

JavaScript desde cero

Módulo 4

Eventos

¿Qué son los eventos?

En una aplicación de *software*, hablamos de **eventos** cuando nos referimos a la **interacción generada entre el usuario y el *software***.



Eventos en el desarrollo web

Y si hablamos de desarrollo web, los eventos que se producen en este terreno, **son múltiples**:

- *Scroll* del usuario por la página.
- Clic en botones y puntos de menú.
- Escribir datos en un formulario web.
- Subir o descargar archivos.
- Reproducir audio o video.
- Entre otros.

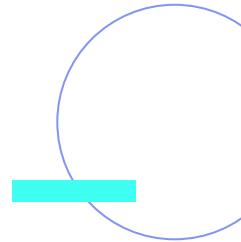
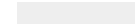
Todos los eventos mencionados en el slide anterior, son generados por el usuario, durante el **proceso de interacción**.

Y por cada evento del usuario, existe una *reacción* de la aplicación como respuesta.

Por ejemplo: ante el evento *clic*, la aplicación responde con una acción asociada, para que el usuario consiga hacer efectiva su necesidad.

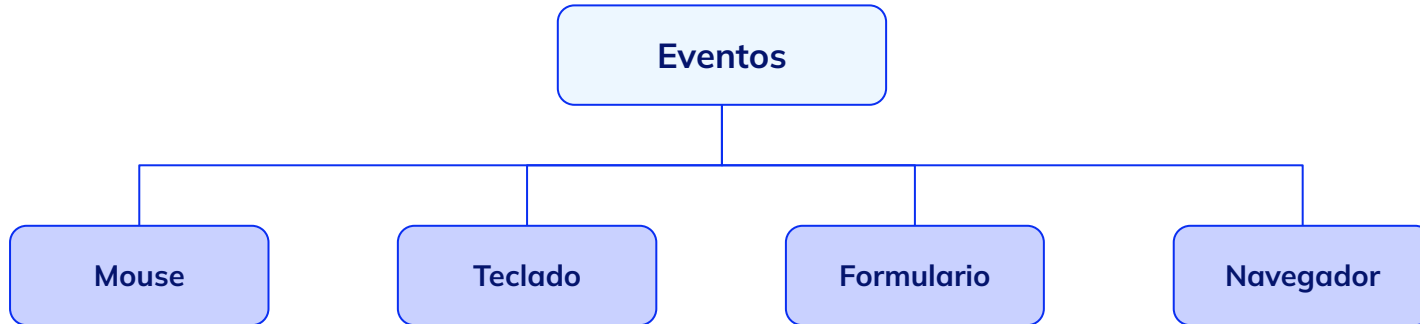
JS nos brinda mecanismos para detectar cuándo ocurren diferentes eventos, y por lo tanto que podamos definir un código específico para actuar en consecuencia ante los mismos.

El listado de eventos en JS es muy extenso. Lo bueno, es que la mayoría es común a cualquier elemento HTML, por lo tanto, es más fácil pensar en cómo aplicarlos.



Categorías de eventos

Categorizar los diferentes eventos ayuda a ordenar la forma en la que recordaremos su uso. Esta es una manera óptima de ordenarlos:



Mouse

Los eventos del *mouse* están asociados a **todo aquel evento disparado por el puntero**, e incluso por el “*tap*” que se realiza sobre elementos HTML, a través de pantallas táctiles.



Mouse

click

dblclick

mousedown

mouseup

mouseover

mouseout

mousemove

Teclado

Los eventos de teclado están regidos por todo lo que refiere a **la pulsación de teclas**.

En particular, el evento **change** aplica a poder detectar cuando se realizó un cambio en el contenido de algún elemento HTML.



(*) responde a detectar el evento equivalente a ENTER en un *input type* "search", el cual inicia un evento asociado a una búsqueda.

Formulario

Estos eventos engloban la detección de **acciones sobre elementos de un formulario web**.

