

Recuerden poner a
grabar la clase



Clase 11

Eventos II

Diplomatura UNTREF



Temario

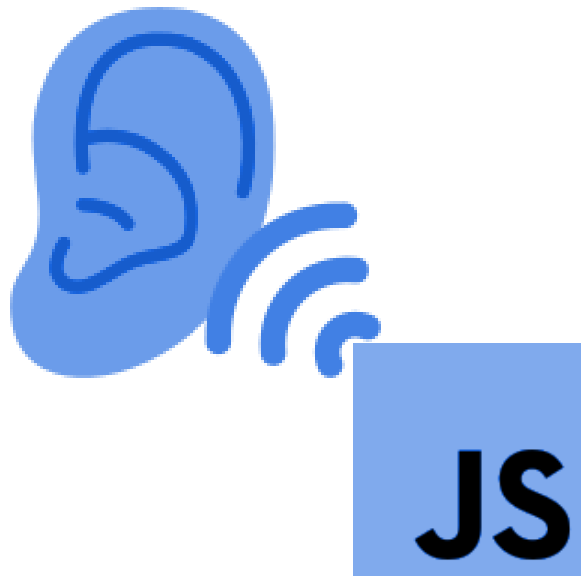
EVENTOS II:

- Metodo addEventListener
- Concepto de callback
- eventos (teclado-formulario-navegador)
- información de un evento (object global Event)

Metodo `addEventListener()`

Método `addEventListener()`

En JavaScript, `addEventListener()` es una función que permite a los desarrolladores asignar uno o más "escuchadores" (también conocidos como "handlers" o "manejadores") a un elemento HTML específico, de manera que se pueda detectar y responder a eventos que ocurran en ese elemento.



Método addEventListener()

La sintaxis de el metodo addEventListener(), consta de lo siguiente.

- **element**: Es el elemento HTML al que deseas añadir el manejador de eventos.
- **eventType**: Es una cadena que especifica el tipo de evento al que deseas responder, como "click", "mouseover", "keydown", etc.

```
element.addEventListener(eventType, handlerFunction, useCapture);
```

Método `addEventListener()`

- **handlerFunction:** Es una función que se ejecutará cuando ocurra el evento. Puede ser una función anónima o una referencia a una función existente.
- **useCapture (opcional):** Es un booleano que indica si el evento debe ser capturado durante la fase de captura (`true`) o en la fase de burbujeo (`false`). Generalmente, se utiliza `false`.

```
element.addEventListener(eventType, handlerFunction, useCapture);
```

Método addEventListener() - ejemplo

Cuando el botón con el id "miBoton" sea clickeado, se ejecutará la función anónima que muestra una alerta en el navegador.

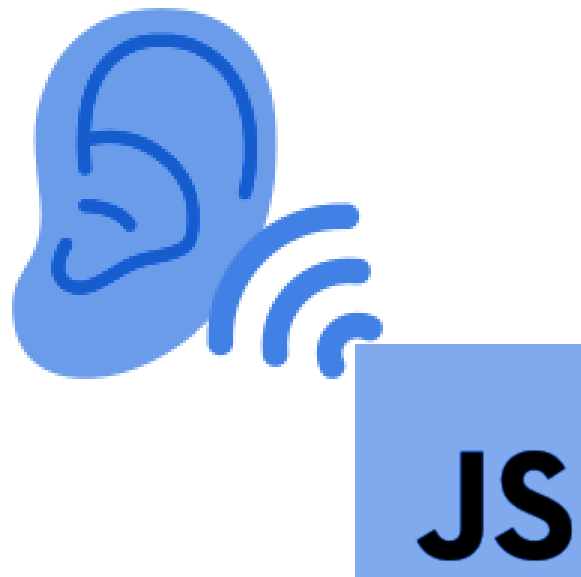
Un beneficio importante de usar addEventListener() es que puedes agregar varios manejadores de eventos para el mismo tipo de evento en un mismo elemento. Esto permite modularizar el código y separar las responsabilidades.

```
// Obtener el elemento del botón
let boton = document.getElementById("miBoton");

// Agregar un manejador de eventos para el clic
boton.addEventListener("click", function() {
    alert("¡El botón fue clickeado!");
});
```


Método `addEventListener()` - consideraciones

Además, este método también es útil para evitar la sobrescritura accidental de los manejadores de eventos. Si necesitas modificar el comportamiento de un evento en el futuro, puedes simplemente agregar o eliminar manejadores sin preocuparte de interferir con otros.

