Tabla de contenido

Los eventos

Los eventos son la base de la interactividad en JS. El usuario realiza alguna acción y en ese momento se desencadena un comportamiento. En este sentido un evento es una acción asíncrona, es decir, no sabemos de antemano cuándo se producirá. Ejemplos son onclick (al pulsar con el ratón), onload (al terminar la carga del documento), onfocus (al obtener el cursor un elemento), onblur (al perder el cursor), etc.

Sin embargo los estamos utilizando de una manera anticuada que no nos permite un control exhaustivo de los mismos. Esta forma se llama **gestión en línea** y en este apartado aprenderemos a utilizar eventos de una forma mucho más moderna y precisa.

Modelo de gestión de eventos en línea.

Es la forma tradicional (que va quedando obsoleta) de gestionar los eventos. Consiste en enlazarlos directamente en el código HTML mediante atributos que comienzan por la palabra "on" (que traducimos aquí por "cuando").

En este modelo para gestionar el evento click() utilizamos un manejador de eventos que es el atributo onclick del elemento donde lo insertamos.

Este tipo de eventos es gestionado directamente por el DOM. Fueron estandarizados en la especificación del W3C conocida como DOM de nivel 2. Y diferencia cuatro grupos de eventos: de ratón, de teclado, de HTML y exclusivos del DOM.

Los manejadores de eventos son atributos HTML y por lo tanto podrían escribirse en mayúsculas o minúsculas indistintamente. Podríamos, por ejemplo, escribir onClick = "…" o bien onclick="…". Sin embargo recuerda que si tu página está escrita en XHTML es obligatorio que los atributos de las etiquetas se escriban en minúsculas. Por este motivo es recomendable acostumbrarse a escribir los eventos siempre en minúsculas (onclick).

Otro factor importante es el de impedir que se produzca un evento. En este modelo de gestión de eventos en línea podemos conseguirlo retornando un valor de false. Observa el ejemplo:

```
<a href="…url…" onclick="return confirmar();"> Ir a … </a>
```

La idea es hacer que la función confirmar sea booleana. S**i al final el gestor de eventos se evalúa como *false* el evento no será disparado** y el comportamiento por defecto no se realizará. En este ejemplo sencillo no cargaremos la url del enlace.

```
function confirmar() { return window.confirm("¿Desea visitar el
enlace?"); }
```

Este mismo mecanismo es el que suele emplearse al realizar la validación en el cliente. Para eso hacemos:

```
<form …. onsubmit="return validarDatos();">
```

Y la función validarDatos() será una función booleana. **Si retorna true es que los datos son correctos y se producirá el submit del formulario. Si retorna false porque se encontró algún error en los datos no se aceptará el formulario**.

Modelo de gestión de eventos tradicional o semántico

El modelo de gestión de eventos en línea visto anteriormente tiene una desventaja importante. La tendencia natural en el desarrollo web estriba en la separación de los códigos html, css y javascript.

Esto se conoce como la **web semántica** donde el HTML sólo debe guardar la estructura semántica de la información (es el principio básico en el que se formuló el HTML5). Será CSS quien se encargue de formatear la misma información para presentarla en diversos medios. Y JavaScript operará con la información básica de partida para realizar transformaciones. Las tres capas de trabajo deben estar separadas.

La página HTML sólo debe contener información Html.
A lo sumo se utilizan atributos id o class para enganchar los estilos.
Todo el código CSS estará separado en ficheros externos.
Todo el código JavaScript estará separado en ficheros externos.
No habrá código JavaScript insertado dentro de html.

Para evitar esto surgió el modelo eventos tradicional o semántico, basado en el uso del DOM, y en concreto en su objeto **events**. Ahora los eventos son propiedades de los objetos del DOM. Y los manejadores de eventos pueden ser fijados directamente en el código JS. Observa el ejemplo, partimos de un código HTML puro:

Código HTML:

```
<input type="button" id="btnCalculo" value="Calcular"/>
```

Y agregamos el evento en el código de JS, sin contaminar el código HTML, éste puede ejecutarse por ejemplo al cargarse el documento.

```
function realizaCalculos() { Aquí realizamos los cálculos necesarios };
var boton = document.getElementById("btnCalculo");
boton.onclick = realizaCalculos; // aquí asignamos la función manejadora del evento.
```

//OJO: no usar paréntesis en el nombre de la función manejador.

Este modelo semántico presenta ventajas evidentes.

- La primera la hemos comentado ya, todo el código JS estará separado del código HTML y puede residir en un fichero externo.
- La segunda ventaja es que también podemos eliminar un evento muy fácilmente. En nuestro ejemplo anterior bastaría hacer:

```
var boton = document.getElementById("btnCalculo");
boton.onclick = null; // aquí eliminamos el evento retirando la función manejadora (event handler)
```

- La tercera ventaja es que el manejador de eventos se convierte en un método del objeto. Por lo tanto para invocarlo bastaría con llamar al método:

```
var boton = document.getElementById("btnCalculo");
boton.onclick(); // aquí disparamos el evento y por lo tanto ejecutaremos la función realizaCalculos()
```

- La cuarta ventaja es también muy importante. Al ser el evento un método del propio objeto, dentro de la función manejadora podemos utilizar this para obtener una referencia al propio objeto. Es decir, en el ejemplo anterior dentro de la función realizaCalculos() podemos utilizar this y representará al <input type="button" id="btnCalculo"/>

this: Cuando trabajamos con objetos representa a la instancia actual. Aplicando esta filosofía al modelo de eventos representa el elemento XHTML que ha producido el evento.

Y en el diseño web moderno también hay una nueva REGLA: **las tablas sólo se usan para presentar datos cuya naturaleza sea tabular, nunca para maquetar**.

Deficiencias del modelo de gestión semántico.

El modelo de gestión de eventos semántico aprendido en el apartado anterior nos ofrecía varias ventajas. Las más destacadas eran la separación del código JS del HTML y la posibilidad de usar this dentro de la función manejadora del evento (llamada **event handler**).

