

***RECUERDA PONER A GRABAR LA  
CLASE***





***¿DUDAS DEL ON-BOARDING?***

**MIRALO AQUI**



**Clase 09. JAVASCRIPT**

# ***EVENTOS***



## ***OBJETIVOS DE LA CLASE***

- Comprender qué son los eventos y para qué sirven.
- Entender cómo escuchar un evento sobre el DOM.
- Conocer los eventos más comunes.
- Identificar qué es la información del evento.

# ***GLOSARIO:***

## ***Clase 8***

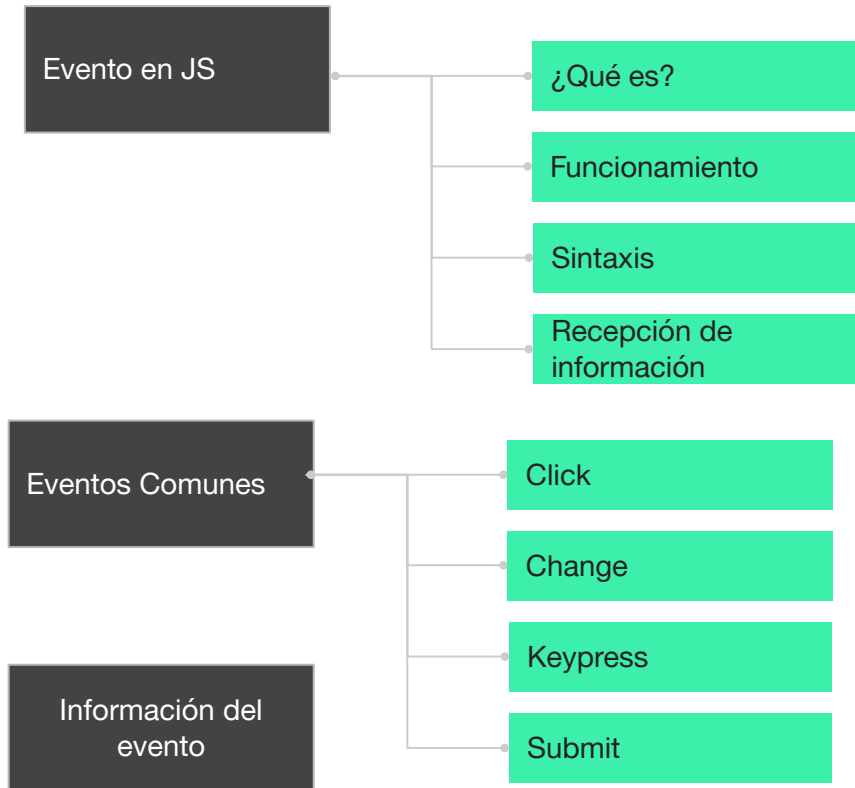
**DOM o Modelo de Objetos del Documento:** es lo que permite interactuar a JS con los diferentes elementos HTML de una web, como también poder operar sobre ellos y modificarlos.

**JavaScript Object Notation (JSON);** es un formato basado en texto plano, para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web.

# ***MAPA DE CONCEPTOS***

# MAPA DE CONCEPTOS CLASE 9

¡Para  
recordar!



# ***CRONOGRAMA DEL CURSO***

## Clase 8



### DOM



EJEMPLOS EN VIVO



INTERACTUAR CON HTML

## Clase 9



### Eventos



EJEMPLOS EN VIVO



INCORPORAR EVENTOS

## Clase 10



### Workshop I



EJEMPLOS EN VIVO



SEGUNDA ENTREGA DEL  
PROYECTO FINAL





# ***GUIÓN DE LA CLASE***

Accede al material complementario [aquí](#).

