Eventos 12

Javascript es un lenguaje basado en eventos. Es decir, cuando sucede 'algo' en nuestra página, el Javascript se activa y se ejecuta. Ese 'algo' puede ser cuando el usuario pase por encima de una imagen, haga click en un enlace, pulse una tecla, presione un botón... A esas acciones que suceden en nuestras páginas y que provocan la ejecución del Javascript se les denomina eventos.

En este tema vamos a ver cuales son los eventos que posee Javascript, como capturarlos y como manejarlos para que ejecuten el código que nosotros queramos.

12.1 Eventos en Javascript

Cada elemento HTML de nuestra página tiene asociado una serie de eventos que pueden realizarse sobre él. Algunos elementos tendrán un tipo de evento que no tengan otros y viceversa. O incluso varios elementos pueden tener eventos comunes.

El nombre de los eventos se forma añadiendo el prefijo `on` y a continuación el nombre de la acción que se realiza (en inglés). Así por ejemplo el evento `onclick` indica que el se va a ejecutar el código Javascript que nosotros indiquemos cuando se haga click sobre el elemento.

El ejemplo de uso más claro lo tenemos en nuestra forma de trabajar (explicada e el capítulo 2). Hasta ahora asociamos el evento `onclick` a un botón, es decir, 'cuando se pulse el botón', se va a ejecutar el código Javascript indicado:

```
<input type="button" value="pulsa" onclick="miFuncion()" >
```

Sin embargo, un elemento no tiene porque tener asociado sólo un evento. Como aprenderemos a lo largo de este capítulo, nosotros vamos a poder asociar tantos eventos como queramos a un elemento o a varios:

```
<input type="button" value="pulsa" onclick="miFuncion()"
                                   onmouseover = "miOtraFuncion()" >
```

En este ejemplo cuando hagamos click sobre el botón se ejecutara la función `miFuncion()` y cuando pasemos el ratón por encima del botón, se ejecutará `miOtraFuncion()`

Vamos a ver los eventos más importantes de Javascript:

Evento	Descripción	Elementos para los que está definido
`onblur`	Deseleccionar el elemento (perder el foco)	`<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 (ganar el foco)	`<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>`
`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

Evento	Descripción	Elementos para los que está definido
`oncontextmenu`	Cuando aparece el menú contextual al pulsar el botón derecho del ratón o la tecla correspondiente.	`<body>` y el documento.
`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>`

12.2 Manejadores de Eventos

Los manejadores de eventos (*event handlers* en inglés) son las formas que tenemos para indicar qué código se debe ejecutar cuando se produce un evento determinado.

Hasta ahora nosotros hemos estado controlando qué sucede cuando se produce el evento click a través de un atributo en el HTML (`onclick`). Sin embargo, Javascript nos proporciona esta y otras formas más adecuadas de capturar los eventos que suceden en nuestras páginas.

12.2.1 A través de atributos HTML

Esta es la forma más común de manejar eventos en Javascript y la que venimos usando hasta ahora. En el elemento deseado, colocamos como atributo de HTML el nombre del evento a manejar y a continuación decidimos que código es el encargado de ello.

Dependiendo de como pongamos ese código podemos tener dos subcategorías:

12.2.1.1 Forma Directa

El valor del atributo evento son instrucciones de Javascript. Si tenemos más de una instrucción, estas se separan por punto y coma (;):

```
<input type="button" value="pulsa" onclick="alert('hola');
document.write('adios')">
```

Como se puede observar en el ejemplo, en el momento que tengamos un par de sentencias dentro del atributo evento, el código va a empezar a complicarse y a no quedar bien estructurado.

