UD8. GESTIÓN DE EVENTOS

1. INTRODUCCIÓN A LOS EVENTOS

Con esta unidad hemos llegado al colofón de la programación de aplicaciones web del lado cliente (Los elementos fundamentales del lenguaje JavaScript)

Los eventos son el mecanismo fundamental para comunicar la aplicación con el usuario. En esta interacción, la aplicación variará su funcionamiento en razón a las acciones realizadas por el usuario o usuaria. Para que se considere realmente un evento, la aplicación tiene que ser capaz de detectarlo y, lo más importante, ejecutar un código asociado a dicho evento.

Los eventos se asocian a un elemento concreto del DOM, de modo que, por ejemplo, podemos hacer una cosa cuando el usuario haga click sobre un párrafo y otra cuando el clic sea sobre otro párrafo.

La clave de los eventos en JavaScript es su capacidad asíncrona. Podemos estar esperando que el usuario haga clic en un botón, pero no se puede parar el resto del código por esa espera. La espera se produce en segundo plano mediante un proceso especial llamado listener. La gestión de los listeners la hace el navegador.

Por lo tanto, hay un proceso que automatiza la captura, de modo que cuando se produce un evento (por ejemplo, un clic sobre un elemento de la página), se ejecutará el código de una función callback.

2. CAPTURA DE EVENTOS

2.1. MÉTODOS CLÁSICOS Y OBSOLETOS

2.1.1. ATRIBUTOS ASOCIADOS A EVENTOS

Hace años, en las primeras versiones de JavaScript, la captura de un evento se realizaba desde el propio código HTML. Todavía se puede hacer porque se ha mantenido esta forma de captura por cuestiones de compatibilidad, pero no es nada aconsejable, ya que complica enormemente el mantenimiento del código al estar mezclado HTML y JavaScript de forma confusa.

Este primer método utiliza atributos en las etiquetas de los elementos de la página que comienzan con la palabra on seguida del nombre del evento. Por ejemplo: onclick. Como valor del atributo se indica el nombre de la función que contiene el código a ejecutar. Por ejemplo:

```
<p id="parrafo1" onclick="alert('Me has hecho click')">Soy clicable</p>
<p id="parrafo2">Yo no</p>
```

El primer párrafo usa el atributo onclick que permite lanzar código JavaScript cuando el usuario haga clic sobre el párrafo. El segundo párrafo no tiene evento asociado.

2.1.2. PROPIEDADES DE LOS ELEMENTOS ASOCIADOS A EVENTOS

Es una mejora del método anterior, que permite que el código JavaScript se independice de HTML, lo que mejora su mantenimiento. Se trata de usar propiedades que tienen el mismo nombre que los atributos HTML: la palabra on seguida del nombre del evento (onclick, onmouseover, etc). Ha sido el método más utilizado durante muchos años, pero actualmente no se recomienda su uso.

Un ejemplo de funcionamiento sería el siguiente:

```
<body>
<p id="parrafo1">Soy clicable</p>
<p id="parrafo2">Yo no</p>
</body>
<script>
        let parrafo1=document.getElementById("parrafo1");
        parrafo1.onclick=()=>{alert('Me has hecho click')};
</script>
```

Se ha asignado como función callbak una función flecha que muestra el mismo mensaje de tipo alert que en el ejemplo anterior.

2.2. MÉTODO ACONSEJABLE

El método que se aconseja actualmente es utilizar el método addEventListener que está disponible en todos los elementos. A este método hay que indicarle el nombre del evento y la función callback a ejecutar. Es el método que se debe utilizar según la norma actual. Todos los navegadores actules lo reconocen (las versiones anteriores a la 9, no lo hacen) y tiene una ventaja técnica sobre los 2 métodos anteriores: se puede asignar más de una función al mismo evento.

El código equivalente a los usados con los métodos anteriores es:

```
<body>
<p id="parrafo1">Soy clicable</p>
<p id="parrafo2">Yo no</p>
</body>
<script>
        let parrafo1=document.getElementById("parrafo1");
        parrafo1.addEventListener("click",()=>{alert('Me has hecho click')});
</script>
```

Para asignar varias acciones, basta con invocar otra vez a la función **addEventListener** con el mismo tipo de evento y usando otra función callback.

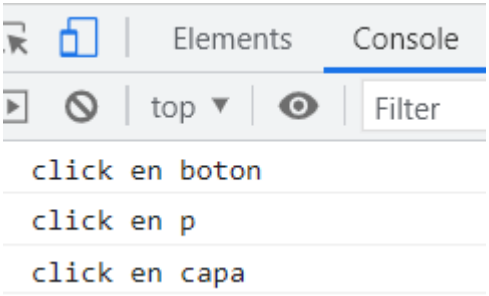
2.3. PROPAGACIÓN DE EVENTOS

Una cuestión fundamental en la captura de eventos es cómo se propagan los eventos sobre los contenedores de elementos. Es decir, supongamos que tenemos una capa de tipo **div**, en ella un párrafo y dentro del párrafo un botón. Hacemos clic sobre el botón. ¿El párrafo podría capturar el click? ¿Y la capa?