# ***EVENTOS EN JS***



## ***¿QUÉ ES UN EVENTO?***

Los eventos son la manera que tenemos en Javascript de **controlar las acciones de los usuarios, y definir un comportamiento de la página o aplicación cuando se produzcan.**

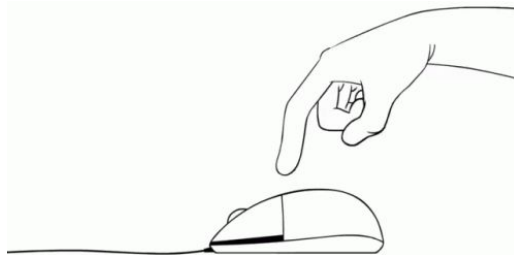
Con Javascript podemos definir qué es lo que pasa cuando se produce un evento, cómo podría ser un clic en cierto elemento, o escribir en un campo.

# ¿CÓMO FUNCIONA?

JavaScript permite **asignar una función a cada uno de los eventos**.

De esta forma, cuando se produce cualquier evento, JavaScript ejecuta su función asociada. Este tipo de funciones se denominan *event handlers* en inglés, y en castellano por "manejadores de eventos".

Los eventos se asocian a cada elemento al cual se lo quiere "escuchar".



# DEFINIR EVENTOS: OPCIÓN 1

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
    <button id="btnPrincipal">CLICK</button>
    <script>
      let boton = document.getElementById("btnPrincipal")
      boton.addEventListener("click", respuestaClick)
      function respuestaClick () {
        console.log("Respuesta evento");
      }
    </script>
  </body>
</html>
```

El método `addEventListener()` permite definir qué evento escuchar sobre cualquier elemento en el código HTML.

El primer parámetro corresponde al nombre del evento y el segundo a la función de respuesta.

# DEFINIR EVENTOS: OPCIÓN 2

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
    <button id="btnPrincipal">CLICK</button>
    <script>
      let boton = document.getElementById("btnPrincipal")
      boton.onclick = () =>{console.log("Respuesta 2")}
    </script>
  </body>
</html>
```

Emplear una propiedad del nodo para definir la respuesta al evento. Las propiedades se identifican con el nombre del evento y el prefijo *on*. También es posible emplear funciones anónimas para definir los manejadores de eventos.

# ***SINTAXIS: OPCIÓN 3***

Determinar el evento especificando el manejador de evento en el atributo de una etiqueta HTML. La denominación del atributo es idéntica al de la propiedad de la opción 2 (prefijo *on*)

```
<input type="button" value="CLICK2" onclick="alert('Respuesta 3');" />
```

La función puede declararse entre la comillas o bien tomarse una referencia existen en el script.

# ***¿Y CUÁL CONVIENE USAR?***

Las opciones 1 y 2 son las recomendadas, si bien se pueden presentar casos de aplicación específico (por ejemplo, en la opción 1 el nombre del evento puede venir de una variable al usar la propiedad, y esto no puede hacerse en la 2), se identifican como formas de definición de evento equivalentes.

La opción 3, aunque es de fácil implementación, no es recomendada para proyectos en producción ya que no es considerada una buena práctica declarar funciones y código JavaScript dentro del HTML.



Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

***CODER HOUSE***



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

# ***EVENTOS MÁS COMUNES***

# EVENTOS DEL MOUSE

Los eventos del mouse o MouseEvent son aquellos que se producen por la interacción del usuario con el mouse. Entre ellos destacamos:

- **mousedown/mouseup**: Se oprime/suelta el botón del ratón sobre un elemento.
- **mouseover/mouseout**: El puntero del mouse se mueve sobre/sale del elemento.
- **mousemove**: El movimiento del mouse sobre el elemento activa el evento.
- **click**: Se activa después de **mousedown** o **mouseup** sobre un elemento válido.

```
//CODIGO HTML DE REFERENCIA
```

```
<button id="btnMain">CLICK</button>
```

```
//CODIGO JS
```

```
let boton = document.getElementById("btnMain");
```

```
boton.onclick = () => {console.log("Click")};
```

```
boton.onmousemove = () => {console.log("Move")}
```

# EVENTOS DE TECLADO

Los eventos de teclado o KeyboardEvent describen una interacción del usuario con el teclado son aquellos que se producen por la interacción del usuario con el teclado.

