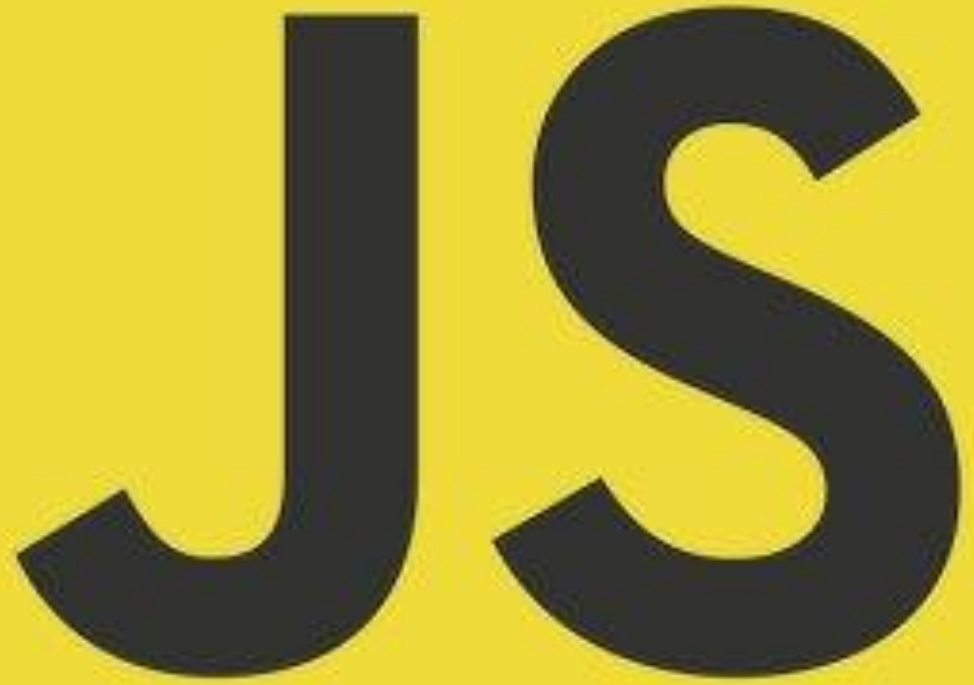
# UT 6_1 - Interacción con el usuario: Eventos




Marina Navarro Pina

cc BY NC SA

# *Eventos*

Los eventos son acciones o sucesos que ocurren en una aplicación web como resultado de la interacción del usuario, como: *pulsar teclas, hacer clic con el ratón o tocar la pantalla*.

Estos eventos son fundamentales para que el código de la aplicación pueda responder adecuadamente a las acciones del usuario

# *Eventos: Tipos comunes - Eventos de la página*

Estos eventos están relacionados con el documento HTML y generalmente afectan al elemento **<body>**:

- **load:** Se activa cuando *el documento HTML ha terminado de cargarse*. Es útil para ejecutar acciones que requieren que el DOM esté completamente cargado, como modificar elementos o consultar datos.
- **DOMContentLoaded:** Se produce *cuando se ha cargado el árbol DOM pero no se han cargado imágenes, hojas de estilo, ni subframes*. Para acciones que no necesitan ni las imágenes ni el CSS.
- **unload:** Ocurre cuando *el documento se destruye*, por ejemplo, al cerrar la pestaña del navegador.
- **beforeunload:** Se dispara *justo antes de que el usuario cierre la página*. Por defecto, muestra un mensaje que pregunta al usuario si realmente desea salir, aunque se pueden ejecutar otras acciones.
- **resize:** Se activa cuando *el tamaño del documento cambia*, comúnmente utilizado cuando la ventana del navegador se redimensiona.

# *Eventos: Tipos comunes - Eventos del teclado*

Estos eventos se generan a partir de la interacción del usuario con el teclado:

- **keydown:** Se activa cuando *el usuario presiona una tecla*. Si la tecla se mantiene presionada, este evento se generará repetidamente.
- **keyup:** Se lanza cuando *el usuario deja de presionar una tecla*.
- **keypress:** Similar a keydown, se activa *durante la acción de pulsar y levantar una tecla*.

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

# *Eventos: Tipos comunes - Eventos del ratón*

Estos eventos están relacionados con la interacción del usuario mediante el ratón:

- **click:** Ocurre cuando *el usuario hace clic en un elemento* (presiona y levanta el botón del ratón). También se activa con eventos táctiles (tap).
- **dblclick:** Se activa cuando *se realiza un doble clic en un elemento*.
- **mousedown:** Se dispara cuando *el usuario presiona un botón del ratón*.
- **mouseup:** Se activa cuando *el usuario levanta el botón del ratón*.
- **mouseenter-mouseover:** Se lanza cuando *el puntero del ratón entra en un elemento*.
- **mouseleave-mouseout:** Se activa cuando *el puntero del ratón sale de un elemento*.
- **mousemove:** Se llama repetidamente cuando *el puntero del ratón se mueve dentro de un elemento*.