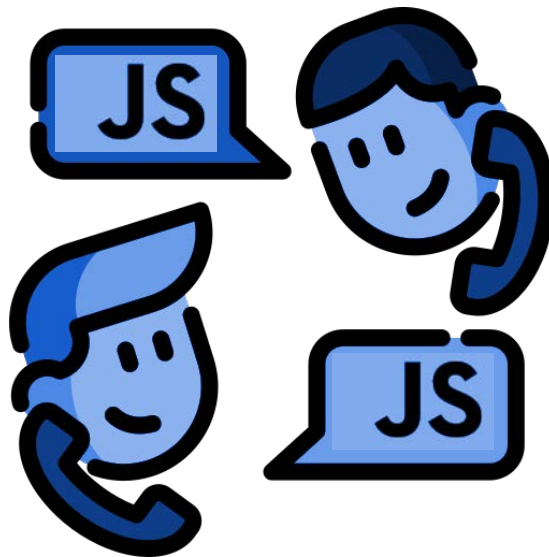


Concepto Call Back

¿Qué es un callback?

Un **callback** en JavaScript es una función que se pasa como argumento a otra función y que se ejecuta después de que esa función haya completado su tarea.

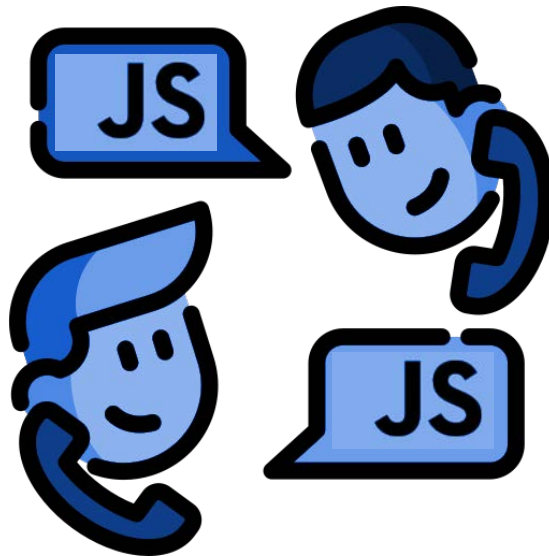
En esencia, un **callback** es una manera de decirle a una función que haga algo específico después de que otra función haya terminado su ejecución.



¿Qué es un callback?

Los **callbacks** son especialmente útiles en situaciones donde tienes operaciones asincrónicas, como solicitudes de red, lecturas de archivos o cualquier tarea que tome tiempo en completarse.

En lugar de bloquear el flujo de ejecución mientras esperas a que estas operaciones finalicen, puedes proporcionar una función de callback que se ejecutará una vez que la operación asincrónica termine.



callback en acción

En este ejemplo, la función `hacerAlgo()` simula una operación asíncronica utilizando `setTimeout()`.

Una vez que la tarea simulada ha finalizado (después de 2 segundos), se llama al callback `completado()`.

```
function hacerAlgo(callback) {  
  console.log("Haciendo algo...");  
  setTimeout(function() {  
    console.log("Tarea finalizada");  
    callback();  
  }, 2000);  
}  
  
function completado() {  
  console.log("La tarea ha sido completada.");  
}  
  
hacerAlgo(completado);
```

¿Qué es un callback?