Entre ellos destacamos:

- **keydown**: Cuando se presiona.
- **keyup**: Cuando se suelta una tecla.

```
//CODIGO HTML DE REFERENCIA
<input id = "nombre" type="text">
<input id = "edad" type="number">

//CODIGO JS
let input1 = document.getElementById("nombre");
let input2 = document.getElementById("edad");
input1.onkeyup = () => {console.log("keyUp")};
input2.onkeydown = () => {console.log("keyDown")};
```

# EVENTO CHANGE

El evento **change** se activa cuando se detecta un cambio en el valor del elemento. Por ejemplo, mientras estamos escribiendo en un input de tipo texto, no hay evento **change**, pero cuando pasamos a otra sección de la aplicación entonces ocurre el evento *change*.

```
//CODIGO HTML DE REFERENCIA
<input id = "nombre" type="text">
<input id = "edad" type="number">

//CODIGO JS
let input1 = document.getElementById("nombre");
let input2 = document.getElementById("edad");

input1.onChange = () => {console.log("valor1")};
input2.onChange = () => {console.log("valor2")};
```

# EVENTO SUBMIT

El evento `submit` se activa cuando el formulario es enviado, normalmente se utiliza para validar el formulario antes de ser enviado al servidor o bien para abortar el envío y procesarlo con JavaScript.

```
//CODIGO HTML DE REFERENCIA
<form id="formulario">
  <input type="text">
  <input type="number">
  <input type="submit" value="Enviar">
</form>

//CODIGO JS
let miFormulario = document.getElementById("formulario");
miFormulario.addEventListener("submit", validarFormulario);

function validarFormulario(e){
  e.preventDefault();
  console.log("Formulario Enviado");
}
```

# ***OTROS EVENTOS***

Existen otros eventos que podemos utilizar. Algunos son eventos estándar definidos en las especificaciones oficiales, mientras que otros son eventos usados internamente por navegadores específicos.

La forma de declararlos es similar a lo abordado en esta clase, lo que necesitamos aprender es **bajo qué condición se disparan los eventos que buscamos implementar.**

Para conocer más eventos se recomienda verificar la [referencia de eventos](#) en la documentación.



# ***INFORMACIÓN DEL EVENTO***

# ***INFORMACIÓN DEL EVENTO***

En algunos casos, necesitamos obtener **información contextual** del evento para poder realizar acciones. Por ejemplo, ante el evento submit necesitamos prevenir el comportamiento por defecto para operar correctamente.

Para esto existe en JavaScript el objeto **event**.

En todos los navegadores modernos se **crea de forma automática** un parámetro que se pasa a la función manejadora, por lo que no es necesario incluirlo en la llamada.

Ese parámetro puede o no usarse en el evento, pero siempre estará disponible en la llamada.

```
//CODIGO HTML DE REFERENCIA
<form id="formulario">
  <input type="text">
  <input type="number">
  <input type="submit" value="Enviar">
</form>

//CODIGO JS
let miFormulario = document.getElementById("formulario");
miFormulario.addEventListener("submit", validarFormulario);

function validarFormulario(e){
  //Cancelamos el comportamiento del evento
  e.preventDefault();
  //Obtenemos el elemento desde el cual se disparó el evento
  let formulario = e.target
  //Obtengo el valor del primero hijo <input type="text">
  console.log(formulario.children[0].value);
  //Obtengo el valor del segundo hijo <input type="number">
  console.log(formulario.children[1].value);
}
```

# ***EJEMPLO APLICADO: DATOS DEL FORMULARIO USANDO EVENT***

Ejemplo  
en vivo



***¡VAMOS A PRACTICAR LO VISTO!***

***CODER HOUSE***



# ***INCORPORAR EVENTOS***

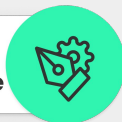
Con lo que vimos sobre DOM, ahora puedes sumarlo a tu proyecto, para interactuar entre los elementos HTML y JS.