Si hacemos doble click sobre un elemento la secuencia de eventos que se produciría es:

*mousedown -> mouseup -> click -> mousedown -> mouseup -> click -> dblclick*

# *Eventos: Tipos comunes - Eventos touch (de toque)*

Estos eventos son específicos *para dispositivos con pantallas táctiles*:

- **touchstart:** Se activa cuando se detecta *un toque en la pantalla*.
- **touchend:** Ocurre cuando *se deja de tocar la pantalla*.
- **touchmove:** Se lanza cuando *un dedo se desplaza sobre la pantalla*.
- **touchcancel:** Ocurre cuando *un evento táctil se interrumpe*.

# *Eventos: Tipos comunes - Eventos de formulario*

Estos eventos están relacionados con la *interacción del usuario con formularios*:

- **focus:** Se ejecuta cuando *un elemento recibe el foco* (es seleccionado o activo).
- **blur:** Se activa cuando *un elemento pierde el foco*.
- **change:** Se dispara cuando *el contenido, selección o estado de un elemento* (como un checkbox) *cambia*.
- **input**: Se produce cuando *el valor de un elemento <input> o <textarea> cambia*.
- **select:** Se activa cuando *el usuario selecciona texto en un <input> o <textarea>*.
- **submit:** Se ejecuta cuando *un formulario es enviado*, y este envío puede ser cancelado.

# *Eventos: Manejo de eventos clásico (menos recomendado)*

Mediante el uso de atributos HTML que comienzan con "on“:

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

En este caso, al hacer clic en el input, se ejecutará la función que muestra una alerta.
Una mejora era llamar a una función:

```
<input type="button" id="boton1" onclick="clicked()" />
```

```
function clicked() {
  alert('Se ha pulsado');
}
```

# *Eventos: Manejo de eventos clásico (menos recomendado)*

Como se trata de poner un atributo al elemento podemos usar DOM para evitar “ensuciar” con código la página HTML:

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

*Si asociamos un evento a un elemento que aún no existe (porque aún no lo ha renderizado el navegador) no se produce ningún error, pero no tendrá evento. Para evitar esto es necesario poner escuchas para saber cuando el está cargado el árbol del DOM.*

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

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

# *Eventos: Manejo de eventos - Event listeners*
*(recomendado)*

Este método permite **añadir múltiples manejadores para un mismo evento** y es más limpio y organizado.

Se utiliza el método **`addEventListener`**, que toma al menos dos parámetros: el *nombre del evento* (sin 'on') y *la función* que se ejecutará cuando ocurra el evento (si se especifica nombre, sin **()** ).

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

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

# *Eventos: Manejo de eventos - Event listeners* *(recomendado)*

Debemos estar seguros de que se ha creado el árbol DOM antes de poner un escuchador; por lo que se recomienda ponerlos siempre dentro una función asociada a *window.addEventListener("load", ...)* o *document.addEventListener("DOMContentLoaded", ...)*:

```
window.addEventListener('load', () => {
  document.getElementById('acepto').addEventListener('click', () => {
    alert('Se ha aceptado');
  })
})
```

```
document.addEventListener('DOMContentLoaded', () => {
  document.getElementById('acepto').addEventListener('click', () => {
    alert('Se ha aceptado');
  })
})
```

# *Eventos: Manejo de eventos - Event listeners (recomendado)*

Una ventaja de esta forma de poner escuchadores es que podemos poner varios escuchadores para el mismo evento y se ejecutarán todos ellos:

```
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!")
}

input.addEventListener("click", inputClick)
input.addEventListener("click", inputClick2)

// Para eliminar los manejadores
input.removeEventListener("click", inputClick)
input.removeEventListener("click", inputClick2)
```

Para eliminar un escuchador se usa el método **`removeEventListener`**.
*No se puede eliminar un escuchador si hemos usado una función anónima; debemos usar una función con nombre en tal caso.*

*Ejemplo1*

# *Eventos: Objetos*

Al producirse un evento se generan automáticamente en su función manejadora **2 objetos**:

- **this:** hace referencia al elemento que ha disparado el evento. Por ejemplo, si un botón es clicado, **this** dentro de la función manejadora se refiere a ese botón. *El valor de **this** puede ser sobreescrito en el contexto de la función manejadora.*

- **event:** es un objeto y la función manejadora lo recibe como parámetro. Tiene propiedades y métodos que nos dan información sobre el evento.

# *Eventos: Objeto Event - Propiedades Generales*

- **.type**: Indica el tipo de evento que ha ocurrido, como 'click', 'keypress', 'submit', etc. Esto permite saber qué *acción específica se ha realizado*.

- **.target**: Esta propiedad hace referencia al elemento que ha disparado el evento. Por ejemplo: *si un usuario hace clic en un botón,* `event.target` *será el botón en cuestión*.

- **.currentTarget**: Este es el elemento que tiene el escuchador del evento. Normalmente, es el mismo que this, pero *puede diferir si el evento se propaga desde un elemento hijo*.

