

`removeEventListener()` elimina una asociación previa

Parámetros

nombre del evento
función encargada de procesarlo
tipo de flujo *bubbling* (default false)

Internet Explorer

attachEvent()
detachEvent()

2 parámetros

nombre del evento
función encargada de procesarlo

Disparo de eventos

Js

El API **estándar** del DOM (DOM nivel 2) *Event Target* define un método

dispatchEvent() → capaz de lanzar (*trigger* o *dispatch*)
un evento como si se tratara de
un evento de lanzamiento directo.

```
elemento.dispatchEvent(new Event(<nombre evento>))
```

El único parámetro es un objeto Evento
(con el nombre del evento)

La escucha de estos eventos es idéntica a la de los del sistema

```
elemento.addEventListener(<nombre evento>, funcion)
```

Ejercicio (17)

Párrafos que se muestran u ocultan

Crear una página HTML donde se incluyan diversos párrafos (inicialmente ocultos), con su título visible.

- Al pasar el ratón sobre un título, se desplegará la primera parte del párrafo con un enlace al final que permita ocultar o mostrar el resto del párrafo.
- Si se está mostrando el párrafo completo continuará desplegado aunque el ratón se desplace a cualquier otro punto de la pantalla.

Se utilizarán los manejadores semánticos del DOM Level 2

Objetivo: Familiarizarnos con el uso de los Eventos y los manejadores de eventos semánticos del DOM Level 2 (`addEventListener...`).

Eventos de teclado

Js

Se suceden

- keydown → Info más sencilla
 - keyCode (código interno de tecla)
 - charCode (no definida)
- keypress → Info más técnica (*):
 - código del carácter
 - keyCode (teclas especiales)
 - charCode (teclas normales)
- keyup → Info más técnica
 - keyCode (código interno de tecla)
 - charCode (no definida)

En keypress, de código de carácter a carácter : String.fromCharCode()

```
String.fromCharCode(evento.charCodeAt() || evento.keyCode)
```

Eventos de ratón

Js

Coordenadas de la posición
del puntero del ratón

propiedades

respecto de
la ventana del navegador

clientX
clientY

respecto de
la pantalla completa

screenX
screenY

respecto del
origen de la página

pageX
pageY

(serán distintas de las primeras si el
usuario ha hecho scroll)

Diferente en IE

clientX + document.body.scrollLeft;
clientY + document.body.scrollTop;

Ejercicio (18)

Eventos de teclado y ratón

Crear una página html que muestre posición del ratón (respecto del navegador y respecto de la página) y teclas pulsadas (carácter y código), tal como aparece en la imagen

Ratón

Navegador [606, 457]

Página [606, 113]

Teclado

Carácter [c]

Código [99]

Objetivo: Familiarizarnos con el uso de los Eventos de teclado y ratón

Disparo de eventos

Js

El API **estándar** del DOM (DOM nivel 2) *Event Target* define un método

- dispatchEvent()** → capaz de lanzar (*trigger* o *dispatch*)
un evento como si se tratara de
un evento de lanzamiento directo.

```
elemento.dispatchEvent(new Event(<nombre evento>))
```

El único parámetro es un objeto Evento
(con el nombre del evento)

La escucha de estos eventos es idéntica a la de los del sistema

```
elemento.addEventListener(<nombre evento>, funcion)
```

Eventos propios

Js

Se pueden crear eventos "a medida" gracias a la función constructora Event()

```
var oEvento = new Event(<nombre>);
```

Los eventos así creados se manejan con los métodos del API *Event*
Target igual que si fueran eventos predefinidos lanzados
directamente por el sistema

```
// Escuchar el evento.  
elemento.addEventListener(<nombre>', function (e) { ... }, false);
```

```
// Disparar el evento  
elemento.dispatchEvent(oEvento);
```

Efectos

miércoles, 5 de septiembre de 2018 9:40

Imágenes para la web

Redimensionar y crear thumbnails o miniaturas de las imágenes

- [JPEG Resizer](#)
- [Easy Thumbnails](#)
- VSO Image Resizer
- (ahora [Light Image Resizer](#))

