Eventos

Introducción

Nos permiten detectar acciones que realiza el usuario o cambios que suceden en la página y reaccionar en respuesta a ellas. Existen muchos eventos diferentes (podéis ver la lista en w3schools) aunque nosotros nos centraremos en los más comunes.

Javascript nos permite ejecutar código cuando se produce un evento (por ejemplo el evento *click* del ratón) asociando al mismo una función. Hay varias formas de hacerlo.

Cómo escuchar un evento

La primera manera "estándar" de asociar código a un evento era añadiendo un atributo con el nombre del evento a escuchar (con 'on' delante) en el elemento HTML. Por ejemplo, para ejecutar código al producirse el evento 'click' sobre un botón se escribía:

```
<input type="button" id="boton1" onclick="alert('Se ha pulsado');" />
```

Una mejora era llamar a una función que contenía el código:

```
<input type="button" id="boton1" onclick="clicked()" />
function clicked() {
  alert('Se ha pulsado');
}
```

Esto "ensuciaba" con código la página HTML por lo que se creó el modelo de registro de eventos tradicional que permitía asociar a un elemento HTML una propiedad con el nombre del evento a escuchar (con 'on' delante). En el caso anterior:

```
document.getElementById('boton1').onclick = function () {
  alert('Se ha pulsado');
}
...
```

NOTA: hay que tener cuidado porque si se ejecuta el código antes de que se haya creado el botón estaremos asociando la función al evento *click* de un elemento que aún no existe así que no hará nada. Para evitarlo siempre es conveniente poner el código que atiende a los eventos dentro de una función que se ejecute al producirse el evento *load* de la ventana. Este evento se produce cuando se han cargado todos los elementos HTML de la página y se ha creado el árbol DOM. Lo mismo habría que hacer con cualquier código que modifique el árbol DOM. El código correcto sería:

```
window.onload = function() {
  document.getElementById('boton1').onclick = function() {
    alert('Se ha pulsado');
  }
}
```

Event listeners

La forma **recomendada** de hacerlo es usando el modelo avanzado de registro de eventos del W3C. Se usa el método **`addEventListener`** que recibe como primer parámetro el nombre del evento a escuchar (sin 'on') y como segundo parámetro la función a ejecutar (OJO, sin paréntesis) cuando se produzca:

```
document.getElementById('boton1').addEventListener('click', pulsado);
...
function pulsado() {
  alert('Se ha pulsado');
})
```

Habitualmente se usan funciones anónimas ya que no necesitan ser llamadas desde fuera del listener:

```
document.getElementById('boton1').addEventListener('click', function() {
  alert('Se ha pulsado');
});
```

Si prefieres utilizar las funciones flecha de Javascript, quedaría incluso más legible:

```
const button = document.getElementById('boton1');
const pulsado = () => alert("Se ha pulsado'");
button.addEventListener('click', pulsado);
```

NOTA: igual que antes debemos estar seguros de que se ha creado el árbol DOM antes de poner un listener por lo que se recomienda ponerlos siempre dentro de la función asociada al evento **window.onload** o mejor **window.addEventListener('load', ...)** como en el ejemplo anterior.

Una ventaja de este método es que podemos poner varios listeners para el mismo evento y se ejecutarán todos ellos. Para eliminar un listener se usa el método **removeEventListener**.

```
document.getElementById('acepto').removeEventListener('click', aceptado);
```

NOTA: no se puede quitar un listener si hemos usado una función anónima, para quitarlo debemos usar como listener una función con nombre.JEM

EJEMPLO1:

```
<!DOCTYPE html>

<html>
<body>

<h2>JavaScript addEventListener()</h2>
<p>Este Ejemplo utiliza el metodo addEventListener() para anyadir un Evento a un Boton.</p>

<button id="miBoton">PULSA</button>
<p id="demo"></p>

<script>
document.getElementById("miBoton").addEventListener("click", displayDate);

function displayDate() {
  document.getElementById("demo").innerHTML = Date(); //Añade la Fecha
}
</script>
</body>
</html>
```