La primera de ellas se produce cuando queremos aplicar a un objeto varias veces el mismo evento. Con el modelo tradicional sólo podemos hacer:

```
objeto.onclick = primerafuncionalidad;
```

Y NOOOOO podríamos hacer lo siguiente:

```
objeto.onclick = segundafuncionalidad; //Se sobreescribe la
propiedad
```

Bueno, pues puedo hacer una función que incluya a las dos y ya está solucionado el probema

```
function funciongenerica() {
      primerafuncionalidad(); // llamada al primer manejador
      segundafuncionalidad(); // llamada al segundo manejador
}
objeto.onclick = funciongenerica; // event handler del evento onclick
```

PEEEEEEROOOOOOO, y si yo sólo quiero que funcione una de las dos en determinadas ocasiones…

```
objeto.onclick = null; //Esto desactivaría las dos funcionalidades a
la vez.
```

1ª Limitación del Modelo tradicional o semántico: Un evento sólo podrá ser gestionado por un manejador (event handler).

Y existe además una segunda limitación del modelo tradicional que tiene que ver con la propagación y cancelación de eventos. Imagina que tenemos un div y dentro colocamos un botón. Ambos además están dentro de la ventana.

Actividad 1

Crea en una página un div y dentro un botón. En JavaScript se asignamos los siguientes eventos:

```
document.getElementById("midiv").onclick = pulsadoDiv;
document.getElementById("miboton").onclick = pulsadoBoton;
window.onclick = pulsadoVentana;
```

Las funciones pulsadoDiv, pulsadoBoton y pulsadoVentana sólo harán un alert indicando "se ha pulsado el div", "se ha pulsado el botón" y "se ha pulsado la ventana".

¿Qué pasaría cuando pulsamos el botón? Indique la secuencia de mensajes que se producen en Chrome, Firefox, Ópera y en IE si lo tiene instalado.

2ª Limitación del Modelo tradicional o semántico: No hay control sobre el flujo de eventos.

El flujo de eventos.

En el ejemplo anterior hemos visto cómo cada navegador puede implementar su propio modelo de gestión del flujo de los eventos. Cuando la guerra entre navegadores estaba entre Netscape y Microsoft cada uno decidió tomar un modelo diferente:

Netscape se decantó por desencadenar primero el evento en el elemento más externo. Así se produciría el evento en la ventana, después en el div y por último en el botón. A esto se le llamó **modelo de captura**. El evento se dispara en el exterior y se va propagando (capturando) en el interior.

Microsoft por el contrario se decantó por el camino inverso. Primero reacciona el botón y el evento burbujea hacia el div y de éste burbujea hacia la ventana. Es lo que se conoce como **modelo de bubbling (burbujeo).** El evento se dispara en el interior y se va propagando (burbujeando) hacia el exterior.

Los navegadores posteriores intentaron con mayor o menor éxito implementar los dos modelos. Mozilla (del que Firefox es un ejemplo) se decantó por el bubbling pero propagándolo hasta el objeto window, mientras que IE utiliza también bubling pero lo propaga sólo hasta el objeto document (edge propaga hasta la ventana). Sólo Netscape (ya extinto) se decantó por el modelo de captura.

El modelo de registro avanzado.

Este modelo trata de superar las dos limitaciones explicadas para el modelo semántico, que eran: imposibilidad de asignar más de un manejador al mismo evento y permitir un control básico en el flujo de eventos.

En este modelo los eventos se registran mediante un método nuevo llamado **addEventListener**. Y se pueden eliminar mediante el método **removeEventListener**.

Esta forma de funcionamiento acerca JS a la gestión de eventos realizada en otros lenguajes como es el caso de Java.

SINTAXIS

```
objeto.addEventListener("tipoevento", manejador, cuando);
objeto.removeEventListener("tipoevento", manejador, cuando);
```

El primer parámetro es una cadena con el nombre del evento pero ojo: sin la palabra "on". Ejemplos: "click", "mouseover", "focus".

- El segundo parámetro es la función que actuará como event handler.
- El tercer parámetro es un parámetro booleano que trata de indicar cuándo se atenderá al evento. Tiene que ver con el flujo de eventos descrito.
    - Valor true: se producirá en la fase de captura (descendente)
    - Valor false: se producirá en la fase de bubbling (ascendente)

Veamos un ejemplo:

```
elemento.addEventListener("click", realizarCalculos, false);
elemento.addEventListener("click", ocultarDatos, false);
elemento.addEventListener("click", hacerConsulta, false);
elemento.addEventListener("change", validarElemento, false);
```

En este ejemplo hemos agregado a elemento un total de cuatro manejadores, los tres primeros para el evento onclick y el último para el evento onchange. Como ves hemos superado la primera limitación del modelo semántico porque en alguna otra parte del código si así lo deseamos podríamos eliminar la funcionalidad hacerConsulta:

```
elemento.removeEventListener("click", hacerConsulta, true);
```

Y el resto del evento onclick no se ve afectado. Siguen activos los manejadores realizarCalculos y ocultarDatos.

Dentro de los manejadores el uso de this sigue funcionando exactamente igual. Mantiene una referencia al elemento que disparó el evento.

Cuando se empezó a utilizar este modelo de registro avanzado inmediatamente los programadores encontraron que sería necesario poder consultar cuántos manejadores tiene activo un elemento.

En el modelo semántico para saber si un objeto tiene un evento activo bastaba con preguntar: `if (objeto.onclick == undefined)` en este caso el objeto no tiene evento onclick.
En este modelo un mismo objeto puede tener varios manejadores para el evento onclick. ¿Cómo sabemos cuáles son? El W3C en la implementación del DOM de nivel 3 incluyó un nuevo método llamado **eventListenerList**. Este método devuelve una lista con referencias a todas las funciones manejadoras. Esto sí, recuerda que no todos los navegadores implementan completamente el DOM de nivel 3, con lo que puede que este método no esté disponible.

El modelo de registro avanzado en IE.