Programas para un completo tratamiento de las imágenes

Photoshop

gIMP 2.8

<http://www.gimp.org/>

gIMP 2.5 (<http://www.gimp.org.es/>)

parte del proyecto GNU: disponible bajo la Licencia pública general de GNU y GNU Lesser General Public License



El objeto Image()

Js

Normalmente las imágenes se incorporan en HTML con la etiqueta

También es posible almacenar una imagen como un **objeto Imagen** creado en JS gracias a su **función constructora** y vinculado con un fichero:

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
```

Entre las propiedades de este objeto están *width* y *height*, que toman como valor las dimensiones reales de la imagen una vez que esta se ha cargado

Esta técnica permite **pre-cargar** imágenes en memoria y tenerlas disponibles para poder usarlas más adelante

```
dom.src = newPhoto.src
```

Datos de imágenes precargadas

Almacenar una imagen como un objeto Imagen creado en JS y vinculado con un fichero determinado:

```
var imgRoyo = new Image();
imgRoyo.src = "205356_luis_royo.jpg"
```

Esta imagen pre-cargada puede ser incorporada desde JS a una etiqueta IMG,

```
var domImg2 = document.getElementById("lroyo2");
domImg2.src = imgRoyo.src;
domImg2.alt = "Ilustración de Luis Royo";
imgRoyo.onload = function () {
    domImg2.width = imgRoyo.width / 2;
}
```

en este caso con su tamaño al 50% calculado a partir del valor del objeto Imagen

Cambio de imágenes (Rollover)

Js

Rollover

Consiste en cambiar la imagen que aparece en la página en respuesta a algún evento desencadenado por el usuario

```
function cambiaImagen(oE) {  
    if (oE.type == "mouseover") {  
        oE.target.src = "images/537278_luis_royo.jpg"  
    } else {  
        oE.target.src = "images/205356_luis_royo.jpg"  
    }  
}
```

En lugar de lo que aparece en el ejemplo, Lo habitual es utilizar imágenes pre-cargadas

Alternativa

El evento puede desencadenarlo otro elemento

Botones asociados a imágenes que cambian

Cambiar de Imágenes



Para conseguir renovar toda la información de la imagen usamos un **objeto Image**

```
var newPhoto = new Image();
newPhoto.src = 'images/newImage.jpg';
```

Cambiamos de imagen con el atributo src, pero se mantienen el resto de los atributos establecidos previamente

```
$('#photo').attr('src','images/newImage.jpg');
```

Damos valores a los atributos del elemnto imagen existente en la web a partir del objeto imagen, que toma sus valores del fichero asociado a él