Tipos de eventos

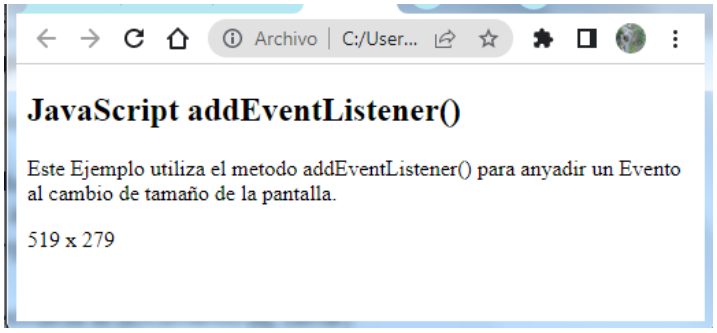
Según qué o dónde se produce un evento estos se clasifican en:

Eventos de página

Se producen en el documento HTML, normalmente en el BODY:

- **load**: se produce cuando termina de cargarse la página (cuando ya está construido el árbol DOM). Es útil para hacer acciones que requieran que el DOM esté cargado como modificar la página o poner listeners de eventos
- **unload**: al destruirse el documento (ej. cerrar)
- **beforeUnload**: antes de destruirse (podríamos mostrar un mensaje de confirmación)
- **resize**: si cambia el tamaño del documento (porque se redimensiona la ventana).

EJERCICIO1: Pon un evento a la página de ejemplo1 para que al redimensionar la ventana nos aparezca un texto indicando la altura y el ancho de esta.



Eventos de ratón

Los produce el usuario con el ratón:

- **click** / **dblclick**: cuando se hace click/doble click sobre un elemento
- **mousedown** / **mouseup**: al pulsar/soltar cualquier botón del ratón
- **mouseenter** / **mouseleave**: cuando el puntero del ratón entra/sale del elemento (tb. podemos usar mouseover/mouseout)
- **mousemove**: se produce continuamente mientras el puntero se mueva dentro del elemento

NOTA: si hacemos doble click sobre un elemento la secuencia de eventos que se produciría es: *mousedown -> mouseup -> click -> mousedown -> mouseup -> click -> dblclick*

EJERCICIO2: Pon un listener al botón de la página de ejemplo1 para que al hacer click se muestre el texto 'Botón Pulsado!. Ponle otro para que al pasar el ratón sobre él se muestre 'Ratón Sobre el Botón' y por último otro que cuando quitemos el ratón del botón muestre 'Ratón Sale'.

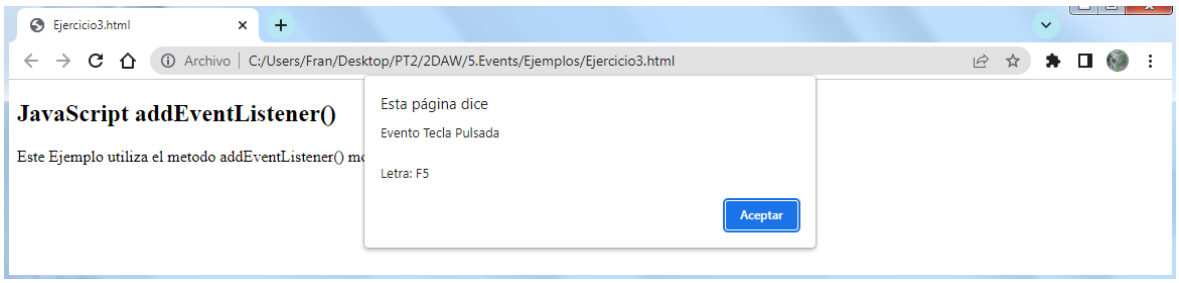
Eventos de teclado

Los produce el usuario al usar el teclado:

- **keydown**: se produce al presionar una tecla y se repite continuamente si la tecla se mantiene pulsada
- **keyup**: cuando se deja de presionar la tecla
- **keypress**: acción de pulsar y soltar (sólo se produce en las teclas alfanuméricas)