Aunque prácticamente todos los navegadores se ajustaron al modelo de registro de eventos propuesto por el W3C e implementaron los métodos **addEventListener** y **removeListener**, ocurre que *Microsoft desarrolló su propio modelo de registro*.
Se trata de un modelo completamente parecido pero utiliza métodos con distintos nombres:

En lugar de addEventListener se utiliza **attachEvent(evento, funcion)** y en lugar de removeEventListener se utiliza **detachEvent(evento, funcion)**. Ambos métodos reciben sólo los dos primeros parámetros.

NOTA: en el primer parámetro sí que incluimos la palabra on en el nombre del evento.

Ejemplos:

```
objeto.attachEvent("onclick", realizarCalculos);
objeto.detachEvent("onclick", realizarCalculos);
```

No existe el tercer parámetro porque en IE los eventos siempre utilizan la propagación por burbujeo (del interior hacia el exterior). No existe la fase de captura.

Otra diferencia es que dentro de las funciones manejadoras del evento en IE no contamos con la referencia this, que apunta siempre al objeto window

Resumen de diferencias en IE

El nombre del evento sí que incluye la palabra "on". Ej: "onclick"
Todos los eventos se generan mediante burbujeo (del interior al exterior).
No podemos detectar quién disparó el evento con this.

Para soportar todos los navegadores (solución cross-browser). La diferencia estará en utilizar addEventListener o attachEvent cuando proceda.

if (unobjeto) {…}: el motor de JS no produce errores si un objeto no existe. Simplemente lo considera con valor **undefined** y hace una equivalencia a false para no ejecutar el código dentro del if. Esto es lo que se conoce como detección de objetos.

Ahora crearemos una función genérica utilizando este mecanismo:

```
function agregaEvento(evento, objeto, funcion) {
  if (objeto.addEventListener) //estamos en un DOM del estándar
    objeto.addEventListener(evento,funcion,false);
  else if (objeto.attachEvent) // estamos en un DOM de IExplorer
    objeto.attachEvent("on"+evento, funcion);
  else // no estamos en un DOM ni W3C ni de IExplorer? modelo tradicional
    objeto["on"+evento] = funcion; // recuerda: objetos son arrays asociativos
}
// aquí agregamos el evento onload a la ventana.
    agregaEvento("load", window, configuraEventos);
//Y aquí agregamos onclick al botón id="btn" manejado por la función mifuncion():
    miobjeto = document.getElementById("btn");
    agregaEvento("click", miobjeto, mifuncion);
```

Controlando la propagación.

Ya hemos visto la diferencia entre el modelo de captura (el evento se genera primero en el elemento más externo) y el modelo de burbujeo (el evento se genera primero en el elemento más interno y se propaga hacia afuera).

Por otra parte hemos visto que IE tiene su propio sistema y que siempre utiliza burbujeo.

Para el resto de navegadores es posible fijar el modelo (captura o burbujeo) utilizando el tercer parámetro del método addEventListener. Si es true se utiliza captura, y si es false se utiliza burbujeo.

Actividad 2

Utiliza el siguiente código:

```
<div id="principal">
        Este es el contenedor principal
        <div id="secundario">
                Este es el contenedor segundario
                <input id="miboton" type="button" value="OK">
        </div>
</div>
```

Y el .js:

```
window.onload = function() {
        document.getElementById("principal").addEventListener("click", function () {
                alert("Pulsado el contenedor principal");
        }, false); // estamos utilizando burbujeo
        document.getElementById("secundario").addEventListener("click", function () {
                alert("Pulsado el contenedor secundario");
        }, false); // estamos utilizando burbujeo
        document.getElementById("miboton").addEventListener("click", function () {
                alert("Pulsado el botón");
        }, false); // estamos utilizando burbujeo
}
```

Prueba que funciona el código, el método del burbujeo. Luego cambia el método de burbujeo por el de captura.

Detener el burbujeo

Los eventos gestionados con captura no permiten parar el flujo de la propagación. Sin embargo cuando trabajamos con el burbujeo sí que podemos detenerlos utilizando el método **stopPropagation**() del objeto **event**. En IE este método sólo existe desde la versión 9. Y una buena noticia: desde IE9 se admiten **addEventListener** y **removeEventListener**!!!

Por otra parte debes saber que no todos los eventos provocan burbujeo. El evento onclick sí lo hace (y lo hemos probado). Pero el evento onfocus no, sólo afecta al elemento que recibe el foco y no se propaga.

stopPropagation() debemos conocer que es un método del objeto event. En JS podemos acceder a este objeto como un argumento que se pasa de forma automática a las funciones manejadoras.

Ejemplo:

```
miobjeto.addEventListener("keyDown", manejador, false);
function manejador(event) { // dentro de esta función el objeto event representa al evento.
```

```
        event.stopPropagation();
}
```

Actividad 3

Añade el siguiente código a la actividad 2 y comprueba que funciona ¿qué hace?

```
document.getElementById("secundario").addEventListener("click",
        function (e) {
              alert("Pulsado el contenedor secundario y paramos la
        propagación");
              e.stopPropagation();
}, true);
```

El objeto event.

Este objeto se asigna automáticamente como el único parámetro de las funciones manejadoras. Podemos acceder a él indicando un parámetro en la función manejadora o bien accediendo con **arguments[0]**. El objeto event aporta información necesaria sobre el evento: quién lo ha producido, qué tecla se ha pulsado, qué posición tenía el ratón, etc.

Ya hemos visto su método **stopPropagation**(). Pero este objeto cuenta con otras propiedades interesantes. De nuevo encontramos que los navegadores que utilizan el DOM delW3C presentan una estructura en el objeto event diferente a I.E.

Otro método muy utilizado de este objeto es **preventDefault**(). Permite cancelar un evento (si se trata de un evento cancelable). Esto significa que no se realizará la tarea que generalmente se espera. Por ejemplo al pinchar un enlace se espera que carguemos la url. Con **preventDefault**() evitaremos esta carga. Al aceptar un formulario se espera que se realice una petición al servidor solicitando el recurso del atributo action. Este método también podrá abortar este comportamiento.

