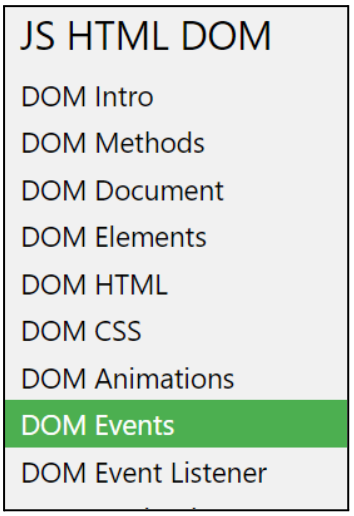
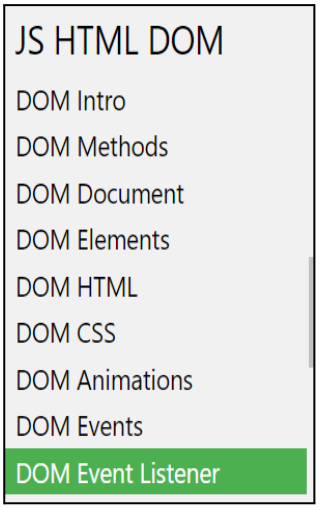
DOM-Tarea5 - event handlers - manejadores de eventos y captura de eventos

DAM2 - DWEB

Objetivo: practicar con DOM (Eventos) y JavaScript HTML DOM EventListener
Realiza una lectura comprensiva de los enlaces propuestos revisando y probando los ejemplos planteados.





Un evento de JavaScript por sí mismo carece de utilidad. Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o el código JavaScript que se definen para cada evento se denominan **manejadores de eventos** (*event handlers* en inglés) y como JavaScript es un lenguaje muy flexible, existen varias formas de indicar los manejadores:

1. Manejadores definidos como **atributos de los elementos HTML**
   a. escribiendo el código javascript en el marcado

```
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
```

   b. invocando a funciones javascript también desde el marcado

```
<h1 onclick="changeText(this)">Click on this text!</h1>
<script>
function changeText(id) {
  id.innerHTML = "Ooops!";
}
</script>
```

2. Manejadores definidos **usando el DOM HTML.** Son los denominados **manejadores "semánticos"**. Se especifican fuera del marcado como propiedades Dom.

```
document.getElementById("myBtn").onclick = displayDate;
```

Veamos cada una de estas formas:

1a. Manejadores de eventos como atributos HTML
El evento se define en el marcado HTML empleando el "atributo de evento" del elemento HTML seguido de código JavaScript. Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento.
Ejemplo:

```
<h1 onclick="this.innerHTML = 'Ooops!'">Click on this text!</h1>
```

*Nota acerca de la palabra clave **this:***
Se trata de una propiedad que apunta al objeto en uso. Así que si utilizas esta palabra clave, cuando se activa un evento el objeto al que apunta será el elemento que desencadena el evento. Ejemplo:
Púlseme

En este otro ejemplo, cuando el usuario pincha sobre el elemento se muestra un mensaje y cuando el usuario pasa el ratón por encima del elemento, se muestra otro mensaje:

```
<div onclick="console.log('Has pinchado con el ratón');" onmouseover="conso
le.log('Acabas de pasar el ratón por encima');">
    Puedes pinchar sobre este elemento o simplemente pasar el ratón por enc
ima
</div>
```

1b. Manejadores de eventos como atributos HTML con funciones externas
El evento se define en el marcado HTML empleando el atributo de evento del elemento HTML pero pasando a dicho atributo una función JavaScript. Ejem.:

```
<h1 onclick="changeText(this)">Click on this text!</h1>
```

Utilizar los atributos HTML o funciones externas para añadir manejadores de eventos tiene un grave inconveniente: "ensucian" el código HTML de la página. Como es conocido, al crear páginas web se recomienda separar los contenidos (HTML) de la presentación (CSS). En lo posible, también se recomienda separar los contenidos (HTML) de la programación (JavaScript). Mezclar JavaScript y HTML complica excesivamente el código fuente de la página, dificulta su mantenimiento y reduce la semántica del documento final producido.