NOTA: el orden de secuencia de los eventos es: *keyDown -> keyPress -> keyUp*

EJERCICIO3: Pon un listener sobre el ejemplo1 que al pulsar cualquier tecla nos muestre un alert con 'Evento tecla pulsada' y la letra que se ha pulsado.
Ayuda : Ver método **event.key**



Eventos de toque

Se producen al usar una pantalla táctil:

- **touchstart**: se produce cuando se detecta un toque en la pantalla táctil
- **touchend**: cuando se deja de pulsar la pantalla táctil
- **touchmove**: cuando un dedo es desplazado a través de la pantalla
- **touchcancel**: cuando se interrumpe un evento táctil.

Eventos de formulario

Se producen en los formularios:

- **focus** / **blur**: al obtener/perder el foco el elemento
- **change**: al perder el foco un <input> o <textarea> si ha cambiado su contenido o al cambiar de valor un <select> o un <checkbox>
- **input**: al cambiar el valor de un <input> o <textarea> (se produce cada vez que escribimos una letra es estos elementos)
- **select**: al cambiar el valor de un <select> o al seleccionar texto de un <input> o <textarea>
- **submit** / **reset**: al enviar/recargar un formulario

Los objetos *this* y *event*

Al producirse un evento se generan automáticamente en su función manejadora dos objetos:

- **this**: siempre hace referencia al elemento que contiene el código en donde se encuentra la variable *this*. En el caso de una función listener será el elemento que tiene el listener que ha recibido el evento
- **event**: es un objeto y la función listener lo recibe como parámetro. Tiene propiedades y métodos que nos dan información sobre el evento, como:
  - **.type**: qué evento se ha producido (click, submit, keyDown, ...)
  - **.target**: el elemento donde se produjo el evento (puede ser *this* o un descendiente de *this*)
  - **.currentTarget**: el elemento que contiene el listener del evento lanzado (normalmente el mismo que *this*). Por ejemplo si tenemos un <*p*> al que le ponemos un listener de 'click' que dentro tiene un elemento _, *si hacemos _click* sobre el __ **event.target** será el __ que es donde hemos hecho click (está dentro de <*p*>) pero tanto __ como _event.currentTarget_ será _<p>_ (que es quien tiene el listener que se está ejecutando).
  - **.relatedTarget**: en un evento 'mouseover' **event.target** es el elemento donde ha entrado el puntero del ratón y **event.relatedTarget** el elemento del que ha salido. En un evento 'mouseout' sería al revés.
  - **cancelable**: si el evento puede cancelarse. En caso afirmativo se puede llamar a **event.preventDefault()** para cancelarlo

- **.preventDefault()**: si un evento tiene un listener asociado se ejecuta el código de dicho listener y después el navegador realiza la acción que correspondería por defecto al evento si no tuviera listener (por ejemplo un listener del evento *click* sobre un hiperenlace hará que se ejecute su código y después saltará a la página indicada en el *href* del hiperenlace). Este método cancela la acción por defecto del navegador para el evento. Por ejemplo si el evento era el *submit* de un formulario éste no se enviará o si era un *click* sobre un hiperenlace no se irá a la página indicada en él.
- **.stopPropagation**: un evento se produce sobre un elemento y todos sus padres. Por ejemplo si hacemos click en un que está en un <p> que está en un que está en el BODY el evento se va propagando por todos estos elementos y saltarían los listeners asociados a todos ellos (si los hubiera). Si alguno llama a este método el evento no se propagará a los demás elementos padre.
- dependiento del tipo de evento tendrá más propiedades:
  - eventos de ratón:
    - **.button**: qué botón del ratón se ha pulsado (0: izq, 1: rueda; 2: dcho).
    - **.screenX** / **.screenY**: las coordenadas del ratón respecto a la pantalla
    - **.clientX** / **.clientY**: las coordenadas del ratón respecto a la ventana cuando se produjo el evento
    - **.pageX** / **.pageY**: las coordenadas del ratón respecto al documento (si se ha hecho un scroll será el clientX/Y más el scroll)
    - **.offsetX** / **.offsetY**: las coordenadas del ratón respecto al elemento sobre el que se produce el evento
    - **.detail**: si se ha hecho click, doble click o triple click
  - eventos de teclado: son los más incompatibles entre diferentes navegadores. En el teclado hay teclas normales y especiales (Alt, Ctrl, Shift, Enter, Tab, flechas, Supr, ...). En la información del teclado hay que distinguir entre el código del carácter pulsado (e=101, E=69, €=8364) y el código de la tecla pulsada (para los 3 caracteres es el 69 ya que se pulsa la misma tecla). Las principales propiedades de *event* son:
    - **.key**: devuelve el nombre de la tecla pulsada
    - **.which**: devuelve el código de la tecla pulsada
    - **.keyCode** / **.charCode**: código de la tecla pulsada y del carácter pulsado (según navegadores)