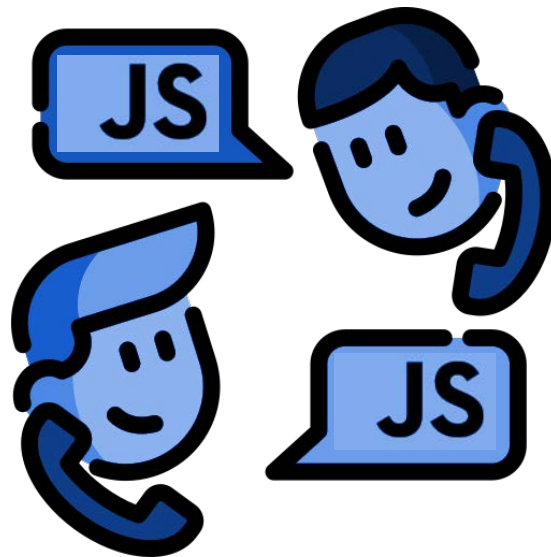
Los **callbacks** son fundamentales en JavaScript, especialmente en el contexto de operaciones asincrónicas, como solicitudes **AJAX**, **lectura de archivos**, interacciones con **bases de datos y eventos** del usuario.

Sin embargo, a medida que el código crece, el uso excesivo de **callbacks** puede llevar a un patrón conocido como "**callback hell**" (infierno de callbacks), donde múltiples niveles anidados de callbacks pueden dificultar la comprensión y el mantenimiento del código.

```
leerArchivo(function(data) {
  procesarDatos(data, function(result) {
    enviarDatos(result, function(response) {
      // Y así sucesivamente...
    });
  });
});
```

¿Qué es un callBack?

Para abordar este problema, han surgido patrones y enfoques alternativos, como las **Promesas** y las **async/await** (introducidas en ECMAScript 2017), que proporcionan una forma más estructurada y legible de manejar operaciones asincrónicas en comparación con los callbacks anidados.

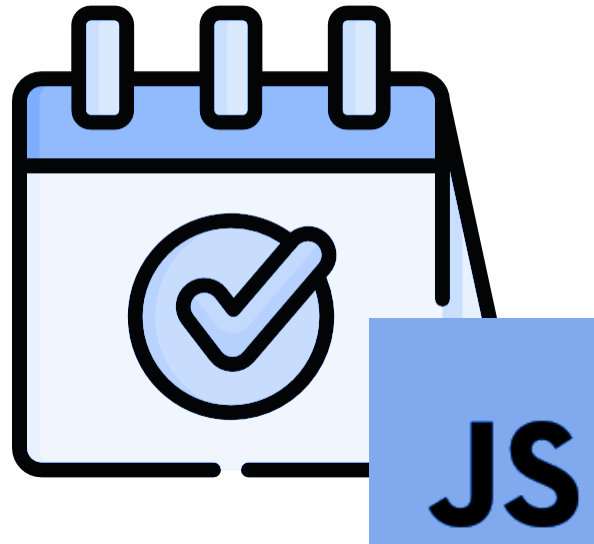


Eventos (teclado, formularios,
navegador)

Eventos, teclados - formularios - navegador

Los **eventos** en JavaScript son mecanismos que permiten a tu código responder a las interacciones del usuario o a otros cambios en la página web.

Los **eventos** pueden ser generados por el navegador, por el usuario (como hacer clic en un botón), por el teclado, o por otras fuentes.



Eventos - teclado

Eventos de teclado

Los eventos de teclado se activan cuando el usuario interactúa con el teclado. Algunos de los eventos de teclado más comunes son:




Eventos de teclado - keydown

El **evento keydown** se dispara cuando una tecla es presionada.

Esto ocurre en el momento exacto en que el usuario presiona la tecla, antes de que sea liberada.

Durante el evento keydown, puedes acceder a la propiedad `event.key` para obtener el valor de la tecla que fue presionada.

Este evento es útil para crear interacciones que respondan inmediatamente cuando una tecla es presionada, como juegos o aplicaciones que requieren una respuesta rápida.



```
document.addEventListener("keydown",  
function(event) {  
    console.log("Tecla presionada:", event.key);  
});
```

Eventos de teclado - keyup

El **evento keyup** se dispara cuando una tecla es liberada.

Esto ocurre en el momento en que el usuario suelta la tecla que había presionado previamente.