Afortunadamente, existe un método alternativo para definir los manejadores de eventos de JavaScript. Esta técnica consiste en asignar las funciones externas mediante las

propiedades DOM de los elementos HTML, tenemos así los llamados **manejadores de eventos semánticos** que vemos a continuación.

2. Manejadores de eventos semánticos

El DOM HTML permite asignar eventos a elementos HTML usando JavaScript. De esta manera la definición queda fuera del marcado; el evento se asigna mediante *las propiedades DOM de los elementos HTML*.

Ejem.:

```
<h1 id="myh1">Click on this text!</h1>
....
<script>
document.getElementById("myh1").onclick = changeText;
...
</script>
```

El código HTML resultante es más "limpio", ya que no se mezcla con el código JavaScript. La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento HTML mediante el atributo id.
2. Crear la función de JavaScript encargada de manejar el evento.
3. Asignar la función al evento concreto del elemento HTML mediante DOM como una "propiedad DOM del elemento"

```
document.getElementById("myh1").onclick = changeText;
```

*Ojo!!, al asignar la función externa has de **indicar solamente el nombre de la función**, es decir, prescindir de los paréntesis al asignar la función. Si se añaden los paréntesis al final, en realidad se está invocando la función y asignando el valor devuelto por la función al evento onclick de elemento. Y lo que queremos no es invocar la función sino asignar la función al evento.*

El único inconveniente de este método es que los manejadores se asignan mediante las funciones DOM, que solamente se pueden utilizar después de que la página se ha cargado completamente. Por tanto, para que la asignación de los manejadores no resulte errónea, es necesario asegurarse de que la página se ha cargado. Una de las formas más sencillas de asegurarnos que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar **el evento onload**. Esta técnica utiliza una función anónima para asignar algunas instrucciones al evento onload de la página (en este caso se ha establecido mediante el objeto window). Para asegurarnos que cierto código se va a ejecutar después de que la página se haya cargado, sólo es necesario incluirlo en el interior de la siguiente construcción:

```
window.onload = function() {
   ...
}
```

Ejemplo:

```
window.onload = function() {
  document.getElementById("pinchable").onclick = muestraMensaje;
}
```

> *Observa que el efecto es el mismo que si empleas `<body onload="">`, sólo que si lo haces así ten en cuenta que estarás definiendo el manejador en el marcado y no debes olvidar que es un estándar aceptado separar contenido, diseño y comportamiento.*

De los tres métodos hasta aquí expuestos, el que más se acerca al ideal de separar las capas de contenido, presentación y comportamiento de un documento es el tercero: definir los manejadores usando el DOM HTML. Si queremos acercarnos a este ideal deberíamos mantener el marcado limpio de atributos como por ejemplo *"onclick"*, que pertenece propiamente al comportamiento. Y ¿cómo podemos lograr esto? la respuesta es ***"escuchando"*.**

La especificación **DOM** define para ello dos métodos, denominados ***addEventListener()*** **y** ***removeEventListener()*** que nos permiten asociar y desasociar manejadores de eventos.

Sintaxis:

```
element.addEventListener(event, function, useCapture);
```

Requiere tres parámetros:

- el tipo de evento ("`click`", "`mousedown`",...); aquí puedes consultar la relación de eventos HTML DOM
- una referencia a la función encargada de procesar el evento *(puede ser una función anónima o función nombrada)* y
- un tercer parámetro opcional que indica el tipo de "propagación del evento" utilizado. Toma uno de los siguientes valores:
  - *"false"*: el evento se propaga mediante *"Burbujeo"*; es el valor por defecto cuando no se especifica este parámetro
  - *"true":* el evento se propaga mediante *"Captura"*

