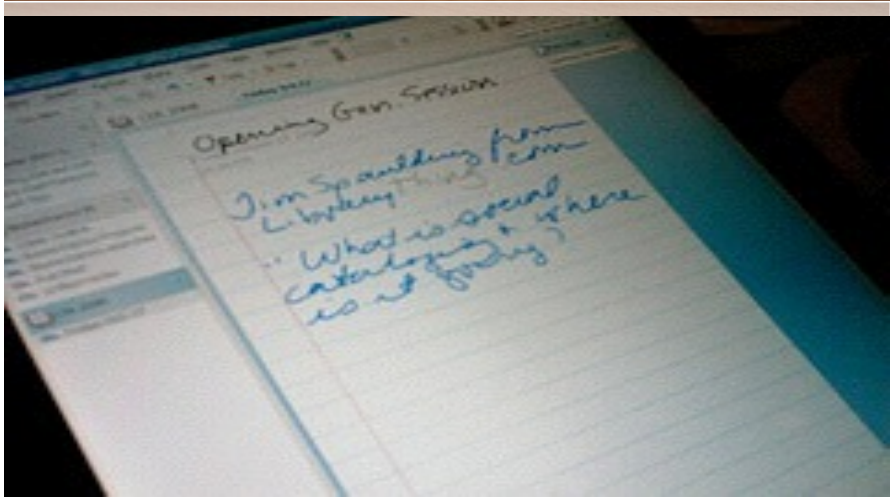


http://www.flickr.com/photos/dakdad/6011630039

3.5 Eventos

Introducción a Javascript

Reaccionando a las acciones del usuario

Eventos y manejadores

- Los eventos son acciones que ocurren generalmente porque el usuario hace algo sobre un objeto. Por ejemplo,
  - Hacer click sobre un botón
  - Introducir texto en un campo de formulario
  - Mover el ratón sobre un enlace ...

- Se pueden controlar esas acciones con un '**manejador**' de eventos (***handler***), para que el script reaccione ante ese suceso.

Algunos tipos de eventos

Evento	Se aplica a...	Ocurre cuando...	Handler
abort	imagenes	El usuario aborta la carga de una imagen (por ejemplo haciendo click sobre un enlace o en el boton del navegador) .	onAbort
blur	ventanas, frames, y todos los elementos de formularios	El usuario cambia el foco a otro elemento (ventana, frame, o elemento del formulario).	onBlur
click	Casi todos los elementos HTML	El usuario hace click en algo. Devolver false para cancelar la acción por defecto	onClick
change	campos de texto, area de texto, listas de selecisn.	El usuario cambia el valor de un elemento.	onChange
error	imagenes, ventanas	La carga de un documento o imagen causa un error	onError
focus	ventanas, frames, y todos los elementos de formularios	El usuario pasa el foco a otro elemento (ventana, frame, o elemento del formulario).	onFocus
load	body	El usuario carga la pagina.	onLoad

Algunos tipos de eventos *(cont.)*

Evento	Se aplica a...	Ocurre cuando...	Handler
mouseout	Casi todos los elementos HTML	El usuario mueve el puntero del ratsn fuera de un area (imagen) o enlace.	onMouseout
mouseover	Casi todos los elementos HTML	El usuario mueve el puntero del ratsn sobre un area (imagen) o enlace. En enlaces, devolver true para que no aparezca en la barra de status	onMouseover
reset	Formularios.	El usuario resetea un formulario (clicks en el botsn Reset) Devolver false para que no se haga el reset.	onReset
select	campos de texto, area de texto	El usuario selecciona un campo de entrada del formulario	onSelect
submit	formularios	El usuario envía un formulario. Devolver false para que no se haga el envío.	onSubmit
unload	Body,frameset	El usuario sale de la pagina	onUnload

Eventos y manejadores (forma inline)

1. Definir un manejador en un atributo HTML *onXXX="codigo-JavaScript"*
   - Problema: mezclamos HTML y javascript

```
<form name="prueba">
   <input type="button" value="Pulsa aquí"
        onClick="alert('hola');/>
</form>
```

2. Especificar valor de retorno
   - Devolviendo false o true se puede anular el comportamiento por defecto de algunos eventos. La mayoría de las veces es false, pero algunas es true

```
<a href="hola.htm" onclick="alert('hola');
return false;">hola</a>
```

Definir manejador de forma “no intrusiva”