# *Eventos: Objeto Event - Propiedades Generales*

- .cancelable: Esta propiedad devuelve un valor booleano (true o false) que indica si el evento puede ser cancelado. Si es true, se puede llamar a **`event.preventDefault()`** para *evitar que se ejecute la acción predeterminada asociada con el evento* (como el envío de un formulario o la navegación a un enlace).

- .preventDefault(): Este método se utiliza para prevenir el comportamiento por defecto del evento. Por ejemplo, *si se usa en un evento de clic en un enlace, evitará que el navegador navegue a la URL del enlace*.

- .stopPropagation(): Este método detiene la propagación del evento hacia los elementos padres. Es útil *cuando se quiere evitar que otros manejadores de eventos en la jerarquía de elementos se ejecuten*.

# *Eventos: Objeto Event - Propiedades Específicas (MouseEvent)*

- **.button**: Indica qué botón del ratón fue presionado (0 para el botón izquierdo, 1 para la rueda del ratón, 2 para el botón derecho).
- **.clientX, .clientY**: Proporcionan las coordenadas del puntero del ratón en relación con la **ventana** del navegador en el momento en que se disparó el evento.
- **.pageX, .pageY**: Proporcionan las coordenadas del puntero del ratón en relación con el **documento HTML**, teniendo en cuenta cualquier desplazamiento (scroll) que se haya realizado.
- **.screenX, .screenY**: Proporcionan las coordenadas absolutas del puntero del ratón en la **pantalla**.
- **.detail**: Indica cuántas veces se ha pulsado el botón del ratón (por ejemplo, un clic, doble clic, etc.).

# *Eventos: Objeto Event - Propiedades Específicas (KeyboardEvent)*

- **.key:** Devuelve el nombre de la tecla que fue pulsada.

- **.keyCode:** Proporciona el código del carácter Unicode correspondiente a la tecla que fue presionada durante los eventos keypress, keyup o keydown.

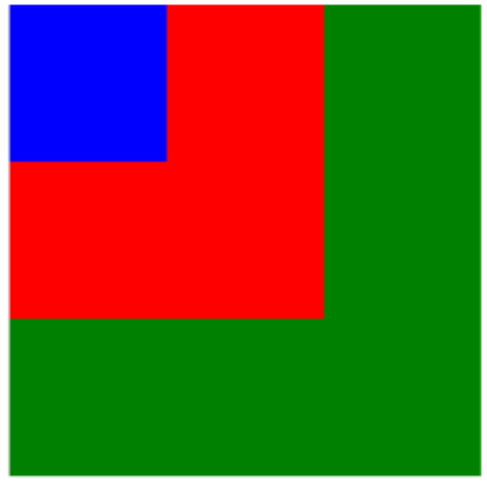
- **.altKey, .ctrlKey, .shiftKey, .metaKey:** Estas propiedades devuelven un valor booleano que indica si las teclas "Alt", "Control", "Shift" o "Meta" (tecla de Windows) estaban presionadas durante el evento. Esto es útil *para detectar combinaciones de teclas, como Ctrl+C para copiar*.

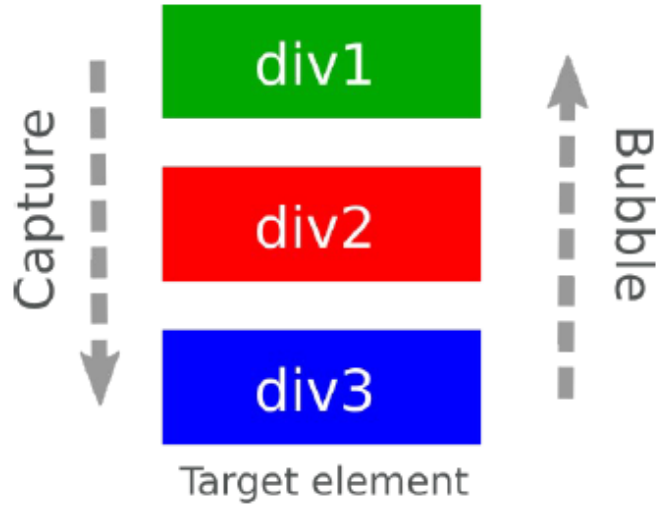
# *Eventos: Propagación (bubbling)*

Comportamiento en el que un evento que ocurre en un elemento hijo se propaga hacia arriba a través de sus elementos padres.

Por ejemplo, si se hace clic en un párrafo dentro de un , el evento se ejecutará primero en el párrafo y luego en el .

*Ejemplo2*





# *Eventos: Eventos personalizados*

Podemos mediante código lanzar manualmente cualquier evento sobre un elemento con el método **`dispatchEvent()`** e incluso crear eventos personalizados:

```
const event = new Event('build');

// Listen for the event.
elem.addEventListener('build', (e) => { /* ... */ });

// Dispatch the event.
elem.dispatchEvent(event);
```

# *Eventos - Referencias*

- https://www.w3schools.com/jsref/dom_obj_event.asp
- https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Events