Propiedades definidas por DOM

La siguiente tabla recoge las propiedades definidas para el objeto event en los navegadores que siguen los estándares:

Propiedad/Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `ALT` y `false` en otro caso
`bubbles`	Boolean	Indica si el evento pertenece al flujo de eventos de *bubbling*
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: `0` – Ningún botón pulsado `1` – Se ha pulsado el botón izquierdo `2` – Se ha pulsado el botón derecho `3` – Se pulsan a la vez el botón izquierdo y el derecho `4` – Se ha pulsado el botón central `5` – Se pulsan a la vez el botón izquierdo y el central `6` – Se pulsan a la vez el botón derecho y el central `7` – Se pulsan a la vez los 3 botones

Propiedad/Método	Devuelve	Descripción
`cancelable`	Boolean	Indica si el evento se puede cancelar
`cancelBubble`	Boolean	Indica si se ha detenido el flujo de eventos de tipo *bubbling*
`charCode`	Número entero	El código unicode del carácter correspondiente a la tecla pulsada
`clientX`	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
`clientY`	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
`ctrlKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `CTRL` y `false` en otro caso
`currentTarget`	Element	El elemento que es el objetivo del evento
`detail`	Número entero	El número de veces que se han pulsado los botones del ratón
`eventPhase`	Número entero	La fase a la que pertenece el evento: `0` – Fase capturing `1` – En el elemento destino `2` – Fase bubbling
`isChar`	Boolean	Indica si la tecla pulsada corresponde a un carácter
`keyCode`	Número entero	Indica el código numérico de la tecla pulsada
`metaKey`	Número entero	Devuelve `true` si se ha pulsado la tecla `META` y `false` en otro caso
`pageX`	Número entero	Coordenada X de la posición del ratón respecto de la página
`pageY`	Número entero	Coordenada Y de la posición del ratón respecto de la página
`preventDefault()`	Función	Se emplea para cancelar la acción predefinida del evento
`relatedTarget`	Element	El elemento que es el objetivo secundario del evento (relacionado con los eventos de ratón)
`screenX`	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
`screenY`	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
`shiftKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `SHIFT` y `false` en otro caso
`stopPropagation()`	Función	Se emplea para detener el flujo de eventos de tipo *bubbling*
`target`	Element	El elemento que origina el evento
`timeStamp`	Número	La fecha y hora en la que se ha producido el evento

Propiedad/Método	Devuelve	Descripción
`type`	Cadena de texto	El nombre del evento

Al contrario de lo que sucede con Internet Explorer, la mayoría de propiedades del objeto `event` de DOM son de sólo lectura. En concreto, solamente las siguientes propiedades son de lectura y escritura: **`altKey`**, **`button`** y **`keyCode`**.

La tecla `META` es una tecla especial que se encuentra en algunos teclados de ordenadores muy antiguos. Actualmente, en los ordenadores tipo PC se asimila a la tecla `Alt` o a la *tecla de Windows*, mientras que en los ordenadores tipo Mac se asimila a la tecla `Command`.

currentTarget: es una propiedad MUY interesante. Nos retorna una referencia al elemento que es el objetivo del evento. En la especificación de ActionScript 3.0 también se utiliza esta propiedad. Target almacena una referencia al elemento que originó el evento. En los navegadores de la familia de IE no existen estas propiedades y se utiliza **srcElement**.

Propiedades definidas por Internet Explorer

La siguiente tabla recoge las propiedades definidas para el objeto event en los navegadores de la familia Internet Explorer:

Propiedad/Método	Devuelve	Descripción
`altKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `ALT` y `false` en otro caso
`button`	Número entero	El botón del ratón que ha sido pulsado. Posibles valores: `0` – Ningún botón pulsado `1` – Se ha pulsado el botón izquierdo `2` – Se ha pulsado el botón derecho `3` – Se pulsan a la vez el botón izquierdo y el derecho `4` – Se ha pulsado el botón central `5` – Se pulsan a la vez el botón izquierdo y el central `6` – Se pulsan a la vez el botón derecho y el central `7` – Se pulsan a la vez los 3 botones
`cancelBubble`	Boolean	Si se establece un valor `true`, se detiene el flujo de eventos de tipo *bubbling*
`clientX`	Número entero	Coordenada X de la posición del ratón respecto del área visible de la ventana
`clientY`	Número entero	Coordenada Y de la posición del ratón respecto del área visible de la ventana
`ctrlKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `CTRL` y `false` en otro caso

Propiedad/Método	Devuelve	Descripción
`fromElement`	Element	El elemento del que sale el ratón (para ciertos eventos de ratón)
`keyCode`	Número entero	En el evento `keypress`, indica el carácter de la tecla pulsada. En los eventos `keydown` y `keyup` indica el código numérico de la tecla pulsada
`offsetX`	Número entero	Coordenada X de la posición del ratón respecto del elemento que origina el evento
`offsetY`	Número entero	Coordenada Y de la posición del ratón respecto del elemento que origina el evento
`repeat`	Boolean	Devuelve `true` si se está produciendo el evento `keydown` de forma continuada y `false` en otro caso
`returnValue`	Boolean	Se emplea para cancelar la acción predefinida del evento
`screenX`	Número entero	Coordenada X de la posición del ratón respecto de la pantalla completa
`screenY`	Número entero	Coordenada Y de la posición del ratón respecto de la pantalla completa
`shiftKey`	Boolean	Devuelve `true` si se ha pulsado la tecla `SHIFT` y `false` en otro caso
`srcElement`	Element	El elemento que origina el evento
`toElement`	Element	El elemento al que entra el ratón (para ciertos eventos de ratón)
`type`	Cadena de texto	El nombre del evento
`x`	Número entero	Coordenada X de la posición del ratón respecto del elemento padre del elemento que origina el evento
`y`	Número entero	Coordenada Y de la posición del ratón respecto del elemento padre del elemento que origina el evento

Listado de eventos:

De ratón

Evento	Descripción
`click`	Se produce cuando se pulsa el botón izquierdo del ratón. También se produce cuando el foco de la aplicación está situado en un botón y se pulsa la tecla `ENTER`
`dblclick`	Se produce cuando se pulsa dos veces el botón izquierdo del ratón
`mousedown`	Se produce cuando se pulsa cualquier botón del ratón
`mouseout`	Se produce cuando el puntero del ratón se encuentra en el interior de un elemento y el usuario mueve el puntero a un lugar fuera de ese elemento
`mouseover`	Se produce cuando el puntero del ratón se encuentra fuera de un elemento y el usuario mueve el puntero hacia un lugar en el interior del elemento

Evento	Descripción
mouseup	Se produce cuando se suelta cualquier botón del ratón que haya sido pulsado
mousemove	Se produce (de forma continua) cuando el puntero del ratón se encuentra sobre un elemento

Eventos de teclado

Evento	Descripción
keydown	Se produce cuando se pulsa cualquier tecla del teclado. También se produce de forma continua si se mantiene pulsada la tecla
keypress	Se produce cuando se pulsa una tecla correspondiente a un carácter alfanumérico (no se tienen en cuenta telas como SHIFT, ALT, etc.). También se produce de forma continua si se mantiene pulsada la tecla
keyup	Se produce cuando se suelta cualquier tecla pulsada
Evento	Descripción
load	Se produce en el objeto window cuando la página se carga por completo. En el elemento <img> cuando se carga por completo la imagen. En el elemento <object> cuando se carga el objeto
unload	Se produce en el objeto window cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo). En el elemento <object> cuando desaparece el objeto.
abort	Se produce en un elemento <object> cuando el usuario detiene la descarga del elemento antes de que haya terminado
error	Se produce en el objeto window cuando se produce un error de JavaScript. En el elemento <img> cuando la imagen no se ha podido cargar por completo y en el elemento <object> cuando el elemento no se carga correctamente
select	Se produce cuando se seleccionan varios carácteres de un cuadro de texto (<input> y <textarea>)
change	Se produce cuando un cuadro de texto (<input> y <textarea>) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento <select>
submit	Se produce cuando se pulsa sobre un botón de tipo submit (<input type="submit">)
reset	Se produce cuando se pulsa sobre un botón de tipo reset (<input type="reset">)
resize	Se produce en el objeto window cuando se redimensiona la ventana del navegador
scroll	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento <body> contiene la barra de scroll de la página completa
focus	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento obtiene el foco

Evento	Descripción
`blur`	Se produce en cualquier elemento (incluido el objeto window) cuando el elemento pierde el foco

Eventos HTML

Evento	Descripción
`load`	Se produce en el objeto `window` cuando la página se carga por completo. En el elemento `<img>` cuando se carga por completo la imagen. En el elemento `<object>` cuando se carga el objeto
`unload`	Se produce en el objeto `window` cuando la página desaparece por completo (al cerrar la ventana del navegador por ejemplo). En el elemento `<object>` cuando desaparece el objeto.
`abort`	Se produce en un elemento `<object>` cuando el usuario detiene la descarga del elemento antes de que haya terminado
`error`	Se produce en el objeto `window` cuando se produce un error de JavaScript. En el elemento `<img>` cuando la imagen no se ha podido cargar por completo y en el elemento `<object>` cuando el elemento no se carga correctamente
`select`	Se produce cuando se seleccionan varios carácteres de un cuadro de texto (`<input>` y `<textarea>`)
`change`	Se produce cuando un cuadro de texto (`<input>` y `<textarea>`) pierde el foco y su contenido ha variado. También se produce cuando varía el valor de un elemento `<select>`
`submit`	Se produce cuando se pulsa sobre un botón de tipo submit (`<input type="submit">`)
`reset`	Se produce cuando se pulsa sobre un botón de tipo reset (`<input type="reset">`)
`resize`	Se produce en el objeto `window` cuando se redimensiona la ventana del navegador
`scroll`	Se produce en cualquier elemento que tenga una barra de scroll, cuando el usuario la utiliza. El elemento `<body>` contiene la barra de scroll de la página completa
`focus`	Se produce en cualquier elemento (incluido el objeto `window`) cuando el elemento obtiene el foco
`blur`	Se produce en cualquier elemento (incluido el objeto `window`) cuando el elemento pierde el foco

Eventos DOM

Aunque los eventos de este tipo son parte de la especificación oficial de DOM, aún no han sido implementados en todos los navegadores. La siguiente tabla recoge los eventos más importantes de este tipo:

Evento	Descripción
`DOMSubtreeModified`	Se produce cuando se añaden o eliminan nodos en el subárbol de un documento o elemento
`DOMNodeInserted`	Se produce cuando se añade un nodo como hijo de otro nodo
`DOMNodeRemoved`	Se produce cuando se elimina un nodo que es hijo de otro nodo
`DOMNodeRemovedFromDocument`	Se produce cuando se elimina un nodo del documento
`DOMNodeInsertedIntoDocument`	Se produce cuando se añade un nodo al documento

Validación y expresiones regulares

Carácter	Significado
\	Buscará coincidencias conforme a las siguientes reglas: Una barra invertida precediendo un carácter simple indica que éste debe ser interpretado como un carácter especial y no de forma literal. Por ejemplo, una 'b' sin '\' precediéndole coincidirá con cualquier 'b' minúscula en la cadena, sin embargo, '\b' no coincidirá con algún carácter en específico, representará el delimitador especial de palabras. Una barra invertida que precede a un carácter especial indica que éste deberá ser interpretado literalmente, esto es, como un carácter simple y no como un carácter especial. A esto se le denomina `escapado`. Por ejemplo, en el patrón '/a*/' el '*' indica que se deberá buscar una secuencia de 'a' cero o más veces, por el contrario, el cambiar el patrón a '/a*/', el carácter especial es interpretado como un carácter simple, y cadenas como 'a*' harán coincidencia. No se olvide de *escapar* la propia barra invertida al usarla en expresiones regulares con strings - RegExp("patron") - ya que la \ es un carácter de escapado en strings.
^	Coincide con el principio de la entrada. Si la bandera de multilínea está activada, también coincidirá inmediatamente después de un salto de línea. Por ejemplo, `/^A/` no coincide con la 'A' en "an A", pero sí con la 'A' en "An E". El carácter '^' tiene un significado diferente cuando aparece como el primer carácter en un patrón. Véase patrones complementarios para mayores detalles y ejemplos.