Al igual que con **keydown**, puedes usar `event.key` para acceder al valor de la tecla liberada.

Este evento es especialmente útil para detectar cuándo el usuario ha dejado de presionar una tecla, lo que puede ser utilizado para activar ciertas acciones después de que una tecla sea liberada.

```
document.addEventListener("keyup",  
function(event) {  
    console.log("Tecla liberada:", event.key);  
});
```

Eventos de teclado - keypress


El evento `keypress` se dispara cuando una tecla es presionada y luego liberada.

Sin embargo, este evento se utiliza principalmente para teclas que generan caracteres imprimibles en la pantalla (letras, números, símbolos, etc.).

No se dispara para teclas como **Shift**, **Ctrl**, **Alt**, etc.

Dado que **keypress** se enfoca en caracteres imprimibles, es menos común de usar en comparación con **keydown** y **keyup**.

En la mayoría de los casos, **keydown** y **keyup** son más adecuados para capturar la interacción del usuario con el teclado.



```
document.addEventListener("keypress",  
function(event) {  
    console.log("Tecla presionada y liberada:",  
event.key);  
});
```

Eventos - formulario

Eventos de teclado - formulario

Los **eventos de formulario** son utilizados para rastrear la interacción del usuario con elementos de formulario, como campos de texto, casillas de verificación y botones.



Eventos de teclado - submit

El **evento submit** se dispara cuando el usuario envía un formulario.

Por lo general, esto ocurre cuando el usuario hace clic en un botón "Enviar" dentro de un formulario o presiona la tecla Enter mientras está dentro de un campo de texto.

Este evento es útil para realizar validaciones y procesamiento antes de que los datos del formulario sean enviados al servidor.

```
const formulario =  
document.getElementById("miFormulario");  
  
formulario.addEventListener("submit",  
function(event) {  
    event.preventDefault(); // Evita el  
    comportamiento por defecto de enviar el  
    formulario  
  
    // Realizar validaciones y procesamiento  
    console.log("Formulario enviado");  
});
```

Eventos de teclado - input

El **evento input** se dispara cuando el contenido de un campo de entrada cambia.

Esto puede ocurrir cuando el usuario escribe, pega o elimina contenido en un campo de texto, por ejemplo.

Es un evento asincrónico que se dispara cada vez que el contenido del campo cambia, incluso si el cambio es causado por el teclado, el ratón o cualquier otra forma de entrada.

```
const inputElement =  
document.getElementById("miInput");  
  
inputElement.addEventListener("input",  
function(event) {  
    const inputValue = event.target.value;  
    console.log("Contenido del campo de entrada:",  
inputValue);  
});
```

Eventos de teclado - change

El **evento change** se dispara cuando el contenido de un campo de entrada cambia y el campo pierde el enfoque.

En otras palabras, se activa cuando el usuario modifica el contenido de un campo y luego hace clic fuera de ese campo.

Este evento es comúnmente utilizado en elementos como selectores de opciones (`<select>`) y casillas de verificación (`<input type="checkbox">`).

```
const selectElement =  
document.getElementById("miSelect");  
  
selectElement.addEventListener("change",  
function(event) {  
    const selectedOption = event.target.value;  
    console.log("Opción seleccionada:",  
selectedOption);  
});
```

Eventos - navegador

Eventos de Navegador

Los eventos de navegador son eventos generales que ocurren en el navegador y no están específicamente relacionados con elementos individuales en la página.



Eventos de Navegador

Estos eventos te permiten controlar aspectos generales de la interacción entre el usuario y la página.

Algunos ejemplos de eventos de navegador son:

- **load:** Se dispara cuando la página web y sus recursos han terminado de cargarse.
- **unload:** Se dispara cuando la página está a punto de descargarse (no es muy común).
- **resize:** Se dispara cuando la ventana del navegador cambia de tamaño.
- **scroll:** Se dispara cuando el usuario se desplaza por la página.

```
window.addEventListener("resize", function() {  
    // Ajustar el diseño de la página en función  
    del tamaño de la ventana  
});
```

Eventos de Navegador - load

El **evento load** se dispara cuando la página web y todos sus recursos (imágenes, scripts, hojas de estilo, etc.) han terminado de cargarse completamente.