```
var photo = $('#photo');
photo.attr('src',newPhoto.src);
photo.attr('width',newPhoto.width);
photo.attr('height',newPhoto.height);
```

Ejercicio (28a)

Imágenes.

Utilizando JS o JQuery, crear el efecto de una imagen cambia por otra (rollover image) al pasar el ratón por encima.

Comprobar el resultado cargando directamente las imágenes o pre-cargándolas mediante un objeto Image()

Objetivo: comprobar la gestión de las imágenes desde JQuery

Ejercicio (28b)

Imágenes.

Utilizando JS o JQuery crear el efecto de que aparece una imagen u otra al pasar el ratón por encima del texto que describe (o hace referencia) a cada una de ellas..

Modificación. En lugar de los textos utilizamos ***thumbnails*** o miniaturas de las imágenes

Objetivo: comprobar la gestión de las imágenes desde JQuery



Galerías

domingo, 4 de junio de 2017 20:34

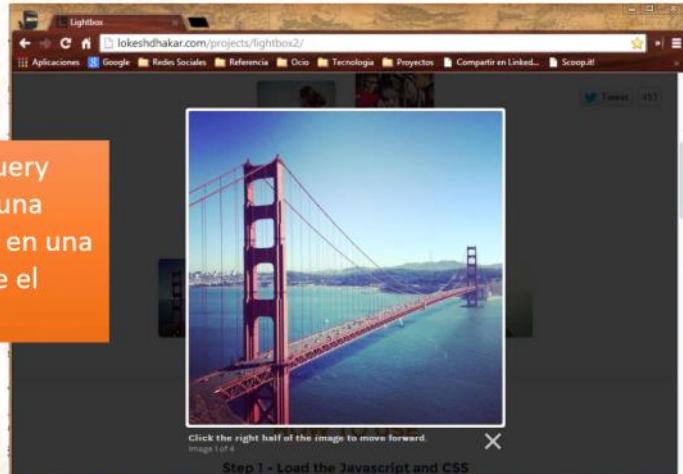
AC22

Galerías de imágenes



Existen muchas librerías disponibles para crear galerías de imágenes. La más conocida probablemente sea **Lightbox**, ahora en su versión 2
<http://lokeshdhakar.com/projects/lightbox2/>

Librería basada en JQuery que permite recorrer una sucesión de imágenes en una ventana flotante sobre el navegador

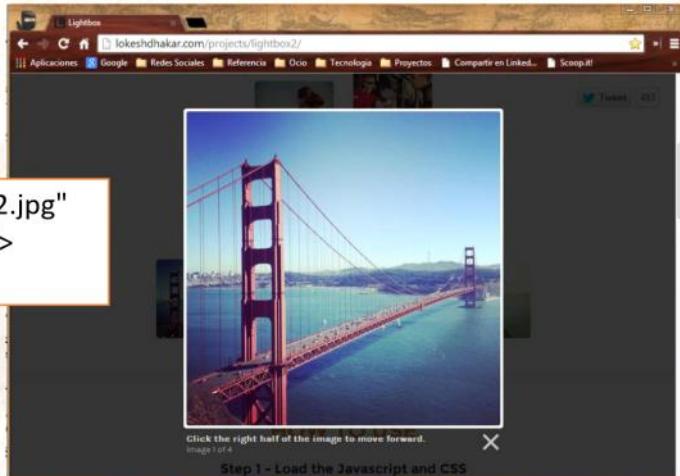


Galerías de imágenes - Lightbox



```
<a href="<imagen para el carrusel>"
  data-lightbox="<nombre del carrusel>">
  Texto o miniature de la imajen</a>
```

```
<a href="images/image-2.jpg"
  data-lightbox="roadtrip">
  Image #2</a>
```



JavaScript & jQuery: The Missing Manual, Second Edition
David Sawyer McFarland
O'Reilly Media, 2011

Ejemplo de Slider sencillo

```
$function () {
    var domContenedor =...
    var domLista = ...
    var domItems = ...
    var domBotones =...
}

.animate()
.stop()
.width()
.hasClass()
```

```
    var animateSlider = function (direction, duration, callback) {
        domLista.stop(true, true).animate({
            "margin-left": direction + "=300px"
        }, duration, callback);
    }

    var isAtStart = function () {
        return parseInt(domLista.css("margin-left"), 10) > -300;
    }

    var isAtEnd = function () {
        var imageWidth = domItems.first().width();
        var itemCount = domItems.length;
        var maxMargin = -1 * (imageWidth * (itemCount - 2));
        return parseInt(domLista.css("margin-left"), 10) < maxMargin;
    }

    domBotones.on("click", function () {
        var isBackBtn = $(this).hasClass("back");
        if ((isBackBtn && isAtStart()) || (!isBackBtn && isAtEnd())) { return; }
        animateSlider(isBackBtn ? "+" : "-", 1000);
    });
}
```

Ejercicio (29)

Imágenes con *Lightbox*.

Instalaremos la librería y realizaremos una galería con una serie de imágenes.

Objetivo: comprobar como podemos mejorar la gestión de las imágenes desde JQuery con alguna de las numerosas librerías existentes complementarias a JQuery

Ejercicio (3o)

Presentación de Imágenes: crear un *Slider*

Sin utilizar ninguna librería extra realizaremos una galería con una serie de imágenes. Utilizamos la pre-carga de las imágenes para crear un array con las imágenes

Objetivo: comprobar como podemos realizarla gestión de las imágenes desde JS/JQuery sin utilizar librerías complementarias a JQuery

DOM: desplazamientos



JQuery incorpora métodos que permiten modificar la posición de un elemento, cuando esta definida de forma absoluta o relativa

.offset() → recoge información / modifica posiciones absolutas

.position() → recoge información / modifica posiciones relativas