Se incluye al *tag* `<form>` como también a los `<input type>` que se agreguen.



Navegador

Los eventos del navegador ocurren en su mayoría **sobre el objeto JavaScript window.**

Y otros eventos ocurren sobre el objeto JavaScript **navigator.**

Por ejemplo:

- Los eventos **offline** y **online** ocurren sobre el navegador web (**navigator**), quien dialoga con el S.O. para poder navegar por Internet.
- Mientras que, los eventos **scroll** o **resize**, ocurren en este caso sobre la ventana/pestaña del *browser*, así es como los mismos se interceptan con el objeto **window**.



Manejo de eventos

Manejo de eventos

Aquí tenemos un resumen de las diferentes formas que podemos manejar eventos dentro de una aplicación JavaScript:

1

atributos on

2

eventos on

3

Event Listener

atributos *on*

Forman parte de los **elementos HTML**. Se conocen con este nombre porque, **en el nombre de cada evento, se antepone el término *on***.

Se definen dentro del elemento HTML, y se les asocia una función que se ejecuta ante la ocurrencia de ese evento.

```
atributos on

//JavaScript
function saludar() {
  console.log("Hola mundo!");
}

//HTML
<button class="btn btn-blue white-text" onclick="saludar();">MENSAJE</button>
```

eventos *on*

Los **eventos on** son la evolución de los **atributos on**.

Se manejan íntegramente desde JS. Debemos enlazarlos con un elemento HTML desde JS, y luego utilizar el evento **on**, al que le definimos una **función como referencia** o declaramos una **función anónima**.



```
//HTML
<button class="btn btn-blue white-text">MENSAJE</button>

//JavaScript
const button = document.querySelector("button");

button.onClick = saludar; //nombre de la función a ejecutar
```

```
//HTML
<button class="btn btn-blue white-text">MENSAJE</button>

//JavaScript
const button = document.querySelector("button");

button.onClick = function() {
  console.log("Hola mundo!");
}
```

addEventListener

Los *event listener* se conforman mediante el uso de un método llamado **addEventListener()**, y espera **dos parámetros**:

1. El nombre del evento.
2. La función o código a ejecutar.

Si se implementan los *event listeners*, al referenciar cada evento, debemos obviar **anteponer en ellos la palabra on**.

```
Event Listener

//HTML
<button class="btn btn-blue white-text">MENSAJE</button>

//JavaScript
const button = document.querySelector("button");

button.addEventListener("click", saludar);
```



¿Cuál implementar?

Sí utilizar

- En la actualidad, **la forma más efectiva y amplia en cuanto a capacidades de controlar eventos**, es con el método **`addEventListener()`**. Además de poder referenciar eventos, se pueden cambiar o quitar mediante otros métodos complementarios.
- También sigue siendo válida la aplicación de **eventos on**.



No utilizar

- Los **atributos on** siguen existiendo en JavaScript por retrocompatibilidad pero, desde que JS comenzó a interactuar con el DOM HTML (en 1999), se recomienda **desestimar su uso** para no agregar código JS en un documento HTML.



Funciones anónimas

Funciones anónimas

Una *función anónima* es una forma diferente de escribir una función. Se puede **declarar una constante y definir que ésta sea igual a una función anónima**, o se puede **asignar una función anónima a un evento**, sin tener que definir un nombre.

```
const miFuncion = function() {  
  console.warn("Estoy ejecutando una función anónima");  
}  
  
miFuncion(); //llamamos a estas igual q a una función normal
```

```
buttonEnviar.onClick = function() {  
  console.log("Código para enviar el contenido");  
}
```