Vamos lo que ocurre con esta página completa:

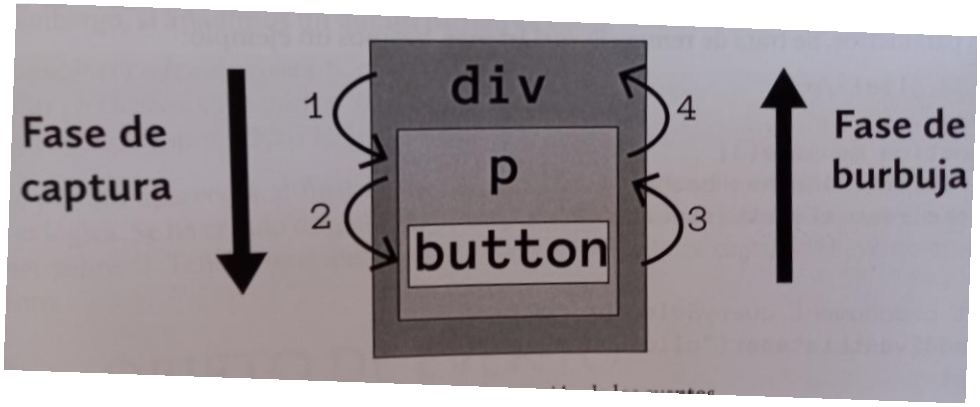
```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<title>Propagación de eventos</title>
<style>
    div{
        position:  fixed;
        left: 0;
        top: 0;
        height: 50%;
        width: 100%;
        background-color: gray;
    }
   p{
        height: 50%;
        margin: 0;
        background-color: lightgray;
    }
</style>
</head>
<body>
   <div>
      <p>
         <button>¡Hazme clic!</button>
      </p>
   </div>
</body>
<script>
        let capa=document.querySelector("div");
        let p=document.querySelector("p");
        let boton=document.querySelector("button");
        capa.addEventListener("click",()=>{console.log("click en capa")});
        p.addEventListener("click",()=>{console.log("click en p")});
        boton.addEventListener("click",()=>{console.log("click en boton")});
</script>
</html>
```

La situación comentada es la que presenta el código anterior. Hemos capturado los eventos de tipo click en loes 3 elementos (div, p y button). Si ejecutamos el código, hacemos clic en el botón, por consola vemos este texto:



El evento se propaga desde el botón (elemento más interior) hasta el elemento de tipo div (elemento más cercano a la raíz del DOM).

El proceso de captura de eventos se describe en esta imagen:



Por defecto, la función asociada al evento se lanza en la fase de burbuja, por eso en el código anterior observamos primero el mensaje del botón, luego el del párrafo y luego el de la capa.

Podemos modificar este comportamiento, si indicamos que el lanzamiento sea en la fase de captura. Se consigue este efecto (no disponible en versiones de IE anteriores a la 9) gracias a la que, en realidad, el método **addEventListener** tiene un tercer parámetro relacionado con el tipo de captura. Se trata de un valor booleano que tiene estas posibilidades:

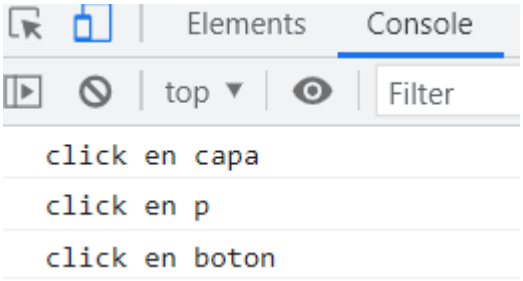
- **true**. El código de captura se lanza en la fase de captura.
- **false**. Es el valor por defecto. El código se lanza en la fase de burbuja.

Por lo tanto, si cambiamos las 3 últimas líneas de código JavaScript anterior:

```
<script>
        let capa=document.querySelector("div");
        let p=document.querySelector("p");
        let boton=document.querySelector("button");
        capa.addEventListener("click",()=>{console.log("click en capa")},true);
        p.addEventListener("click",()=>{console.log("click en p")},true);
```

```
        boton.addEventListener("click",()=>{console.log("click en boton")},true);
</script>
```

Si ejecutamos el código por consola tendremos el siguiente resultado:



2.4. ANULAR EVENTOS

Imaginemos que deseamos que cuando hagamos un clic a un elemento concreto se nos muestre un mensaje, pero queremos que ese mensaje aparezca una sola vez. Por defecto, el evento se captura indefinidamente, pero existe un método contrario a **addEventListener** que tiene los mismos parámetros. Se trata de **removeEventListener**.Veamos un ejemplo:

```
<p>¡Hazme clic!</p>
<script>
    function mensaje(){
        alert("¡Me has hecho un clic!");
        p.removeEventListener("click",mensaje);
    }
    let p=document.querySelector("p");
    p.addEventListener("click",mensaje);
</script>
```

Dentro de la propia función **mensaje**, tras escribir el mensaje en sí, se invoca al método **removeEventListerner** el cual provoca que no se vuelva a invocar a esta función tras otro clic.

Hay que tener en cuenta que solo podemos retirar funciones de captura que tengan nombre, no podemos anular funciones anónimas asociadas a eventos.

2.5. CAPTURA DE EVENTOS EN ELEMENTOS DINÁMICOS

A la hora de implementar capturas de eventos, hay que tener en cuenta que los **listeners** se asocian a elementos del DOM que existan en ese momento. Eso puede dar lugar a problemas si no se tiene cuidado.