Destacar que al estar colocando instrucciones dentro del valor de un atributo HTML y que este usa dobles comillas (“ ”) para englobarlas, ninguna instrucción de Javascript que pongamos puede hacer uso de las dobles comillas. En su lugar se deben usar comillas simples.

```
<input type="button" value="pulsa"
    onclick="alert('hola'); document.write('adios')">
```

Si quisiéramos hacer referencia al objeto donde se produce el evento (para cambiar propiedades CSS de ese elemento, por ejemplo), podríamos usar la sentencia `document.getElementById` como se vio en el tema anterior:

```
<input type="button" value="pulsa" id="botoncito"
onclick="alert('hola');
document.getElementById('botoncito').style.color = 'red'">
```

O usar la variable `this`.

La variable `this` apunta directamente al elemento donde se está produciendo el evento. Así pues, el ejemplo anterior se puede poner un poco más limpio:

```
<input type="button" value="pulsa" id="botoncito"
 onclick="alert('hola'); this.style.color = 'red'">
```

12.2.1.2 Funciones externas

Esta formá es totalmente análoga a la anterior salvo que en lugar de colocar las instrucciones javascript directamente como contenido del atributo evento, se colocan en una función aparte y es el nombre de esta la que se usa como valor del atributo.

```
<head>
  <script>
      function cambiaMiBoton(){
        alert("hola");
        document.write("adios");
        document.getElementById('botoncito').style.color = 'red';
      }
  </script>
</head>
<body>

<input type="button" value="pulsa" id="botoncito"
      onclick="cambiaMiBoton()">

    . . .
```

De esta forma aislamos el código javascript y la estructura de HTML queda más limpia y clara que si usásemos la forma anterior.

También en este caso podemos usar la variable `this` si nos hace falta. Lo único que debemos hacer es pasar dicha variable como parámetro de la función.

```
<head>
  <script>
    function cambiaMiBoton(elemento){
      alert("hola");
```

```
      document.write("adios");
      elemento.style.color = 'red';
   }
  </script>
</head>
<body>

<input type="button" value="pulsa" id="botoncito"
      onclick="cambiaMiBoton(this)">

      ... ... ...
```

12.2.2 Manejadores semánticos

En el apartado anterior hemos visto las dos formas más comunes de controlar los eventos en Javascript. Sin embargo, dichas formas son, con diferencia las peores opciones a elegir.

Siempre hay que intentar separar la estructura de nuestra pagina (HTML) de la presentación (CSS) y de la programación (Javascript). Dado que las formas anteriores ensuciaban mucho el código de nuestras páginas haciendo uso de atributos en lso elementos, es preferible evitar su uso.

Los manejadores semánticos es una evolución de las funciones externas. Las cuales ahora asociamos a los elementos haciendo uso de las funciones de acceso a nodos del DOM vistas en el capítulo anterior.

Vamos a transformar esta forma vista anteriormente:

```
<head>
  <script>
      function cambiaMiBoton(){
        alert("hola");
```

```
         document.write("adios");
         document.getElementById('botoncito').style.color = 'red';
       }
  </script>
</head>
<body>

<input type="button" value="pulsa" id="botoncito"
       onclick="cambiaMiBoton()">

    . . .
```

En esta otra mucho más correcta:

```
. . .
<head>
  <script>

     //Funciones
     function cambiaMiBoton(){
        alert("hola");
        document.write("adios");
        document.getElementById('botoncito').style.color =
'red';
      }

     //manejadores
     document.getElementById('botoncito').onclick =
cambiaMiBoton;
  </script>
</head>
<body>
```

```
. . .

 <input type="button" value="pulsa" id="botoncito">
```

Importante destacar que cuando usamos este tipo de manejador para asociar una función, esta se pone sin los paréntesis (solo el nombre de la función). Si colocamos los paréntesis no estamos asociando esa función al evento (asociamos su salida) y eso no es lo que pretendemos.

Como con esta forma de manejar eventos no podemos hacer uso de los paréntesis de la definición de la función ¿que ocurre si la función tiene parámetros? Sencillo, que no podemos usarla. Es decir, si queremos utilizar esta forma de tratar eventos, debemos hacerlo haciendo uso de funciones sin parámetros

Si aún seguimos empeñados en usar una función con parámetros para manejar un evento (cosa totalmente lógica y normal) vamos a tener que hacer uso de las funciones anónimas.