Ejemplos

- Aquí vemos asociada una función anónima al evento **click** de un botón.
- De igual forma lo podemos hacer sobre el método **.addEventListener**.

```
const botonEnviar = document.querySelector("button btn-enviar");

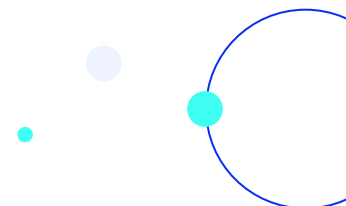
botonEnviar.onClick = function() {
  console.log("Código para enviar el contenido");
}
```

```
const botonEnviar = document.querySelector("button btn-enviar");

botonEnviar.addEventListener("click", function() {
  console.log("Código para enviar el contenido");
});
```

Si bien, en ambos casos, se llega al mismo resultado, lo recomendable de las funciones anónimas es aplicarlas como **callback** sobre un evento específico, en lugar de definirle un nombre.

```
function hacerAlgo() {  
    console.log("Función convencional para hacer algo.");  
}  
  
const hacerAlgo = function() {  
    console.log("Función anónima para hacer algo.");  
}
```




Asignar múltiples eventos

Asignar múltiples eventos

Cuando tenemos un **listado de elementos que repiten su estructura**, como ser un listado de productos con un botón de acción para cada uno de ellos, tenemos que pensar en la **asignación de eventos múltiples a partir de un mismo código**.

Además, es probable que el listado con elementos sea dinámico, por lo tanto, es imposible que desarrollemos un evento dedicado para cada uno de los elementos en cuestión.



The screenshot shows a web browser window titled "Carrito de Compras" with the URL "127.0.0.1:5501/index.html". The page displays a list of products under the heading "Productos". The table has four columns: "Código", "Producto", "Precio", and "Acción". Each row represents a product with a unique code, name, price, and a purple button with a white plus sign for the action.

Código	Producto	Precio	Acción
123	Notebook EXO E17	\$ 59799	+
456	Macbook Air 13 M2	\$ 279799	+
789	Macbook Pro 14	\$ 459799	+
890	iPad Air 10,1	\$ 259799	+
901	Asus Gamer D17	\$ 579799	+
567	Teclado inalámbrico Aluminum	\$ 89759	+
234	Mouse BT - Milky-White	\$ 39855	+

Pasos a seguir con el método *getElementsByClassName*

1. El método **getElementsByClassName** permitirá enlazar con todos estos elementos **button**, y crear una colección de elementos HTML.

```
<button class="button btn-agregar">+</button>

const botonesAregar = document.getElementsByClassName("btn-agregar");
//obtenemos una colección de elementos HTML del btipo button
```



2. Luego, podremos recorrer esa colección utilizando el ciclo **for...of** y aplicar en cada iteración, el evento **click** sobre el botón.

```
● ● ●  
  
<button class="button btn-agregar">+</button>  
  
const botonesAgregar = document.getElementsByClassName("btn-agregar");  
//obtenemos una colección de elementos HTML del btipo button
```

3. Dado que cada botón de dicha colección requiere también **tener un identificador unívoco**, podemos aprovechar el **atributo ID** de estos para asignarle, por ejemplo, el código unívoco de cada producto del listado.



```
const botonesAgrega = document.getElementsByClassName("btn-agregar");

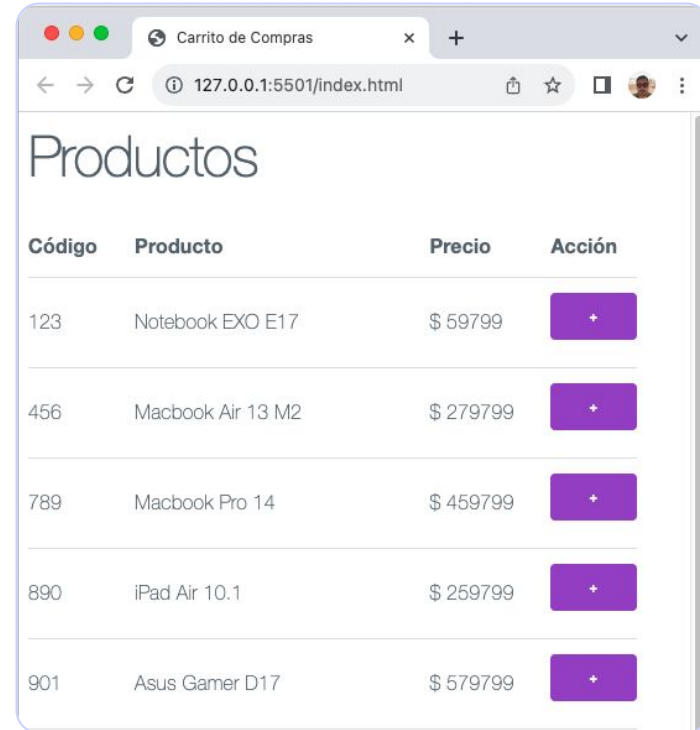
for (boton of botonesAgrega) {
  boton.onClick = funcion() {
    console.log("Controlamos el evento click en el botón");
  }
}
```