```
$('#id').offset(objetoCoordenadas);
```

Las posiciones se definen como un objeto con los valores top/bottom/right/left necesarios

```
{ top:100,  
left:100 }
```

DOM: Tamaño (Alto/Ancho)



En el mismo API, JQuery incorpora otros métodos que permiten modificar algunas propiedades CSS relacionadas con el aspecto de los elementos, además del método genérico .css()

.height() → recoge información / modifica
.width() altura y anchura

.innerHeight() → se refiere a la altura / anchura interna
.innerWidth() incluido padding pero no borde

.outerHeight() → se refiere a la altura / anchura interna
.outerWidth() incluido padding y borde

.scrollLeft() → recoge información / modifica
.scrollTop() las barras de scroll

DOM: Scroll



En el mismo API, JQuery incorpora métodos que relacionados con la barra de *scroll*

.scrollLeft()
.scrollTop()

recoge información / modifica
las barras de *scroll* de
cualquier elemento

`$(window).scrollTop()`

indicaría cuantos pixeles se ha desplazado
hacia arriba el documento al hacer *scroll*
vertical

Existe además HTML5 incorpora un evento *scroll* que se dispara
cada vez que hay algún desplazamiento vertical de un elemento
con *scroll*

Nuevos atributos “Evento” en HTML5

Window

onafterprint
onbeforeprint
onbeforeunload
onerror
onhaschange
onmessage
onoffline
ononline
onpagehide
onpageshow
onpopstate
onredo
onresize
onstorage
onundo

Form

oncontextmenu
onformchange
onforminput
oninput
oninvalid

Mouse

ondrag
ondragend
ondragenter
ondragleave
ondragover
ondragstart
ondrop
onmousewheel
onscroll

Media Events

oncanplay
oncanplaythrough
ondurationchange
onemptied
onended
onerror
onloadeddata
onloadedmetadata
onloadstart
onpause
onplay
onplaying
onprogress

onratechange
onreadystatechange
onseeked
onseeking
onstalled
onsuspend
ontimeupdate
onvolumechange
onwaiting

Equivalentes en JS

Js

Las funcionalidades relativas a la posición y el tamaño también aparecen en JS, como propiedades de los objetos Nodos obtenidos a partir del DOM

- element.clientHeight
 - element.clientWidth
 - element.clientLeft
 - element.clientTop
 - element.clientWidth
 - element.offsetHeight
 - element.offsetWidth
 - element.offsetLeft
 - element.offsetParent
 - element.offsetTop
-
- element.scrollHeight
 - element.scrollLeft
 - element.scrollTop
 - element.scrollWidth

Ejercicio

domingo, 4 de junio de 2017 20:41

Ejercicio (31)

Sticky Menu

Creamos un menú que

- inicialmente está por debajo del *header*
- al hacer *scroll* vertical se "pega" ("stick") a la parte superior de la ventana, de modo que nunca desaparece.

Objetivo: comprobar como utilizar las propiedades relativas a la posición y el *scroll* junto con el evento *scroll* de HTML5

UI: Widgets



<https://jqueryui.com/>

- Acordion
- Autocomplete
- Button
- Datepicker
- Dialog
- Menu
- Progressbar
- SelectMenu
- Slider
- Spinner
- Tabs
- Tooltip

Bootstrap - JavaScript

B

Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

[Download Bootstrap](#)

Currently v3.3.1

JavaScript

<http://getbootstrap.com/>

JS Alert

JS Button

JS Carousel

JS Collapse

JS Dropdown

JS Modal

JS Popover

JS Scrollspy

JS Tab

JS Tooltip

Ejercicio (32)

Elementos JS en Bootstrap

Creamos una página web utilizando Bootstrap e incorporamos algunos de los elementos que utilizan JS

- un carrusel de imágenes
- un elemento modal (e.g. div) que se muestra con aspecto de ventana emergente cuando se pulsa un botón.

Objetivo: comprobar como utilizar alguno de los elementos de *Bootstrap* desarrollados en JS