- En el navegador, cada etiqueta HTML del documento tiene un objeto JavaScript equivalente.
  - Se accede al objeto (si la etiqueta tiene un `id`) con `document.getElementById`
  - Ese objeto tiene propiedades predefinidas que se corresponden con los atributos HTML.
  - Por ejemplo, un input tendría una propiedad `value`, otra `onclick`,...
- Ventaja: no mezclamos Javascript con HTML

```
<p id="par">Clícame</p>
<script type="text/javascript">
    function saludo() { alert("Hola")}
    document.getElementById(“par”).onclick = saludo;
    //también podríamos definir la función “sobre la marcha”
    //(función “anónima”)
   document.getElementById(“par”).onclick = function() {
                                            alert(“Hola”)}
</script>
```

this: el objeto que dispara el evento

- En un manejador de evento, **this** referencia al objeto sobre el que se produce el evento

```
<input type="text" id="campo">
<script type="text/javascript">
  function verContenido() {
    alert(this.value);
  }
  campo.onclick = verContenido;
</script>
```

Modificar el HTML: innerHTML

- Propiedad que nos permite leer/modificar el HTML que hay dentro de una etiqueta
- No es estándar en HTML4, pero sí en HTML5
  - Posteriormente veremos la forma estándar (DOM)
  - `innerHTML` no se recomienda para trabajar con tablas

```
<input type="button" value="Pon texto" onclick="ponTexto()"/>
<div id="texto"></div>
<script type="text/javascript">
  function ponTexto() {
    var mensaje = prompt("Dame un texto y lo haré un párrafo")
    var etiq = document.getElementById("texto")
    etiq.innerHTML += "<p>" + mensaje + "</p>"
  }
</script>
```

Eventos de formulario

- onsubmit (en el <form>): al intentar enviar los datos al servidor
  - Si el manejador devuelve `false`, se anula el envío
- onchange (en un campo): al cambiar el valor del campo (normalmente se dispara al pasar el foco a otro campo)

```
<script type="text/javascript">
  function checkForm() {
    var valor = document.getElementById("nombre").value
    if ((valor=="") || (valor==null))
        {alert("No puede estar vacío")
        return false;}
    else return true;}
</script>
<form action="loquesea.php" id="miform" onsubmit="return checkForm()">
    Nombre: <input type="text" id="nombre">
    <input type="submit" value="Enviar">
</form>
<!-- forma ALTERNATIVA a poner el "onsubmit=..." del formulario -->
<script type="text/javascript">
    document.getElementById("miform").onsubmit = checkForm;
</script>
```

Información sobre el evento

- El evento también es un objeto JavaScript con propiedades
  - Por ejemplo en un click, la propiedad `button` nos dice qué botón se ha pulsado
- Incompatibilidad
  - Explorer: el evento es un objeto global llamado "event"
  - Estándar W3C: se le pasa como argumento al manejador de evento

```
<p id="par">Clícame</p>
<script type="text/javascript">
    function saludo(e) {                    //"e" solo definido en Estándar
        if (window.event) {                 //event solo definido en Explorer
           e = window.event;
        }
        alert("Se ha pulsado el botón  " + e.button);
    }
   document.getElementById("par").onclick = saludo;
</script>
```

Event *bubbling*

- En realidad el evento no se dispara solo sobre el objeto que actúa directamente, continúa con los objetos "padre", "abuelo",...

```
<body onclick="alert('soy el body')">
   <p onclick="alert('soy el p')" id="parrafo">
        <a href="http://www.ua.es"
                onclick="alert('soy el a')">Clícame</a>
   </p>
</body>
```

http://jsfiddle.net/XHEeh/

- Se puede **cancelar** el *bubbling*, aunque API de IE<=9 no es estándar

```
document.getElementById("parrafo").onclick = function(event){
    event = event || window.event // cross-browser event
    if (event.stopPropagation) {  // W3C standard variant
        event.stopPropagation()
    } else {                      // IE pre-9
        event.cancelBubble = true
    }
}
```

Clausuras (*closures*)

- Una función siempre tendrá acceso a las variables que eran visibles cuando se definió
  - La clausura evita que las variables sean eliminadas por el "recolector de basura" mientras la función exista