# INCORPORAR EVENTOS

**Formato:** Página HTML y código fuente en JavaScript. Debe identificar el apellido del estudiante en el nombre de archivo comprimido por “claseApellido”

**Sugerencia:** Es posible asociar más de un evento a un elemento y se pueden emplear función comunes, anónimas y arrow para los manejadores de eventos.

Desafío  
entregable



**>> Consigna:** Con lo que vimos sobre DOM, ahora puedes sumarlo a tu proyecto, para interactuar entre los elementos HTML y JS. Es decir, asociar eventos que busquemos controlar sobre los elementos de la interfaz de nuestro simulador

**>>Aspectos a incluir en el entregable:**

Archivo HTML y Archivo JS, referenciado en el HTML por etiqueta `<script src="js/miarchivo.js"></script>`, que incluya la definición de un algoritmo en JavaScript que opere sobre el DOM manejando eventos.

**>>Ejemplo:**

- Cuando el usuario completa algún dato, por ejemplo cantidad de cuotas, se captura ese dato y se agregan elementos al DOM mediante JS.
- Capturar la tecla ENTER para confirmar alguna acción.



## ***GENERAR HTML***

Codifica un script que utilice la información de un array de objetos, para generar elementos del DOM de forma procedural.

# GENERAR HTML

**Formato:** código fuente en JavaScript

[Sublime Text](#) o [VisualStudio](#).

**Sugerencia:** en esta actividad, puedes generar los nuevos elementos del DOM mientras recorres un array de objetos, creando en cada iteración (con `createElement`) uno o más elemento html, agregándolos como hijos al body u otro nodo (con `appendChild`).

Desafío  
Complementario



**>> Consigna:** codifica un script cuyas instrucciones permitan generar de forma dinámica una sección del HTML. Los valores que alimentan este proceso comprenden una array de objetos, cuyos datos deberán incluirse empleando métodos del DOM y elementos apropiados para su representación.

**>>Aspectos a incluir en el entregable:**

Archivo HTML y archivo JavaScript referenciado, que incluya la definición un array de objetos, la declaración y llamada de una función que genere proceduralmente una sección del HTML.

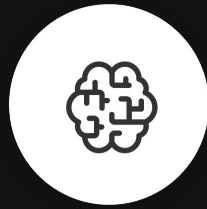
**>>Ejemplo de estructura HTML resultante:**

- 1) Generar títulos y párrafos a partir de un array de “Publicaciones”.
- 2) Generar cards y botones a partir de un array de “Productos”.
- 3) Generar imágenes y badges a partir de un array de “Personas”.



***¿PREGUNTAS?***





## ***¡PARA PENSAR!***

*¿Te gustaría comprobar tus conocimientos de la clase?*

Te compartimos a través del chat de zoom  
el enlace a un breve quiz de tarea.

*Para el profesor:*

- Acceder a la carpeta "Quizzes" de la camada
  - Ingresar al formulario de la clase
  - Pulsar el botón "Invitar"
  - Copiar el enlace
- Compartir el enlace a los alumnos a través del chat



# ***RECURSOS:***

Material  
ampliado



- Ejemplos interactivos: Eventos |  
*Introducción a eventos del navegador*  
*Acciones predeterminadas del navegador*  
*Eventos change, input, cut, copy, paste*  
*Formularios: evento y método submit*
- Documentación |  
*Documentación Eventos*  
*Referencia de Eventos*

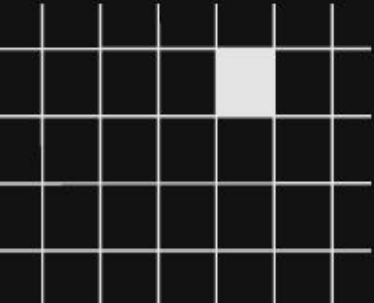
Disponible en [nuestro repositorio](#).

***CODER HOUSE***



# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Eventos: qué son y para qué sirven.
  - Tipos de eventos.
  - Parámetro del evento.
- 



***OPINA Y VALORA ESTA CLASE***