Supongamos que queremos crear un documento en el que consigamos que al hacer clic en todos los párrafos de tipo p, se muestre un cuadro de alerta, el texto del párrafo. El código podría ser este:

```
<p>Uno</p>
<p>Dos</p>
<p>Tres</p>
<p>Cuatro</p>
...
<script>
let parrafos=document.querySelectorAll("p");
```

```
        for(let parrafo of parrafos){
        parrafo.addEventListener("click",()=>{alert(parrafo.textContent)});
}
```

La página posee 4 párrafos, con los textos **Uno**, **Dos**, **Tres** y **Cuatro**. El código JavaScript asocia todos los párrafos a la variable que se llama **parrafos**. Pero, para implementar el control de eventos, necesitamos asignar un evento a cada párrafo. Por eso hay un bucle que recorre cada elemento devuelto por el método **querySelectorAll** y así vamos asignando a cada párrafo la captura del evento. Si ejecutamos este código, veremos que, efectivamente, al hacer clic en cualquiera de los 4 párrafos, se nos muestra el mensaje.

Sin embargo, si añadimos un quinto párrafo de tipo p mediante JavaScript.

```
let nuevoParrafo=document.createElement("p");
nuevoParrafo.textContent="Cinco";
document.body.appendChild(nuevoParrafo);
```

Este párrafo aparecerá al final de los otros 4. Al hacer clic en él, no ocurrirá nada. El motivo es porque se ha creado después del bucle y por ello no se ha ejecutado el método **addEventListener** sobre él. Tendríamos que añadir el código que permita capturar el evento en el nuevo elemento.

3. OBJETO DE EVENTO

3.1. UTILIDAD Y USO DEL OBJETO DE EVENTO

Cuando se produce un evento, el navegador crea automáticamente un objeto cuyas propiedades pueden ser muy útiles a los desarrolladores. La información fundamental que graban son las coordenadas del cursor del ratón, si hay teclas pulsadas, el elemento que ha producido el evento, etc.

La función que se invoca de forma automática al producirse el evento puede tener un parámetro que será una referencia al objeto de evento. Gracias a ese parámetro podremos leer la información del evento.

Una de las propiedades que posee este objeto se llama target y es una referencia al elemento que causó el evento.

Veamos un ejemplo usando esta propiedad para entender cómo funciona el objeto de evento:

```
<p>Uno</p>
<p>Dos</p>
<p>Tres</p>
...
<script>
   function escribeContenido(evento){
        console.log(evento.target.textContent);
   }
   let parrafos=document.querySelectorAll("p");
   for(let parrafo of parrafos){
        parrafo.addEventListener("click",escribeContenido);
   }
</script>
```

En este código se asigna a los elementos de los 3 párrafos, el código de la función **escribeContenido** que será ejecutado cuando se produzca un clic en cada párrafo. La función recibe un parámetro en el que se almacenará el objeto con los datos importante del evento. La función en este caso usa la propiedad **target** para llegar al elemento que causó el clic y así acceder al texto de ese elemento que, finalmente, se mostrará en la consola.

3.2. OBTENER COORDENADAS DEL EVENTO

Los objetos de evento poseen 2 propiedades para obtener las coordenadas del ratón en el momento del evento: **clientX** y **clientY**. La primera obtiene la posición horizontal en píxeles y la segunda la vertical. Ambas o hacen utilizando como referencia la esquina superior izquierda de la ventana del navegador, que será el punto (0,0) de coordenadas.

Hay 2 propiedades similares: **screenX** y **screenY**. La diferencia es que el origen de coordenadas no es la esquina del navegador, sino de la pantalla. Coinciden muchas veces, pero si la ventana no está maximizada, no coincidirán.

- ❑ **ACTIVIDAD 1**—CARTEL PERSIGUIENDO AL RATÓN

Otras coordenadas que se almacenan en el objeto de evento son **pageX** y **pageY**. Funcionan como **clientX** y **clientY**, pero no solo toman las de la ventana, tienen en cuenta el desplazamiento realizado en el elemento. Es decir, si hemos avanzado 2 pantallas de 500 píxeles de alto usando las barras de desplazamiento y el cursor está a mitad de pantalla, **clientY** nos diría 250, mientras que **pageY** nos diría 1250.

Puedes comprobarlo reutilizando el código de la Actividad 1. Añade lo siguiente justo antes de cerrar la etiqueta body:

```
<script>
document.addEventListener("mousemove",function(ev){
        console.log(`screenX:${ev.screenX}), screenY:${ev.screenY}\n`+
            `clientX:${ev:clientX}, clientY:${ev.clientY}\n`+
            `pageX:${ev.pageX}, pageY:${ev.pageY}`};
})
</script>
```

Haz que la ventana donde se ejecuta la página no esté maximizada. Desplázate hacia abajo en el documento varias pantallas y ahora, mostrando la consola, compara las coordenadas. Verás como ***pageY*** es la más grande porque cuenta todo el desplazamiento, como ***screenY*** es la segunda porque toma la pantalla y ***clientY*** es la más pequeña.

3.3. OBTENER LA TECLA PULSADA

En eventos de teclado, el objeto de evento posee información sobre la tecla pulsada. Son importantes estos datos en aplicaciones como la programación de juegos o en el control de la entrada por teclado.

