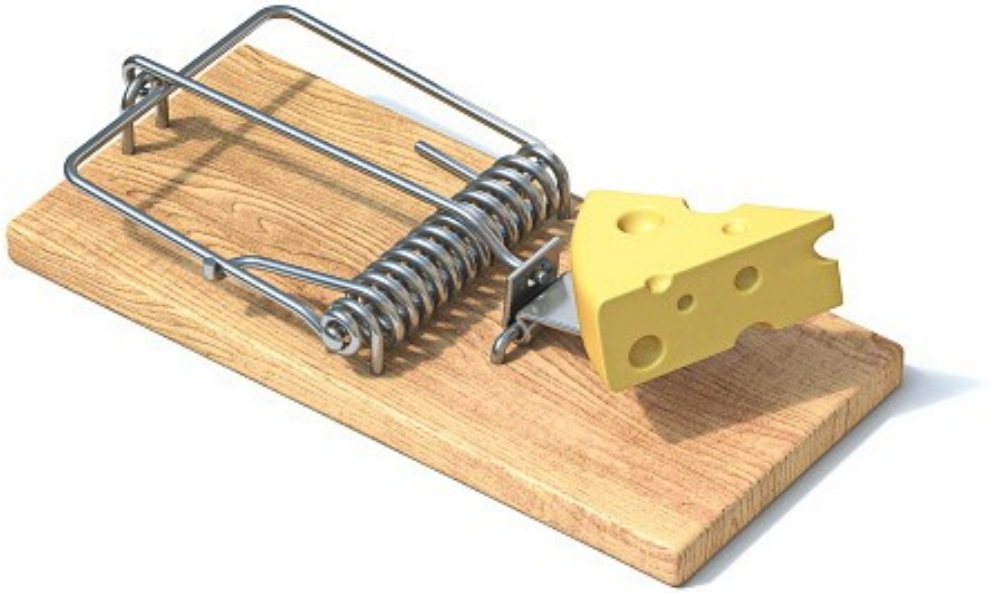
DESARROLLO EN ENTORNO CLIENTE/SERVIDOR

UD7: EVENTOS E INTERACTIVIDAD

JavaScript



UD7 – EVENTOS E INTERACTIVIDAD

1. Evaluación

El presente documento, junto con sus actividades, cubren los siguientes criterios de evaluación:

RESULTADOS DE APRENDIZAJE	CRITERIOS DE EVALUACIÓN
RA0612.5. Desarrolla aplicaciones Web interactivas integrando mecanismos de manejo de eventos.	a) Se han reconocido las posibilidades del lenguaje de marcas relativas a la captura de los eventos producidos. b) Se han identificado las características del lenguaje de programación relativas a la gestión de los eventos. c) Se han diferenciado los tipos de eventos que se pueden manejar. d) Se ha creado un código que capture y utilice eventos. e) Se han reconocido las capacidades del lenguaje relativas a la gestión de formularios Web. f) Se han validado formularios web utilizando eventos. g) Se han utilizado expresiones regulares para facilitar los procedimientos de validación. h) Se ha probado y documentado el código.
RA0612.6. Desarrolla aplicaciones web analizando y aplicando las características del modelo de objetos del documento.	e) Se han asociado acciones a los eventos del modelo.

2. Eventos

Cuando un usuario interactúa con una aplicación, se producen una serie de eventos (de teclado, ratón, etc.) que nuestro código debería manejar de forma adecuada. Hay muchos eventos que pueden ser capturados y procesados, en el siguiente enlace se puede ver la lista completa de eventos disponibles en el DOM: https://www.w3schools.com/jsref/dom_obj_event.asp.

A continuación veremos algunos de los más comunes:

2.1. Eventos de teclado

- **keydown** → El usuario presiona una tecla. Si la tecla se mantiene pulsada durante un tiempo, este evento se generará de forma repetida.
- **keyup** → Se lanza cuando el usuario deja de presionar la tecla
- **keypress** → Más o menos lo mismo que keydown. Acción de pulsar y levantar.