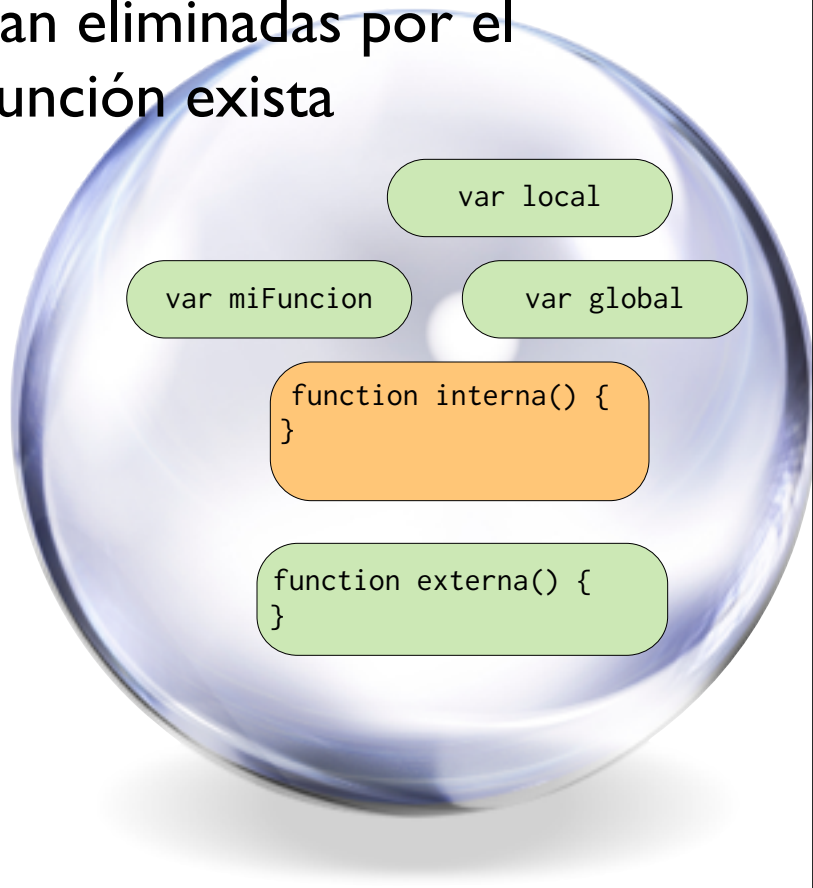
```
var global = "Soy el Mal";
var miFuncion;

function externa() {
	var local = "Yo soy local a 'externa'";

	function interna() {
		console.log(global);
		console.log(local);
	}

	miFuncion = interna;
}

externa();
miFuncion();
```



Eventos y clausuras

- Las clausuras pueden ser útiles para pasar datos al manejador

```
<p id="p1">Bla bla bla</p>
<p id="p2">Blo blo blo</p>
<script>
    function tooltip(mensaje) {
        return function() {
            alert(mensaje)
        }
    }
    document.getElementById("p1").onclick = tooltip("Hola")
    document.getElementById("p2").onclick = tooltip("qué tal")
</script>
```

http://jsfiddle.net/ACpk4/

Las clausuras son El Mal

- Las clausuras también pueden ser problemáticas con los eventos
  - Aviso: este código **no hace** lo que debería hacer

```
<div>div 0</div>
<div>div 1</div>
<script type="text/javascript">
    var divs = document.getElementsByTagName("div");
    for (var i = 0; i < divs.length; i++) {
        divs[i].onclick = function() {
            alert("Click en el div: " + i);
        };
    }
</script>
```

http://jsfiddle.net/zEjzy/

Las clausuras son **El Bien**

- No hay como combatir los problemas causados por una clausura introduciendo OTRA clausura

```
<div>div 0</div>
<div>div 1</div>
<script type="text/javascript">
 var divs = document.getElementsByTagName("div");
 for (var i = 0; i < divs.length; i++) {
    manejador = function(num) {
        return function() {
            alert("Click en el div: " + num);
        }
    }
    divs[i].onclick = manejador(i);
 }
</script>
```

http://jsfiddle.net/CDhaE/

Las clausuras son El Bien (II)

- Lo anterior ligeramente modificado, usando una función definida e invocada "sobre la marcha"

```
<div>div 0</div>
<div>div 1</div>
<script type="text/javascript">
 var divs = document.getElementsByTagName("div");
 for (var i = 0; i < divs.length; i++) {
     divs[i].onclick = (function(num) {
        return function() {
            alert("Click en el div: " + num);
        }
    })(i)
 }
</script>
```

Listeners (estándar W3C)

- Propagación: capturing + bubbling
  - **Capturing** significa que el evento comienza en realidad en el nivel superior
  - El capturing no funciona con los handler convencionales
- En lugar de un manejador de evento se usa un Event listener
  - Ante el mismo evento, un objeto puede tener **solo un handler** pero puede tener **varios listeners**
  - Definición
    - objeto.addEventListener(evento, funcion, captura?) (W3C)
    - objeto.attachEvent(evento, funcion) (Explorer pre-9)
  - Eliminación
    - objeto.removeEventListener(evento, funcion, captura?) (W3C)
    - objeto.detachEvent(evento, funcion) (Explorer pre-9)