La propiedad más importante es **key** que obtiene que indica la tecla pulsada. Es el más cómodo de programar porque es fácil de entender. Así sus valores fundamentales devueltos son:

- Si es una tecla de carácter, nos retorna el carácter. Por ejemplo: "**a**","**g**","**k**","**ñ**",etc. También nos indica si hay mayúsculas porque la letra retornaría mayúscula: "**A**","**G**","**K**","**Ñ**",etc.
- Con las teclas numéricas actúa igual, nos retorna el número. Pero con shift pulsada a la vez, nos devuelve el segundo símbolo de la letra (**&,$,%**,etc). Con Alt Gr nos retorna el tercero (**|,@,#**,etc).
- Actúa igual con cualquier tecla que escriba caracteres, simplemente indica qué carácter normalmente sale escrito.
- En el caso de pulsar, sin más, teclas de control, nos devolverá su nombre, "**Control**", "**Alt**", "**Shift**", "**AltGraph**", "**F5**", "**F6**", "**KeyUp**", "**KeyDown**", "**Home**","**End**", etc.

A veces es necesario saber si se pulsó a la vez que la tecla, las teclas de control especial: **Ctrl**, **Alt** o **Shift**. Tenemos 3 propiedades relacionadas con estas teclas que devolverán true si la tecla en cuestión se pulsó. Son: **altKey**, **ctrlKey**, **shiftKey** y **metaKey** (en los ordenadores Mac).

- ❑ **ACTIVIDAD 2** -- PRÁCTICA DE CAPTURA DE TECLAS

Otra propiedad interesante es **location** que permite saber la localización de la tecla en el teclado. El caso habitual es tener que distinguir una tecla Shift de la otra. La propiedad **location** lo consigue, esta propiedad puede devolver los siguientes valores:

Devuelve el número entero que indica la posición del teclado en la que se encuentra la tecla. Valores posibles que devuelve:

VALOR	CONSTANTE RELACIONADA	SIGNIFICADO
0	DOM_KEY_LOCATION_STANDARD	Teclado normal
1	DOM_KEY_LOCATION_LEFT	Zona izquierda (para distinguir las 2 teclas de control por ejemplo)
2	DOM_KEY_LOCATION_RIGHT	Zona derecha(para distinguir las 2 teclas de control por ejemplo)
3	DOM_KEY_LOCATION_NUMPAD	Teclado numérico

3.3.1. OBTENER LOS BOTONES DEL RATÓN

En los eventos del ratón hay propiedades comunes con las teclas. Por ejemplo, también tenemos las propiedades **altKey**, **ctrlKey**, **shiftKey** y **metaKey**, para saber si la pulsación de alguna de estas teclas acompaña al evento de ratón.

Hay otras propiedades de coordenadas muy interesantes. Así **movementX** y **movementY** nos retornan la diferencia en píxeles de las coordenadas X e Y respecto al último movimiento del ratón.

Además, la propiedad button devuelve el botón de ratón pulsado en el momento del evento. Los valores posibles para la propiedad button son:

VALOR	SIGNIFICADO
0	Botón principal del ratón
1	Botón central del ratón (en muchos casos, el botón de la rueda)
2	Botón secundario del ratón (en el caso de personas diestras, es el botón derecho)
3	Cuarto botón. En muchos ratones es el de retroceder página
4	Quinto botón. En muchos ratones es el de avanzar página

3.4. ANULAR COMPORTAMIENTOS

Además de propiedades, los objetos de evento poseen métodos. Uno de los más importantes es **preventDefault**. No tiene parámetros y lo que hace es conseguir que, si ese evento producía en el elemento un comportamiento concreto por defecto, que ese comportamiento no se produzca.

Es más fácil de entender la idea con un ejemplo. Supongamos que hemos puesto un enlace y deseamos que el usuario confirme que desea ir al destino de ese enlace, de modo que, si no lo confirma, no nos moveremos de la página actual. El código podría ser este:

```
<a href="https://es.wikipedia.org"> Ir al a wikipedia</a>
</body>
<script>
        let enlace=document.querySelector("a");
        enlace.addEventListener("click",function(ev){
            if(confirm("¿Realmente desea abandonar esta página?"))
            location="https://es.wikipedia.org/";
        });
</script>
```

Si ejecutamos este código en una página web, efectivamente se nos requiere confirmar la acción, pero, aunque no aceptemos el cuadro, la acción se lleva a cabo, porque los enlaces, por defecto envían al usuario al destino. Por ello el código podría ser este:

```
<a href="https://es.wikipedia.org"> Ir al a wikipedia</a>
</body>
<script>
        let enlace=document.querySelector("a");
        enlace.addEventListener("click",function(ev){
            if(!confirm("¿Realmente desea abandonar esta página?"))
            ev.preventDefault();
        });
</script>
```

3.5. CANCELAR PROPAGACIÓN

Anteriormente hemos explicado cómo funciona la propagación de eventos. Podemos cancelar la propagación mediante un método llamado **stopPropagation**. Vemos este ejemplo:

```
<body>
<div>
Pinta de rojo
<button>Pinta de verde</button>
</div>
</body>
<script>
        let boton=document.querySelector("button");
        let div=document.querySelector("div");
        boton.addEventListener("click",(ev)=>{
            document.body.style.backgroundColor="green";
        });
        div.addEventListener("click",(ev)=>{
            document.body.style.backgroundColor="red";
        });
</script>
```

El botón está dentro de un elemento de tipo div. Hacer clic en el elemento div (texto Pinta de rojo) provoca colorear todo el cuerpo de la página de rojo. Hacer clic en el botón lo pinta de verde. Sin embargo, al hacer clic en el botón, la página también pinta de rojo. Si pudiéramos ver a cámara lenta el proceso, veríamos primero el color verde y luego el rojo. Predomina el color rojo, porque el evento clic se propaga el elemento div, después de haber sido capturado por el botón.