2.2. Eventos del ratón

- **click** → Este evento ocurre cuando el usuario pulsa un elemento (presiona y levanta el dedo del botón → mousedown + mouseup). También normalmente se lanza cuando un

evento táctil de toque (tap) es recibido.

- **dblclick** → Se lanza cuando se hace un doble click sobre el elemento
- **mousedown** → Este evento ocurre cuando el usuario presiona un botón del ratón
- **mouseup** → Este evento ocurre cuando el usuario levanta el dedo del botón del ratón
- **mouseenter** → Se lanza cuando el puntero del ratón entra en un elemento
- **mouseleave** → Se lanza cuando el puntero del ratón sale de un elemento
- **mousemove** → Este evento se llama repetidamente cuando el puntero de un ratón se mueve mientras está dentro de un elemento

2.3. Eventos Touch

- **touchstart** → Se lanza cuando se detecta un toque en la pantalla táctil
- **touchend** → Se lanza cuando se deja de pulsar la pantalla táctil
- **touchmove** → Se lanza cuando un dedo es desplazado a través de la pantalla
- **touchcancel** → Este evento ocurre cuando se interrumpe un evento táctil.

2.4. Eventos de formulario

- **focus** → Este evento se ejecuta cuando un elemento (no sólo un elemento de un formulario) tiene el foco (es seleccionado o está activo).
- **blur** → Se ejecuta cuando un elemento pierde el foco.
- **change** → Se ejecuta cuando el contenido, selección o estado del checkbox de un elemento cambia (sólo <input>, <select>, y <textarea>)
- **input** → Este evento se produce cuando el valor de un elemento <input> o <textarea> cambia.
- **select** → Este evento se lanza cuando el usuario selecciona un texto de un <input> o <textarea>.
- **submit** → Se ejecuta cuando un formulario es enviado (el envío puede ser cancelado).

3. Manejo de Eventos

Hay muchas formas de asignar un código o función a un determinado evento. Vamos a ver las dos formas posibles (para ayudar a entender código hecho por otros), pero el recomendado, y la forma que daremos por válida para este curso, es usar ***event listeners***.

3.1. Manejo de eventos clásico (menos recomendado)

Lo primero de todo, podemos poner código JavaScript (o llamar a una función) en un atributo de un elemento HTML. Estos atributos se nombran como los eventos, pero con el prefjo 'on' (click → onclick). Vamos a ver un ejemplo del evento click.

```
<!DOCTYPE>
<html>
	<head>
		<title>Ejemplo JS</title>
	</head>
	<body>
		<div id="div1">
			<p>
			<input type="text" onclick="alert('¡Me has pulsado!')" />
			</p>
		</div>
		<script src="./ejemplo1.js"></script>
	</body>
</html>
```

Si llamáramos a una función, podemos pasarle parámetros (si es necesario. La palabra reservada this y event se pueden usar para pasar el elemento HTML afectado por el evento y/o el objeto con información del evento a la función.

```
Archivo: ejemplo1.html

<input type="text" id="input1" onclick="inputClick(this, event)" />

Archivo: ejemplo1.js

function inputClick(element, event) {
	// Mostrará "Un evento click ha sido detectado en #input1"
	alert("Un evento" + event.type + " ha sido detectado en #" + element.id);
}
```

Podemos añadir un manejador (función) de evento desde código, accediendo a la propiedad correspondiente (onclick, onfocus, etc.), o asignarles un valor nulo si queremos dejar de escuchar algún evento:

```
let input = document.getElementById("input1");

input.onclick = function(event) {
	// Dentro de esta función, 'this' se refere al elemento
```

```
      alert("Un evento " + event.type + " ha sido detectado en " + this.id);
}
```

3.2. Event listeners (recomendado)

El método para manejar eventos explicado arriba tiene algunas desventajas, por ejemplo, no se pueden gestionar varias funciones manejadoras para un mismo evento, ya que al sobreescribir el atributo, se machaca toda la información que hubiese anteriormente.