Carácter	Significado
$	Busca el final de la entrada. Si la bandera de multilínea se establece en true, también buscará inmediatamente antes de un carácter de salto de línea. Por ejemplo, la expresión `/r$/` no encontrará el carácter 'r' en la cadena "cenaremos", pero sí la encontrará en la cadena "cenar".
*	Busca el carácter precedente 0 (cero) o más veces. Es equivalente a {0,}. Por ejemplo, la expresión `/bo*/` encontrará la subcadena 'boooo' en la cadena "A ghost booooed" y el carácter 'b' en la cadena "A bird warbled", pero no encontrará nada en la cadena "A goat grunted".
+	Busca el carácter precedente 1 o más veces. Es equivalente a {1,}. Por ejemplo, la expresión `/u+/` encontrará el carácter 'u' en la cadena "dulce" y todos los carácteres 'u' en la cadena "duuuuulce".
?	Busca el carácter precedente 0 (cero) o 1 (una) vez. Es equivalente a `{0,1}`. Por ejemplo, la expresión `/e?le?/` encontrará la subcadena 'el' en la cadena "angel" y la subcadena 'le' en la cadena "angle" y también el carácter 'l' en la cadena "oslo". Si se utiliza inmediatamente después que cualquiera de los cuantificadores *, +, ?, o {}, hace que el cuantificador no sea expansivo (encontrando la menor cantidad posible de carácteres), en comparación con el valor predeterminado, que sí es expansivo (encontrando tantos carácteres como le sea posible). Por ejemplo, aplicando la expresión `/\d+/` a la cadena "123abc" encuentra "123". Pero aplicando la expresión `/\d+?/` a la misma cadena, encuentra solamente el carácter "1". También se utiliza en coincidencias previsivas, como se describe en las entradas `x(?=y)` y `x(?!y)` de esta tabla.
.	(El punto decimal) coincide con cualquier carácter excepto un carácter de nueva linea. Por ejemplo , `/.n/` coincide 'an' y 'on' en "nay, an apple is on the tree", pero no 'nay'.
(x)	Busca 'x' y recuerda la busqueda, como el siguiente ejemplo lo muestra. Los parentesis son llamados *parentesis de captura*. El '`(foo)`' y '`(bar)`' en el patron `/(foo) (bar) \1 \2/` busca y recuerda las primeras dos palabras en el string "foo bar foo bar". El `\1 y \2` en el patron coincida las dos últimas palabras de la cadena. Nota que `\1, \2, \n` son usados en la parte donde se define la expresión regular. Cuando se usan en la parte de reemplazo, se debe usar la sintaxis `$1, $2, $n` en su lugar; `'bar foo'.replace( /(...) (...)/, '$2 $1' )`.

Carácter	Significado
(?:x)	Coincide con 'x' pero no recuerda la coincidencia. Los paréntesis son llamados paréntesis no capturadores, y permiten definir subexpresiones para manipular con los operadores de las expresiones regulares. Considera la expresión de ejemplo `/(?:foo){1,2}/`. Si la expresión fuera /foo{1,2}/, los carácteres `{1,2}` se aplicarían sólo a la última 'o' en 'foo'. Con los paréntesis no capturadores, `{1,2}` se aplica a la palabra entera 'foo'.
x(?=y)	Coincide con 'x' sólo si 'x' es seguida por 'y'. Esto se denomina previsión (lookahead, mirar adelante). Por ejemplo, `/Jack(?=Sprat)/` coincide con 'Jack' solo si es seguido por 'Sprat'. `/Jack(?=Sprat|Frost)/` conicide con 'Jack' solo si es seguido por 'Sprat' o 'Frost'. Sin embargo, ni 'Sprat' ni 'Frost' serán parte del resultado.
x(?!y)	Coincide con 'x' solo si 'x' no es seguida por 'y'. Es una previsión negativa. Por ejemplo, `/\d+(?!\.)/` coincide con numeros solo si no vienen seguidos por un punto decimal. La expresion regular `/\d+(?!\.)/.exec("3.141")` coincide con '141' pero no con '3.141'.
x|y	Coincide con 'x' o 'y'. Por ejemplo, `/green|red/` coincide con 'green' en "green apple" y 'red' en "red apple."
{n}	Coincide exactamente con n ocurrencias de la expreción. N debe ser un entero positivo. Por ejemplo, `/a{2}/` no coincide con la 'a' en "candy," pero si con las a de "caandy," y las 2 primeras a en "caaandy."
{n,m}	Donde `n` y `m` son enteros positivos y n <= m. Coincide con al menos `n` y no más de `m` ocurrencias de la expreción. Si se omite `m`, no tiene limite de máximo. Por ejemplo, `/a{1,3}/` no coincide con "cndy", pero sí con la 'a' en "candy," las primeras 2 a en "caandy," y las primeras 3 `a` en "caaaaaaandy". Note que en "caaaaaaandy", la coincidencia es "aaa", aunque la cadena contenga más `a` en ella.
[xyz]	Grupo de carácteres. Este tipo de patrón coincide con cada carácter dentro de los corchetes, incluyendo secuencias de escapado. Carácteres especiales como el punto (`.`) y el asterisco (`*`) no son especiales en un grupo, así que no necesitan ser escapados. Puede especificar un rango utilizando un guión, como en el siguiente ejemplo. El patrón `[a-d]`, que equivale a `[abcd]`, coincide con la 'b' en "brisket" y la 'c' in "city". El patrón `/[a-z.]+/` y `/[\w.]+/` coinciden con toda la cadena "test.i.ng".