Para evitar este efecto podemos el método **stopPropagation**:

```
<body>
<div>
Pinta de rojo
<button>Pinta de verde</button>
</div>
</body>
<script>
        let boton=document.querySelector("button");
        let div=document.querySelector("div");
        boton.addEventListener("click",(ev)=>{
            document.body.style.backgroundColor="green";
            ev.stopPropagation();
        });
        div.addEventListener("click",(ev)=>{
            document.body.style.backgroundColor="red";
        });
</script>
```

Hay que observar que hemos indicado el fin de la propagación en el elemento div. De esta forma, este elemento se retira del lanzamiento del evento en la fase de burbuja y así solo se ejecuta el código en el clic del botón.

Por supuesto el funcionamiento también podíamos haberlo modificado si la ejecución de código la hubiéramos obligado a realizar en la fase de captura.

3.6. LANZAR EVENTOS

JavaScript permite lanzar eventos a voluntad. La idea se basa en crear objetos de evento propios y mediante el método **dispatchEvent** de los elementos, enviar el evento creado.

La creación del evento se basa en el objeto **Event** (o en sus descendientes más específicos como **MouseEvent** por ejemplo) el cual funciona indicando el tipo de evento en la creación. Por ejemplo:

```
Let evento=new Event(“click”);
```

Después lanzaríamos el evento a cualquier elemento que esté capturando ese evento. Un ejemplo de uso (poco práctico, pero fácil de entender) será el que proporciona el siguiente código:

```
<p>
<button id="boton1">Coloreo de rojo</button>
<button id="boton2">Yo hago lo mismo</button>
</p>
</body>
<script>
let boton1=document.getElementById("boton1");
let boton2=document.getElementById("boton2");
boton1.addEventListener("click",()=>{
    document.body.style.backgroundColor="red";
});
boton2.addEventListener("click",()=>{
    let e1=new Event("click");
    boton1.dispatchEvent(e1);
});
</script>
```

Aparecerán 2 botones, ambos realizan la misma acción, porque el evento **onclick** del segundo botón realiza un clic sobre el primero.

4. LISTA DE TIPOS DE EVENTOS

En todos los ejemplos anteriores hemos utilizado el evento **click**. Sin embargo, hay muchos otros eventos que podemos capturar. Se detallan los principales a continuación.

4.1. EVENTOS DE RATÓN

Los eventos de ratón los produce este dispositivo. Pero también los dispositivos táctiles pueden producir algunos de estos eventos. Por ejemplo, el evento **click** ocurre cuando el usuario pulsa y despulsa el botón

principal del ratón sobre el objeto que posee el **listener**, pero también ocurre si el usuario, en un dispositivo táctil, golpea y suelta el dedo sobre dicho objeto.

Por otro lado, aunque los eventos se refieren al ratón, realmente valen para cualquier dispositivo apuntador capaz de mover el cursor por pantalla.

La tabla completa de los eventos de ratón es la siguiente:

EVENTO	EXPLICACIÓN
click	El usuario hace click sobre el botón principal del dispositivo apuntador. Esta acción exactamente consiste en bajar y subir rápidamente el botón principal del dispositivo apuntador (o dar un golpe con el dedo en un dispositivo táctil)
dblclick	El usuario golpea 2 veces rápidas sobre el botón principal del dispositivo apuntador (o con el dedo en un dispositivo táctil)
mousedown	El usuario pulsa un botón del dispositivo apuntador. Se produce este evento justo cuando el usuario pulsa el botón, antes de que le suelte.
mouseup	Se produce cuando el usuario suelta el botón de dispositivo apuntador que había pulsado previamente.
mouseenter	Se produce cuando el usuario mueve el cursor del dispositivo apuntador dentro del elemento que captura el evento. Solo se produce si el cursor estaba fuera del elemento y el usuario lo acaba de mover dentro.
mouseleave	Se produce cuando el usuario mueve el cursor del dispositivo apuntador fuera del elemento que captura el evento. Solo se produce si el cursor estaba dentro del elemento y el usuario lo acaba de mover fuera.
mousemove	Se produce cada vez que el usuario mueve el cursor dentro del elemento (estando previamente dentro)
mouseover	Se produce cuando el cursor apuntador entra dentro del elemento o de cualquiera de los hijos del elemento.
mouseout	Se produce cuando el cursor apuntador abandona el elemento o cualquiera de los hijos del elemento.
contextmeu	Se produce cuando el usuario acaba de pedir el menú de contexto. Normalmente ese menú aparece al pulsar el botón secundario del dispositivo apuntador. En dispositivos táctiles suele ocurrir cuando el usuario deja el dedo apretado sobre el elemento 2 o 3 segundos al menos.

- ❑ **ACTIVIDAD 3 –**CAPTURA DE EVENTOS DE RATÓN

4.2. EVENTOS DE TECLADO

EVENTO	EXPLICACIÓN

keypress	Se produce cuando un usuario pulsa y suelta una tecla
keydown	Se produce justo cuando el usuario ha pulsado la tecla (antes de que la suelte)
keyup	Se produce cuando el usuario suelta la tecla.