Para añadir un event listener, usamos el método **addEventListener** sobre el elemento. Este método recibe al menos dos parámetros. El nombre del evento (una cadena) y un manejador (función anónima o nombre de una función existente).

```
let input = document.getElementById("input1");

input.addEventListener('click', function(event) {
      alert("Un evento " + event.type + " ha sido detectado en " + this.id);
});
```

Podemos añadir tantos manejadores como queramos. Sin embargo, si queremos eliminar un manejador, debemos indicar qué función estamos eliminando.

```
let inputClick = function(event) {
      console.log("Un evento "+event.type+" ha sido detectado en "+this.id);
};

let inputClick2 = function(event) {
      console.log("Yo soy otro manejador para el evento click!");
};

let input = document.getElementById("input1");

// Añadimos ambos manejadores. Al hacer clic, se ejecutarían ambos por orden.
input.addEventListener('click', inputClick);
input.addEventListener('click', inputClick2);

// Así es cómo se elimina el manejador de un evento
input.removeEventListener('click', inputClick);
input.removeEventListener('click', inputClick2);
```

4. Objeto del evento

El objeto del evento es creado por JavaScript y pasado al manejador como parámetro. Este objeto tiene algunas propiedades generales (independientemente del tipo de evento) y otras propiedades específicas (por ejemplo, un evento del ratón tiene las coordenadas del puntero, etc.).

Estas son algunas propiedades generales que tienen todos los eventos:

- **target** → El elemento que lanza el evento (si fue pulsado, etc...).
- **type** → El nombre del evento: 'click', 'keypress', …
- **cancelable** → Devuelve true o false. Si el evento se puede cancelar signifca que llamando a event.preventDefault() se puede anular la acción por defecto (El envío de un formulario, el click de un link, etc…).
- **bubbles** → Devuelve cierto o falso dependiendo de cómo se está propagando el evento (Lo veremos más adelante).
- **preventDefault**() → Este método previene el comportamiento por defecto (cargar una página cuando se pulsa un enlace, el envío de un formulario, etc.)
- **stopPropagation**() → Previene la propagación del evento.
- **stopImmediatePropagation**() → Si el evento tiene más de un manejador, se llama a este método para prevenir la ejecución del resto de manejadores.

Dependiendo del tipo de evento, el objeto tendrá diferentes propiedades:

4.1. MouseEvent

- **button** → Devuelve el botón del ratón que lo ha pulsado (0: botón izquierdo, 1: la rueda del ratón, 2: botón derecho).
- **clientX, clientY** → Coordenadas relativas del ratón en la ventana del navegador cuando el evento fue lanzado.
- **pageX, pageY** → Coordenadas relativas del documento HTML, si se ha realizado alǵun tipo de desplazamiento (scroll), este será añadido (usando clientX y clientY no se añade).
- **screenX, screenY** → Coordenadas absolutas del ratón en la pantalla.
- **detail** → Indica cuántas veces el botón del ratón ha sido pulsado (un click, doble, o triple click).

4.2. KeyboardEvent

- **key** → Devuelve el nombre de la tecla pulsada.
- **keyCode** → Devuelve el código del carácter Unicode en el evento keypress, keyup o

keydown.

- **altKey, ctrlKey, shiftKey, metaKey** → Devuelven si las teclas “alt”, “control”, “shift” o “meta” han sido pulsadas durante el evento (Bastante útil para las combinaciones de teclas como ctrl+c). El objeto MouseEvent también tiene estas propiedades.

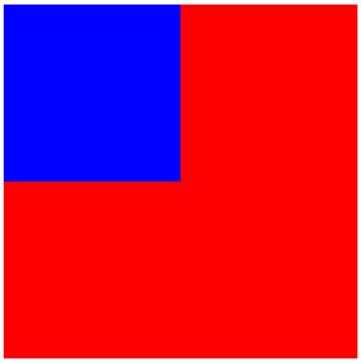
5. Propagación de eventos (bubbling)

Muchas veces, hay elementos en una web que se solapan con otros elementos (están contenidos dentro). Por ejemplo, si pulsamos sobre un párrafo que está contenido en un elemento , ¿el evento se ejecutará en el párrafo, en el o en ambos? ¿Cuál se ejecuta primero?.