12.2.2.1 Funciones anónimas

Como vimos en el capítulo correspondiente a funciones, las funciones anónimas son funciones sin nombre que sólo pueden ser llamadas por el propio Javascript en un momento dado (pej, cuando ocurra un evento) pero que no pueden ser llamadas por los programadores como si fueran funciones normales.

```
<script>
   //manejadores
   document.getElementById('botoncito').onclick = function(){
             //Codigo javascript
   };

   //Funciones normales
   . . .
</script>
```

En el ejemplo se ve que la forma de definirlas es colocando la palabra `function` seguido de paréntesis y abriendo las llaves correspondientes.

Dentro de esas llaves podemos poner el código Javascript que queramos. Variables, bucles o, porque no, llamadas a funciones normales con parámetros.

```
<head>
  <script>
     //Funciones normales
     function darSaludo(saludo){
        alert(saludo);
      }
     //manejadores
     document.getElementById('botoncito').onclick = function(){
               darSaludo("buenos dias");
     }
  </script>
</head>
<body>
  . . .
   <input type="button" value="pulsa" id="botoncito">
```

Incluso podemos usar la variable `this` dentro de las funciones anónimas sin ningún problema.

```
<head>
  <script>
     //Funciones
     function cambiaMiBoton(elemento){
        alert("hola");
        document.write("adios");
        elemento.style.color = 'red';
      }
     //manejadores
      document.getElementById('botoncito').onclick = function()
```

```
      {
               alert(this.value);
               cambiaMiBoton(this);
     }
  </script>
</head>
<body>
  . . .

   <input type="button" value="pulsa" id="botoncito">
```

Es muy importante destacar que al utilizar de manejadores semánticos (con o sin funciones anónimas) estamos usando funciones para acceder a nodos del DOM y, como vimos en el capitulo anterior, el árbol DOM esta listo cuando la página se ha cargado por completo.

Por tanto debemos esperar a que la pagina se cargue para poder ejecutar los manejadores. ¿Como se hace eso? Sencillo, el evento `onload` nos indica cuando se ha terminado de cargar la página, por tanto debemos definir nuestros manejadores dentro del manejador del evento `onload` de la página.

```
<head>
  <script>
     //Funciones
     function cambiaMiBoton(){
        alert("hola");
        document.write("adios");
       document.getElementById('botoncito').style.color = 'red';
      }

     function inicializar(){
      document.getElementById('botoncito').onclick = cambiaMiBoton;
     }
```

```
      //manejadores
      window.onload = inicializar;
  </script>
</head>
<body>
  . . .

   <input type="button" value="pulsa" id="botoncito">
```

O haciendo uso de una función anónima en ese evento:

```
<head>
  <script>

     //Funciones
     function cambiaMiBoton(){
        alert("hola");
        document.write("adios");
       document.getElementById('botoncito').style.color = 'red';
      }

      //manejadores
      window.onload = function(){
     document.getElementById('botoncito').onclick = cambiaMiBoton;
      }
  </script>
</head>
<body>
  . . .
   <input type="button" value="pulsa" id="botoncito">
```

12.3 Qué usar

Como se puede observar en los diferentes ejemplos, la manera más correcta de controlar eventos es haciendo uso de manejadores semánticos dado que no mezclan código Javascript con el HTML. Si además las funciones a asociar tienen parámetros, utilizaríamos los manejadores semánticos con funciones anónimas.

Existe una tercera forma de manejar eventos llamada *funciones asociadoras de eventos*. Esta última forma es la más elegante y correcta de todas. Además, permite el uso de determinados eventos que de otra forma no es posible usarlos.
Se trata básicamente de asociar funciones a eventos de objetos usado funciones propias de los navegadores.

Sin embargo, esta forma se sale un poco del nivel que pretendemos mantener en esta parte del manual por lo que esto es algo que se explicará en la segunda parte.