- ❑ **ACTIVIDAD 4 –** CAPTURA DE EVENTOS DE TECLADO

4.3. EVENTOS DE MOVIMIENTO EN LA VENTANA

EVENTO	EXPLICACIÓN
scroll	Ocurre cuando se ha desplazado la ventana a través de las barras de desplazamiento o usando el dispositivo táctil
resize	Evento de window que se produce cuando se cambia el tamaño de la ventana

- ❑ **ACTIVIDAD 5 –**CAPTURA DEL SCROLL

4.4. EVENTOS SOBRE CARGA Y DESCARGA DE ELEMENTOS

EVENTO	EXPLICACIÓN
load	Se concluyó la carga del elemento. Es uno de los elementos más importantes a capturar para asegurar que el código siguiente funciona con la seguridad de que está cargado lo que necesitamos. Cuando se aplica al elemento **window**, se produce cuando todos los elementos del documento se han cargado.
DOMContentLoaded	Similar al evento load. Se produce cuando el documento HTML ha sigo cargado. A diferencia de load, se dispara sin esperar a que se terminen de cargar las hojas de estilos, imágenes y elementos en segundo plano. En general, para JavaScript, es más conveniente este evento.
abort	Se produce cuando se anula la carga de un elemento
error	Sucede si hubo un error en la carga
progress	Se produce si la carga está en proceso
readystatechange	Ocurre cuando se ha modificado el estado del atributo **readystate**, lo cual ocurre cuando se ha modificado el estado de carga y descarga.

- ❑ **ACTIVIDAD 6—**PANEL DE CARGA

Concretamente, el evento load del objeto window es muy importante. Hay que tener en cuenta que desde JavaScript manejamos elementos del DOM continuamente. Además, JavaScript es un lenguaje asíncrono y eso puede implicar que intente manipular un componente que aun no se ha cargado.

Por ello es muy habitual que todo el código JavaScript para manipular elementos del DOM, se coloque dentro de la función **callback** asociada al evento **load** del objeto **window**.

4.5. OTROS EVENTOS

Debido a la importancia de los formularios en la creación de aplicaciones, dejamos este apartado al final de esta unidad.

Otros eventos más específicos, aunque no son de uso habitual, pero nos pueden servir de utilidad en algunas aplicaciones se detallan a continuación.

4.5.1. EVENTOS SOBRE EL HISTORIAL

EVENTO	EXPLICACIÓN
popstate	Se produce si se cambia el historial

4.5.2. EVENTOS RELACIONADOS CON LA REPRODUCCIÓN DE MEDIOS

EVENTO	EXPLICACIÓN
waiting	Sucede cuando el vídeo se ha detenido por falta de datos
playing	El medio está listo para su reproducción después de que se detuviera por falta de datos u otras causas
canplay	Ocurre cuando se detecta que un vídeo (u otro elemento multimedia) ya se puede reproducir (aunque no se haya cargado del todo)
canplaythough	Se produce si se estima que el vídeo ha cargado suficientes datos para poderse reproducir sin tener que esperar la llegad de más datos
pause	El medio de reproducción se ha pausado
play	Ocurre cuando el medio de reproducción se ha empezado a reproducir tras una pausa
ended	Se produce si la reproducción del vídeo o audio se ha detenido, sea por haber llegado al final o porque no hay más datos disponibles.
loadeddata	Se produce si se ha cargado el frame actual (normalmente el primero)
suspend	Se lanza si se ha suspendido la carga del medio
emptied	Sucede si se ha vaciado el medio por un nuevo intento de carga por parte del usuario o por otras razones
stalled	Sucede ante un fallo en la carga, pero sin que se detenga la misma
seeking	Ocurre cuando se ha iniciado una labor de búsqueda en el medio

seeked	Ocurre cuando se ha finalizado la labor de búsqueda
loadedmetadata	Se han cargado los metadatos del medio
durationchange	Sucede si se modifica el atributo duration del medio
timeupdate	Ocurre si se ha modificado el atributo currentTime del medio
ratechange	Se produce cuando el ratio del vídeo se ha modificado
volumechange	SE lanza si el volumen se ha modificado

4.5.3. EVENTOS DE ARRASTRE

Están relacionados con la interfaz de arrastre que es parte de HTML5. Para que un elemento pueda ser arrastrable debe tener el atributo **draggable** colocado con valor true:

div id=”capaArrastrable” **draggable**=”true”>¡Soy arrastrable!

En este tipo de operaciones hay 2 capas protagonista: una capa arrastrable (la que realmente se arrastra) y un posible destino del arrastre.

La capa que se arrastra puede generar estos eventos:

EVENTO	EXPLICACIÓN
dragstart	Se produce cuando el usuario empieza a arrastrar el elemento
drag	Ocurre, una vez iniciado el arrastre, cada vez que se sigue arrastrando el elemento (cada vez que lo movemos)
dragstop	Se produce cuando el arrastre finaliza

Eventos que se producen en el elemento destino del arrastre

EVENTO	EXPLICACIÓN
dragenter	Se produce cuando el elemento que se está arrastrando, entra en el elemento destino
dragover	Ocurre cada vez que se continúa, tras haber entrado, arrastrando el elemento origen sobre el destino.
dragleave	Ocurre cuando el elemento que se arrastra, sale del destino
drop	Ocurre cuando el elemento origen se suelta dentro del destino. Para que este evento se pueda capturar hay que eliminar el comportamiento por defecto del evento **dragover**.