Por ejemplo, vamos a ver qué ocurre con estos dos elementos (un elemento dentro de otro) cuando hacemos clic en ellos.

Archivo: ejemplo1.html

```
<div id="div1" style="background-color:red; width:200px; height:200px;">
      <div id="div2" style="background-color:blue; width:100px; height:100px;">
      </div>
</div>
```



Archivo: ejemplo1.js

```
let divClick = function(event) {
      console.log("Has pulsado: " + this.id);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
```

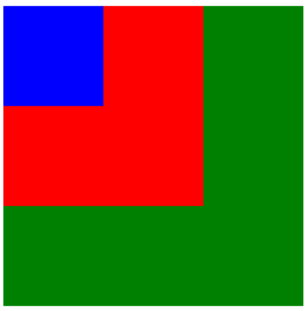
Si hacemos click sobre el elemento rojo , se imprimirá solo “Has pulsado: div1”. Sin embargo, si pulsamos sobre el elemento azul (el cual está dentro del elemento rojo) imprimirá ambos mensajes (#div2 primero). En conclusión, por defecto el elemento el cual está al frente (normalmente un elemento hijo) recibe el evento primero y entonces pasa a ejecutar los manejadores que contiene.

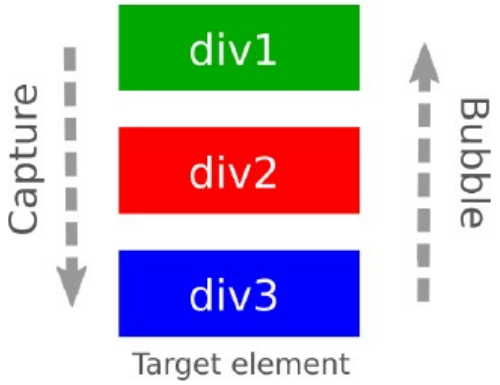
Normalmente, la propagación de eventos va de hijos a padres, pero podemos cambiar este proceso añadiendo un tercer parámetro al método addEventListener y establecerlo a true.

Vamos a ver otro ejemplo:

Archivo: ejemplo1.html

```
<div id="div1" style="background-color:green; width:150px; height:150px;">
  <div id="div2" style="background-color:red; width:100px; height:100px;">
    <div id="div3" style="background-color:blue; width:50px; height:50px;">
    </div>
  </div>
</div>
```





Archivo: ejemplo1.js

```
let divClick = function(event) {
      // eventPhase: 1 -> capture, 2 -> target (objetivo), 3 -> bubble
      console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);
};

let div1 = document.getElementById("div1");
let div2 = document.getElementById("div2");
let div3 = document.getElementById("div3");

div1.addEventListener('click', divClick);
div2.addEventListener('click', divClick);
div3.addEventListener('click', divClick);
```

Por defecto, cuando pulsamos el azul (div3), imprime:

Has pulsado: div3. Fase: 2 → Target element
Has pulsado: div2. Fase: 3 → Bubbling
Has pulsado: div1. Fase: 3 → Bubbling

Si se establece un tercer parámetro a true, se imprimirá:

Has pulsado: div1. Fase: 1 → Propagation
Has pulsado: div2. Fase: 1 → Propagation
Has pulsado: div3. Fase: 2 → Target element

Podemos llamar al método stopPropagation en el evento, no continuará la propagación (si es en la fase de captura) o en (la fase de propagación o bubbling).

```
let divClick = function(event) {
      // eventPhase: 1 -> capture, 2 -> target (clicked), 3 -> bubble
      console.log("Has pulsado: " + this.id + ". Fase: " + event.eventPhase);

event.stopPropagation();
};
```

Ahora, cuando hacemos click el elemento imprimirá solo un mensaje, “Has pulsado: div3. Fase: 2”, si el tercer argumento no se ha establecido (o se ha puesto a false), o “Has pulsado: div1. Fase: 1” si el tercer argumento se ha marcado a true (el elemento padre previene a los hijos de recibirlo en este caso).