Carácter	Significado
[^xyz]	Grupo de carácteres negativo. Significa que coincide con cualquier cosa que no esté en los corchetes. Puede especificar rangos. Todo lo que funciona en el grupo de carácteres positivo funciona también aquí. Por ejemplo, `[^abc]` es lo mismo que `[^a-c], y` coincide con la 'r' en "brisket" y 'h' en "chop."
[\b]	Coincide con backspace (U+0008). Debe ir entre corchetes. (No confundir con `\b`.)
\b	Coincide con un *limite de palabra*. Un *limite de palabra* coincide con la posicion entre donde un carácter de palabra no viene precedido o seguido por otro. Notese que el límite no estará incluido en la coincidencia. En otras palabras, la longitud del limite es cero. (No confundir con `[\b]`.) Ejemplos: `/\bm/` coincide con la 'm' de "moon" ; `/oo\b/` no tiene coincidencias en "moon", porque las 'oo' estan seguidas de una 'n' que es un carácter de palabra; `/oon\b/` coincide con 'oon' en "moon", porque 'oon' es el final de la cadena, por lo cual no va seguido de un carácter de palabra; `/\w\b\w/` no coincidirá con nada, porque un carácter de palabra no puede estar seguido por ambos, un limite y un carácter de palabra.
\B	Coincide con un *no-limite de palabra*. Esto coincide con una posicion donde el anterior y el siguiente carácter son del mismo tipo: ambos son o no son carácteres de palabra. El inicio y el final de una cadena se consideran *no palabras*. Por ejemplo, `/\B../` coincide con 'oo' en "noonday", y `/y\B./` matches 'ye' in "possibly yesterday."
\cX	Donde *X* es un carácter entre A y Z. Coincide con un carácter de control en un string. Por ejemplo, `/\cM/` coincide con control-M (U+000D) en un string.
\d	Coincide con un carácter de número. Equivalente a `[0-9]`. Por ejemplo, `/\d/` or `/[0-9]/` coinciden con el '2' en "B2 is the suite number."
\D	Coincide con cualquier carácter no numérico. Equivalente a `[^0-9]`. Por ejemplo, `/\D/` or `/[^0-9]/` coincide con la 'B' en "B2 is the suite number."
\f	Coincide con un form feed (salto de página) (U+000C).
\n	Coincide con un line feed (salto de linea) (U+000A).
\r	Coincide con un carriage return (retorno de carro) (U+000D).
\s	Coincide con un *carácter de espacio*, entreellos incluidos espacio, tab, salto de página, salto de linea y retorno de carro. Equivalente a `[ \f\n\r\t\v`

Carácter	Significado
	`\u00a0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006 \u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]`. Por ejemplo, `/\s\w*/` coincide con ' bar' en "foo bar."
\S	Coincide con todo menos *carácteres de espacio*. Equivalente a `[^ \f\n\r\t\v\u00a0\u1680\u180e\u2000\u2001\u2002\u2003\u2004 \u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f \u3000]`. Por ejemplo, `/\S\w*/` coincide con 'foo' en "foo bar."
\t	Coincide con tab (U+0009).
\v	Coincide con tab vertical (U+000B).
\w	Coincide con cualquier carácter alfanumérico, incluyendo el guión bajo. Equivalente a `[A-Za-z0-9_]`. Por ejemplo, `/\w/` coincide con 'a' en "apple," '5' en "$5.28," y '3' en "3D."
\W	Coincide con todo menos *carácteres de palabra*. Equivalente a `[^A-Za-z0-9_]`. Por ejemplo, `/\W/` o `/[^A-Za-z0-9_]/` coinciden con '%' en "50%."
n	Cuando *n* es un entero positivo, es una referencia hacia alguna subcadena de parentesis dentro de la misma expresion que coincida con el número (contando los parentesis izquierdos). Por ejemplo, `/apple(,)\sorange\1/` coincide con 'apple, orange,' en "apple, orange, cherry, peach."
\0	Coincide con el carácter NULL (U+0000). No preseda este por otro número, ya que `\0<numero>` se considera una secuencia octal escapada.
\xhh	Coincide con un carácter en exadecimal hh (dos dígitos hexadecimales)
\uhhhh	Coincide con un carácter unicode con el código hhhh (cuatro dígitos hexadecimales).

Ejemplos

`/[ao]$/` Cadenas que terminan con una “a”, o con una “o”.

`/[aeiou]/` Cadenas que tienen una vocal.

`/[A-Z][a-z][1-3]/` Cadenas que tienen una letra mayúscula, seguida de una letra minúscula seguida de un número del 1 al 3

`/[0-9]+/` Cadenas con uno o más dígitos

`/[A-z]/` Cadenas con una letra (sea mayúscula o minúscula).

`/[^357]/` Cualquier carácter que NO es el 3, el 5 o el 7

`/[^aeiouAEIOU]/` Cualquier carácter que NO es una vocal

`/(si|no|puede)/` Cadenas que contienen una de estas tres palabras “si”, “no”, o bien “puede”. Ojo: para buscar subcadenas no usamos corchetes sino paréntesis.