- ❑ **ACTIVIDAD 7 –** CAPTURA DE ARRASTRE

4.5.4. EVENTOS SOBRE ANIMACIONES Y TRANSICIONES

Son eventos que podemos capturar cuando hemos programado animaciones sobre uno o más elementos desde CSS o por transiciones. Tenemos estos eventos posibles a capturar:

EVENTO	EXPLICACIÓN
animationstart	Se produce cuando se inicia una animación sobre un elemento
animationinteraction	Se produce justo cuando se repite la animación
animationend	Se produce cuando finaliza la animación
transitionrun	Se lanza cuando ya se ha preparado para empezar la transición
transitionstart	Ocurre cuando se inicia una transición
transitionend	Ocurre al finalizar una transición

4.5.5. EVENTOS DEL PORTAPAPELES

Permiten capturar los eventos de cortar, copiar y pegar. Todos ellos se relacionan con el objeto clipboard que es el que permite gestionar el portapapeles del sistema.

EVENTO	EXPLICACIÓN
cut	Se produce cuando el usuario intenta cortar contenido del elemento
copy	Se produce cuando el usuario intenta copiar contenido del elemento
paste	Se produce cuando el usuario intenta pegar contenido

4.5.6. EVENTOS ESPECIALES

EVENTO	EXPLICACIÓN
offline	Solo funciona para el objeto window y se produce si el navegador se desconecta de la red
online	Solo funciona para window y se produce si el navegador vuelve a conectarse después de haber estado desconectado
fullscreenchange	OCurre cuando un elemento pasa a modo de pantalla completa
fullscreenerror	Sucede si hay un error al pasar un elemento a modo de pantalla completa
message	Evento que se asocia a numerosos elementos de envío de mensajes como los que se producen a través de las **APIs WebSockets**, **WebWorkers** y otras.

5. FORMULARIOS

5.1. EL FORMULARIO COMO OBJETO DEL DOM

Los formularios son el componente fundamental de captura de eventos en una aplicación web. Aunque nos han sido útiles los métodos **alert**, **prompt** y **confirm**, la realidad es que las aplicaciones profesionales no los utilizan nunca. Si hay que mostrar mensajes se hacen en elementos HTML normales (paneles, capas, etc) y la introducción de datos por parte del usuario se hace siempre con formularios.

Los formularios son la base de la comunicación web con el usuario. El objeto **document** dispone de una propiedad llamada **forms** que retorna una colección con todos los formularios del documento. Así **document.forms[0]** hará referencia al primer formulario del documento. Pero es posible, y más recomendable de hecho, acceder al formulario mediante otros métodos, por ejemplo, su identificador.

Acceder a los elementos del formulario se hacía, de forma clásica, mediante el atributo **name** de los controles del formulario. Este atributo es el que permite dar nombre a la información que se graba con el control y que se enviará cuando se pulse el botón de envío (**submit**) del formulario. Así, si tenemos un control para grabar el primer apellido del usuario y ese control utiliza el atributo **name** para darle el nombre de apellido1, la forma de acceder a ese control es (suponemos que estamos en el primer formulario del documento):

```
document.forms[0].apellido1;
```

Igualmente podríamos:

```
document.forms[0][“apellido1”];
```

Y, también:

```
Console.log(document.querySelector(“[name=’apellido1’]”);
```

Si hay varios elementos con el mismo ombre (como ocurre con los radiobuttons), esta última forma de acceder debe utilizar querySelectorAll, la cual devolverá una colección con todos los botones de radio con ese nombre. Después, tendremos que ir recorriendo cada botón de la colección.

NOTA: Damos por hecho que se conocen los entresijos de los formularios HTML tales como: tipos de controles, envío de datos por **GET** y **POST**, atributos de los controles (especialmente **value**, **name** e **id**), botones **submit** y **reset**, etc. De no ser así, se aconseja un repaso de estos conceptos para aprovechar mejor este apartado.

El objeto de formulario (form) posee estas propiedades y métodos:

PROPIEDAD O MÉTODO	USO
elements	Devuelve una colección que contiene todos los controles del formulario
length	Obtiene el número de controles del formulario
action	Devuelve el contenido de la propiedad action, que marca la URL destino de los datos del formulario. Permite su modificación

method	Retorna el contenido de la propiedad method del formulario, que marca la forma de envío de los datos (get o post). Permite su modificación
enctype	Obtiene o cambia la forma de codificar los datos del formulario
acceptCharset	Obtiene o modifica el conjunto de caracteres del formulario
submit()	Envía los datos del formulario a su destino
reset()	Deja los valores de los controles del formulario a su estado por defecto

Esos métodos van a permitir automatizar acciones de forma muy efectiva en los formularios

5.2. EVENTOS DEL FORMULARIO

5.2.1. EVENTO SUBMIT

Se trata de un evento que produce el envío de los datos del formulario a su destino. Es muy habitual validar los datos previamente a su envío para que el usuario detecte los fallos antes de realizar el envío. Este evento se produce justo antes del envío y se asocia al propio formulario (también se puede capturar en el botón de tipo **submit** del formulario). Tras su captura podemos validar los datos, pero teniendo la precaución de que si no cancelamos la acción por defecto de este evento (mediante el **preventDefault** del objeto de evento), la acción se llevará a cabo.