Es útil para ejecutar scripts o realizar acciones que deben llevarse a cabo después de que todos los elementos de la página se hayan cargado.

```
window.addEventListener("load", function() {  
    console.log("La página y sus recursos han  
cargado completamente.");  
});
```

Eventos de Navegador - unload

El **evento unload** se dispara cuando la página está a punto de ser descargada, ya sea porque el usuario está navegando a otra página o cerrando la ventana del navegador.

Sin embargo, este evento no es muy común de usar, ya que la mayoría de las acciones que desees realizar en este momento pueden ser problemáticas o inciertas.

```
window.addEventListener("unload", function() {  
    console.log("La página está a punto de ser  
descargada.");  
});
```


Eventos de Navegador - resize

El **evento resize** se dispara cuando el usuario cambia el tamaño de la ventana del navegador, ya sea mediante un ajuste manual o mediante el cambio de orientación en dispositivos móviles.

Este evento es útil para adaptar el diseño de tu página a diferentes tamaños de pantalla.

```
window.addEventListener("resize", function() {  
    console.log("La ventana del navegador cambió de  
    tamaño.");  
});
```

Eventos de Navegador - scroll

El **evento scroll** se dispara cuando el usuario se desplaza por la página, ya sea vertical u horizontalmente.

Puedes usar este evento para crear efectos de paralaje, cargar contenido adicional cuando el usuario llega a cierta posición o realizar otras acciones relacionadas con el desplazamiento.

```
window.addEventListener("scroll", function() {  
    console.log("El usuario se está desplazando por  
    la página.");  
});
```

Información de un evento (object global Event)

object global event - event.type

Una cadena que indica el tipo de evento que ha ocurrido, como "click", "keydown", "submit", etc.

```
document.getElementById("miBoton").addEventListener("click",  
function(event) {  
    console.log("Tipo de evento:", event.type); // Imprimirá "click"  
});
```

object global event - event.target

El elemento HTML que originó el evento.

Por ejemplo, si se hace clic en un botón, event.target sería el botón.

```
...  
<button id="miBoton">Haz clic en mí</button>  
  
document.getElementById("miBoton").addEventListener("click",  
function(event) {  
    console.log("Objetivo del evento:", event.target); // Imprimirá el  
    botón  
});
```

object global event - event.preventDefault

Un método que detiene el comportamiento por defecto del evento.

Por ejemplo, en el caso de un enlace, esto evitará que el navegador navegue a la URL del enlace.

```
<a href="https://www.ejemplo.com" id="miEnlace">Enlace</a>

//javascript

document.getElementById("miEnlace").addEventListener("click",
function(event) {
    event.preventDefault(); // Evita que el navegador navegue a la URL
del enlace
    console.log("Enlace fue clickeado, pero no se redireccionará.");
});
```

object global event - event.stopPropagation()

Un método que detiene la propagación del evento a través del modelo de burbujeo o captura. Evita que el evento se propague a elementos padre o hijos.

```
<div id="padre">
  <button id="hijo">Botón hijo</button>
</div>

//javascript
document.getElementById("hijo").addEventListener("click",
function(event) {
  console.log("Clic en el botón hijo.");
  event.stopPropagation(); // Detiene la propagación del evento al
  elemento padre
});

document.getElementById("padre").addEventListener("click",
function(event) {
  console.log("Clic en el elemento padre.");
});
```

object global event - event.key

En eventos de teclado, proporciona el valor de la tecla que fue presionada.

```
document.addEventListener("keydown", function(event) {  
    console.log("Tecla presionada:", event.key); // Imprimirá el valor de  
    la tecla  
});
```


object global event - event.clientX event.clientY

En eventos de ratón, proporcionan las coordenadas horizontales y verticales del puntero del ratón en relación con la ventana del navegador.

¿Dudas o consultas?





¡Muchas Gracias!