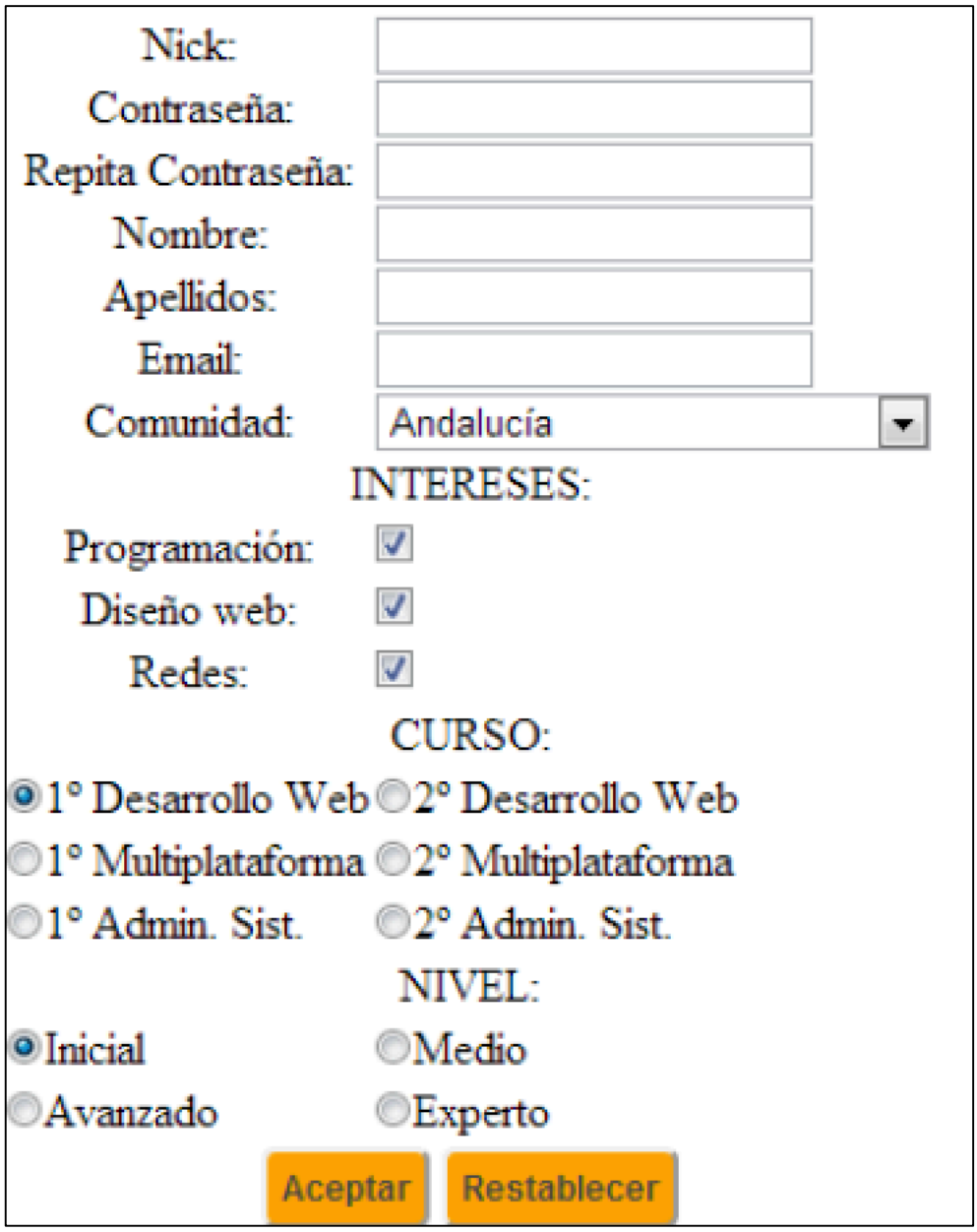
`/a{3}$/` Cadenas terminadas por 3 aes. “casaaa”, “peraaa”

`/^[aeiou]{2}$/i` Sólo dos vocales (da igual si mayúsculas o minúsculas).

`/^[A-z]{5}$/` Cinco letras (también mayúsculas o minúsculas).
`/[0-9]{3,}/` Cadenas que contengan 3 o más dígitos
`/(perro|gato){1,2}/` Cadenas con el texto "perro" o bien "gato" una o dos veces. Ejemplo: "Tenía un perro grande y un gato chico"
`/^ab.z$/` Cadenas formadas por una a seguida de una b, seguida de cualquier carácter y terminada por una z. "abaz", "ab5z", abdz",
`/sa?= de/` Cadenas con la cadena "sa" seguida por un espacio en blanco y "de". Ejemplo: La casa de Pedro
`/sa!= de/` Cadenas con la cadena "sa" que no esté seguida por " de". Ejemplo: La casa de Pedro tiene una mesa grande"

Actividad 4

Recuerdas el formulario de la anterior práctica, recupérala y haz una copia.



1. Modifica todos los eventos para que utilicen el modelo de registro avanzado que pueda funcionar en Internet Explorer, para ello añade la función configurarEventos() donde tendrás que agregar el código para asignar los eventos según el modelo de registro avanzado. Quita el código de tu html. Agrega al evento onload de la ventana esta función para que se ejecute al terminar de cargar la página.
2. En el formulario agrega el evento `onsubmit` la función validarDatos(), ésta irá comprobando cada elemento del formulario.

   Si el elemento está vacío escribiremos un mensaje en un `<div id="info"></div>`, pondremos el foco en el elemento (método focus()) y retornaremos `false` para evitar que se produzca el submit.

   Comprueba también que las contraseñas coincidan. Sólo si llegamos al final de la función validarDatos() sin haber encontrado un error retornaremos true para aceptar el formulario.

3. Si el usuario pasa por el campo Nick y se va a otro campo (es decir, el Nick ha perdido el foco) ya podemos validar si está vacío y mostrar la información pertinente. Podemos realizar una función llamada estaVacio() y asociarla al evento **onblur**. Dentro de la función para saber qué input la llamó utilizaremos this. Esto es posible sólo en esta forma de gestionar los eventos. programar estaVacio() y asociarla mediante eventos a todos los input del formulario. Nota: si además agregamos this.focus() (dependiendo del navegador puede variar su comportamiento) en estaVacio() el usuario no podrá abandonar el elemento sin haber escrito algo.
4. Utiliza una expresión regular para comprobar que el correo electrónico es una dirección válida.
5. Utiliza una expresión regular para comprobar que el nick
   a. Comienza por una letra
   b. Está escrito en minúsculas
   c. Se permiten números y guion medio (-) y guion bajo (_)

Fuentes

http://www.maestrosdelweb.com/que-es-javascript/
http://librosweb.es/libro/javascript/
http://librosweb.es/
http://www.aprenderaprogramar.es/index.php?option=com_content&view=article&id=788:tipos-de-datos-javascript-tipos-primitivos-y-objeto-significado-de-undefined-null-nan-ejemplos-cu01112e-&catid=78:tutorial-basico-programador-web-javascript-desde-&Itemid=206
https://www.todojs.com/ref/javascript/global/
https://jherax.wordpress.com/2014/07/08/javascript-poo-1/