Aprovechando que **cada producto tiene un código unívoco**, y que seguramente este listado de productos se genere de forma dinámica, podremos aprovechar dicho código para **asignarlo como ID a cada botón**.

```
<button id="{producto.id}" class="button btn-agregar">+</button>
```



De esta forma, cuando se pulse el botón, se podrá leer desde JS su propiedad *ID*, y determinar qué producto está seleccionando el usuario, al pulsar el botón.

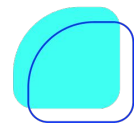


The screenshot shows a web browser window with the title "Carrito de Compras" and the URL "127.0.0.1:5501/index.html". The page content is titled "Productos" and displays a table with the following data:

Código	Producto	Precio	Acción
123	Notebook EXO E17	\$ 59799	<input type="button" value="+"/>
456	Macbook Air 13 M2	\$ 279799	<input type="button" value="+"/>
789	Macbook Pro 14	\$ 459799	<input type="button" value="+"/>
890	iPad Air 10.1	\$ 259799	<input type="button" value="+"/>
901	Asus Gamer D17	\$ 579799	<input type="button" value="+"/>

```
for (boton of botonesAgregar) {  
  boton.onClick = function() {  
    console.log("Pulsaste el botón del Producto:", boton.id);  
  }  
}
```

Así, se controlan con precisión los eventos múltiples, al momento de generarlos. Además, con un código simple, se podrán asignar eventos a uno, dos, diez, cien o mil botones, con tan solo unas pocas líneas de código.



Pasos a seguir con el método `querySelectorAll`

1. El método `querySelectorAll` es también netamente funcional para llegar al mismo resultado de iteración y generación de múltiples eventos desde un mismo código.

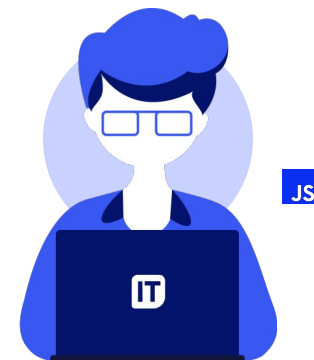


```
const botonesAgregar = document.querySelectorAll("button.btn-agregar");  
  
for (boton of botonesAgregar) {  
  ...  
}
```



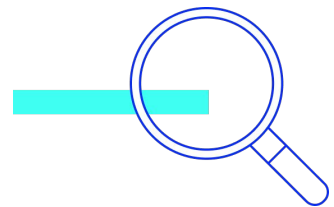
2. De igual forma, el utilizar el método `addEventListener`.

```
for (boton of botonesAgregar) {  
  boton.addEventListener("click", function() {  
    console.log("Pulsaste el botón del Producto:", boton.id);  
  });  
}
```



Revisión

- Repasar el concepto de **evento**.
- Investigar todos los **eventos posibles**.
- Combinar **funciones con eventos**.
- Integrar un **bucle para reutilizar una función** desde el código.
- Trabajar en el ***Proyecto integrador***.
- Realizar las preguntas necesarias a la/el docente antes de continuar.



¡Sigamos
trabajando!