- **.shiftKey** / **.ctrlKey** / **.altKey** / **.metaKey**: si está o no pulsada la tecla SHIFT / CTRL / ALT / META. Esta propiedad también la tienen los eventos de ratón

  NOTA: a la hora de saber qué tecla ha pulsado el usuario es conveniente tener en cuenta:

  - para saber qué carácter se ha pulsado lo mejor usar la propiedad *key* o *charCode* de *keyPress*, pero varía entre navegadores
  - para saber la tecla especial pulsada mejor usar el *key* o el *keyCode* de *keyUp*
  - captura sólo lo que sea necesario, se producen muchos eventos de teclado
  - para obtener el carácter a partir del código: `String fromCharCode(codigo);`

EJERCICIO 4: Pon un listener en la página de ejemplo para que al mover el ratón en cualquier punto de la ventana del navegador, utilizando el objeto **this** se muestre en algún sitio la posición del puntero respecto del navegador y respecto de la página.

EJERCICIO 5: Añade un **textarea de 10x25** y un **input tipo texto** en la página de ejemplo. Cuando pongamos el foco en el input el fondo cambiará a amarillo y cuando pierda el foco se quedará el fondo en blanco. Cuando empecemos a escribir en el textarea el fondo cambiará a gris. Utiliza el objeto this.

Bindeo* del objeto *this

En ocasiones no queremos que *this* sea el elemento sobre quien se produce el evento sino que queremos conservar el valor que tenía antes de entrar a la función listener. Por ejemplo la función listener es un método de una clase en *this* tenemos el objeto de la clase sobre el que estamos actuando pero al entrar en la función perdemos esa referencia.

El método *.bind()* nos permite pasarle a una función el valor que queremos darle a la variable *this* dentro de dicha función. Por defecto a una función listener de eventos se le *bindea* le valor de **event.currentTarget**. Si queremos que tenga otro valor se lo indicamos con **.bind()**:

```
document.getElementById('acepto').removeEventListener('click',
aceptado.bind(variable));
```

En este ejemplo el valor de *this* dentro de la función *aceptado* será *variable*. En el ejemplo que habíamos comentado de un listener dentro de una clase, para mantener el valor de *this* y que haga referencia al objeto sobre el que estamos actuando haríamos:

```
document.getElementById('acepto').removeEventListener('click',
aceptado.bind(this));
```

por lo que el valor de *this* dentro de la función *aceptado* será el mismo que tenía fuera, es decir, el objeto.