> *"Captura" y "Burbujeo" son las dos formas de propagación de eventos que reconoce el DOM HTML. Ambos términos se refieren al* ***orden en que se reconoce la ocurrencia de los eventos.*** *Con el burbujeo el evento del elemento más interno es el primero que se maneja, y con la captura es el evento del elemento más externo el que primero se maneja. Asegúrate que entiendes el ejemplo planteado en el tutorial relativo a este tema (¿qué evento "clic" debe manejarse primero, el de un elemento <p> o el de un elemento que contiene al <p>?...)*

Ejemplo con función anónima:

```
document.getElementById("myBtn").addEventListener("click", function() {
  alert("Hello World!");
});
```

Ejemplo con función nombrada:

```
element.addEventListener("click", myFunction);
function myFunction() {
  alert ("Hello World!");
}
```

El método *addEventListener()* con el que vas a trabajar en esta tarea, permite indicar al agente de usuario que permanezca atento a la interacción de un usuario sobre un elemento en concreto, sin necesidad de tocar un solo carácter del marcado. Cuando se utiliza este método, JavaScript se separa más del marcado HTML aportando mejor legibilidad, permitiendo además agregar escuchas de eventos incluso cuando no controlamos el marcado HTML.

El método *addEventListener()* permite agregar varias veces el mismo tipo de evento a un mismo elemento, sin sobrescribir los eventos existentes, facilitando el control de cómo reacciona el evento al burbujeo. Ejem. podemos agregar dos eventos click a un mismo elemento:

```
element.addEventListener("click", myFunction);
element.addEventListener("click", mySecondFunction);
```

También podemos agregar eventos de diferente tipo a un mismo elemento:

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

Ojo!, porque no se utiliza el prefijo "on": *se usa "click" en lugar de "onclick"*.

De igual manera, se puede eliminar fácilmente un detector de eventos utilizando el método *removeEventListener ()*.

Puesta en práctica:
Tarea 5-1.html
Descarga este archivo y y sin tocar el código HTML:

- Captura el evento *mousemover* a nivel de la marca <*div*> cuyo *id="caja",* de manera que se dispare una acción cada vez que el mouse se desplace dentro de dicho div. Concretamente cuando esto ocurra, has de mostrar en el párrafo "*texto1"* un número aleatorio entre 0 y 1. método random()

- Captura los eventos necesarios en el botón *"b1"* para que:
  - Cuando el ratón entre y salga del botón se muestren en "*texto2"* mensajes indicativos de ello.
  - Cuando se haga clic en el botón se eliminen los controladores de evento creados y se muestre en *"texto2"* el mensaje **STOP**.

- Click en el botón *"b2"* debe mostrar en un *alert* la suma de dos números solicitados al usuario. Revisa el apartado "paso de parámetros": si precisas pasar parámetros a la función que se va a invocar cuando ocurra el evento, debes usar una "función anónima" que llame a la función especificada con los parámetros.

Tarea5-2.html

Descarga este archivo y sin tocar el código HTML captura los eventos:

- ★ clicks sobre las filas 1 y 2 de la tabla
- ★ clicks sobre las celdas dos y cuatro de la tabla

Comportamiento a codificar:

- ★ el click sobre la celda dos debe ser reconocido y disparado su manejador antes que el de la fila a la que pertenece,
- ★ el de la celda cuatro por el contrario, se ha de disparar después del de la fila que lo contiene.

Tarea5-3.html

Aplica dinamismo, el que tu quieras, a una nueva página que crees o bien a una página que ya tengas construida de tareas anteriores (o creada en el módulo LMGI de primero). Para ello debes lograr que tu página reaccione a los siguientes eventos u otros que tú decidas:

- clic sobre un elemento HTML que no sea button
- evento onchange
- eventos onmouseover y onmouseout
- eventos onmousedown, onmouseup
- evento onfocus
- ...

Pon en práctica los diversos métodos mostrados en el tutorial para definir los manejadores de eventos.