Como ejemplo, supongamos que tenemos un formulario en el que pedimos al usuario su nombre de usuario y que dicho nombre solo pueden ser letras del alfabeto inglés y con un mínimo de 6 caracteres. No enviaremos datos si no se cumple esa premisa y cuando detectemos el fallo, avisaremos del error y no permitiremos el envío. El código podría ser este:

```
<form action="https://jorgesanchez.net/servicios/ver-datos.php">
<input type="text" name="usuario" id="usuario"><br>
<button type="submit">Enviar</button>
</form>
<div id="mensaje"></div>
</body>
<script>
        let formulario=document.forms[0];
        let tUsuario=document.getElementById("usuario");
        let cMensaje=document.getElementById("mensaje");
        formulario.addEventListener("submit", function(ev){
            let exp=/^[a-zA-Z]{6,}$/;
            if(exp.text(tUsuario.value)==false){
                ev.preventDefault();
                mensaje.innerHTML="<p>Error:nombre de usuario no válido</p>"
            };
        });
```

La captura del evento permite detener el envío de datos, si no se cumple la expresión regular en el valor del cuadro de texto en el que se escribe el nombre de usuario.

5.2.2. EVENTO RESET

Este evento se produce cuando se ha ordenado, a través de un botón de tipo **reset** por ejemplo, que el formulario muestre los valores iniciales y borre lo que el usuario hubiera escrito. La captura del evento nos permitirá matizar esta operación o añadir acciones a la misma.

5.2.3. EVENTOS DE ENFOQUE

Son eventos asociados a los controles de formulario en los que el usuario puede escribir o elegir opciones. Un control obtiene el **foco** cuando al usar el teclado manejamos dicho control. El enfoque se consigue gracias a la interacción con el dispositivo apuntador (un clic de ratón, por ejemplo) o al cambio de control de formulario gracias a la tecla del tabulador.

Un control de formulario lanza el evento **focus** cuando obtiene el foco y lanza **blur** cuando pierde el foco.

5.2.4. EVENTO DE CAMBIO DE VALOR

El evento **change** es, seguramente el evento más importante en los formularios. Se produce cuando modificamos el valor de cualquier control de formulario. Ejemplos de ello son:

- Cambiar el estado de un botón de radio o de tipo **check**
- Modificar el valor de un cuadro de texto o de contraseña
- Elegir otro valor de una lista de opciones
- Arrastrar un deslizador para modificar su valor

Realmente, el evento se produce cuando se ha modificado el valor y, además, se ha perdido el foco sobre ese control. Es decir, durante el cambio no se lanza el evento.

5.3. PROPIEDADES DE LOS CONTROLES

Para modificar las propiedades de los controles de formulario podemos trabajar sobre sus atributos como hacemos con cualquier otro elemento. Si el usuario ha modificado, por ejemplo, el valor de un cuadro de texto porque ha escrito algo en él, **getAttribute**(“**value**”) no refleja estos cambios. Por ello es mejor usar propiedades que sí reflejen los cambios que hace el usuario. Estas propiedades son:

PROPIEDAD	USO	CONTROLES QUE LA USAN
name	Nombre del control	Prácticamente todos
type	Valor del atributo type	Controles de tipo input
value	Devuleve el valor actual del control	Prácticamente todos
checked	Puede valer true o false. Indica, sobre un control de activar o desactivar, si el control lo está o no.	Botones de radio y botones de tipo check
defaultChecked	Indica el estado inicial de la	Botones de radio y botones de

	propiedad checked en el control	tipo check
disabled	Indica con true o false, si el control está deshabilitado o no	Prácticamente todos
readonly	Indicia con true o false, si el control es de solo lectura	Prácticamente todos
required	Indica con true o false, si es obligatorio o nocambiar el valor del control	Controles de entrada de texto o lista
maxLength	Permite ver y modificar la propiedad que define la anchura máxima del texto	Controles de entrada de texto
min	Valor mínimo posible para el control	Input de tipo numérico o de fecha
max	Valor máximo posible para el control	Input de tipo numérico o de fecha
step	Mínimo valor de cambio del control. Si el valor es 1, los valores avanzan de uno en uno como poco	Input de tipo numérico o de fecha
selectionStart	En controles de entrada de texto indica el índice dentro del texto general en el que comienza la selección actual sobre el texto. Si no hay nada seleccionado, indica la posición del cursor en el texto. Santa Bárbara número El cuadro de texto con selección que se muestra en la imagen anterior tendría estos valores en las propiedades: seleccionStart:6 seleccionEnd:14	Controles de entrada de texto
selectionEnd	En controles de entrada de texto indica el índice dentro del texto general en el que tetmins la selección actual sobre el texto. Si no hay nada seleccionado, indica	Controles de entrada de texto

	la posición del cursor en el texto.	

5.4. MÉTODOS DE LOS CONTROLES

Los controles también ofrecen métodos con los que realizar acciones que simulan la acción del usuario. La lista es la siguiente:

MÉTODO	USO	CONTROLES
Focus()	Fuerza a que el control obtenga el foco	Prácticamente todos
Blur()	Provoca la pérdida de foco en el control	Prácticamente todos
Select()	Selecciona todo el texto en el control y también deja el foco en él	Controles de entrada de texto
SetSelectionRange(inicio, fin)	Selecciona el texto que va desde la posición inicio hasta la posición fin (sin incluir esta última)	Controles de entrada de texto