Propagación de eventos (bubbling)

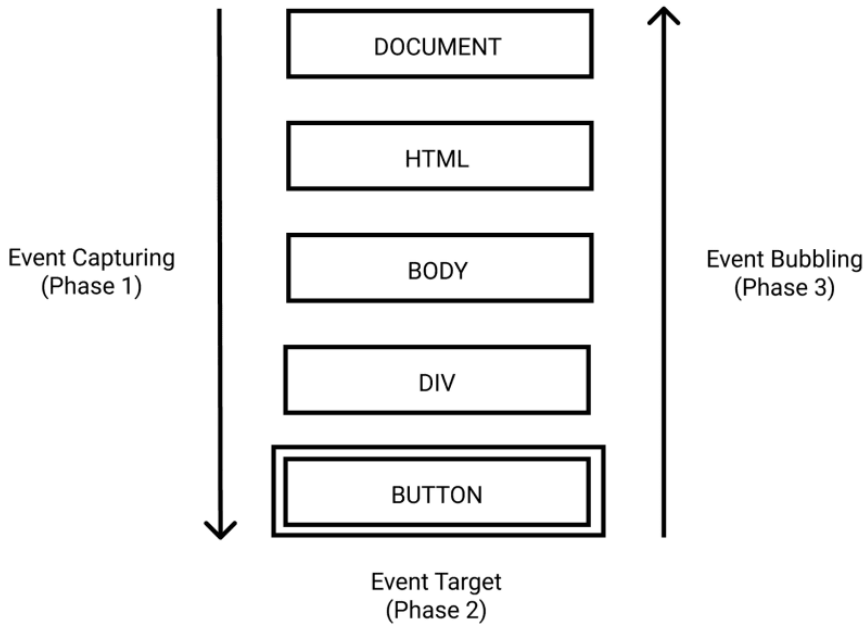
En JavaScript existe un concepto denominado **event bubbling** que se da sobre todo con nodos del DOM que están anidados unos dentro de otros.

El **event bubbling** o burbujeo de eventos es un método de propagación de eventos en la API del DOM.

Se da cuando activamos el evento de un elemento, y si su nodo padre tiene registrado otro evento, este último se activará automáticamente y así ira escalando en la jerarquía del DOM.

Fases del Event bubbling.

El Event bubbling pasa por 3 fases bien definidas:



- **Fase de Captura:** Se busca el elemento más profundo en el DOM que tenga registrado un evento en su listener.
- **Fase de Target:** Ejecuta el evento del elemento en sí.
- **Fase de Burbuja:** Verifica si los elementos padre de dicho elemento tienen eventos registrados en sus listeners, si es así, ejecuta dichos eventos de manera automática.

Estas 3 fases se ejecutan por defecto en JavaScript, es posible cambiar dicho comportamiento, pero esto lo veremos más adelante.

Event bubbling:

```
<div id="padre">Padre
  <div id="hijo">Hijo
    <button id="button">
      PULSA!
    </button>
  </div>
</div>

<script>
  const button = document.querySelector("#button");
  const hijo = document.querySelector("#hijo");
  const padre = document.querySelector("#padre");

  button.addEventListener("click", () => {
    alert("PULSADO");
  })
  hijo.addEventListener("click", () => {
    alert("PULSADO HIJO!");
  })
  padre.addEventListener("click", () => {
    alert("PULSADO PADRE!");
  })
</script>
</body>
```

Los eventos se van expandiendo desde el elemento más profundo al más externo, como si simulara una burbuja.

Una buena manera de desactivar este comportamiento es usando el método **stopPropagation()** del evento en si, de la siguiente manera:

```
evento.stopPropagation();
```

Modo captura

Por defecto, JavaScript se comporta con **Event bubbling**, pero podemos alterar el orden de los eventos cambiando al modo captura.

Esto se logra añadiendo un tercer parámetro a nuestro listener padre, como objeto, pasamos la propiedad **capture:true**.

Ejemplo: **objeto.addEventListener('click', funcion, true);**

Ejercicio 6: Utilizando el ejemplo anterior, cambia los listeners a modo captura y comprueba el resultado obtenido.

Ejercicio7: Con el mismo ejemplo, añade **stopPropagation()** al listener del botón y observa lo que ocurre.

innerHTML y listeners de eventos

Si cambiamos la propiedad *innerHTML* de un elemento del árbol DOM todos sus listeners de eventos desaparecen ya que es como si se volviera a crear ese elemento (y los listeners deben ponerse después de crearse).