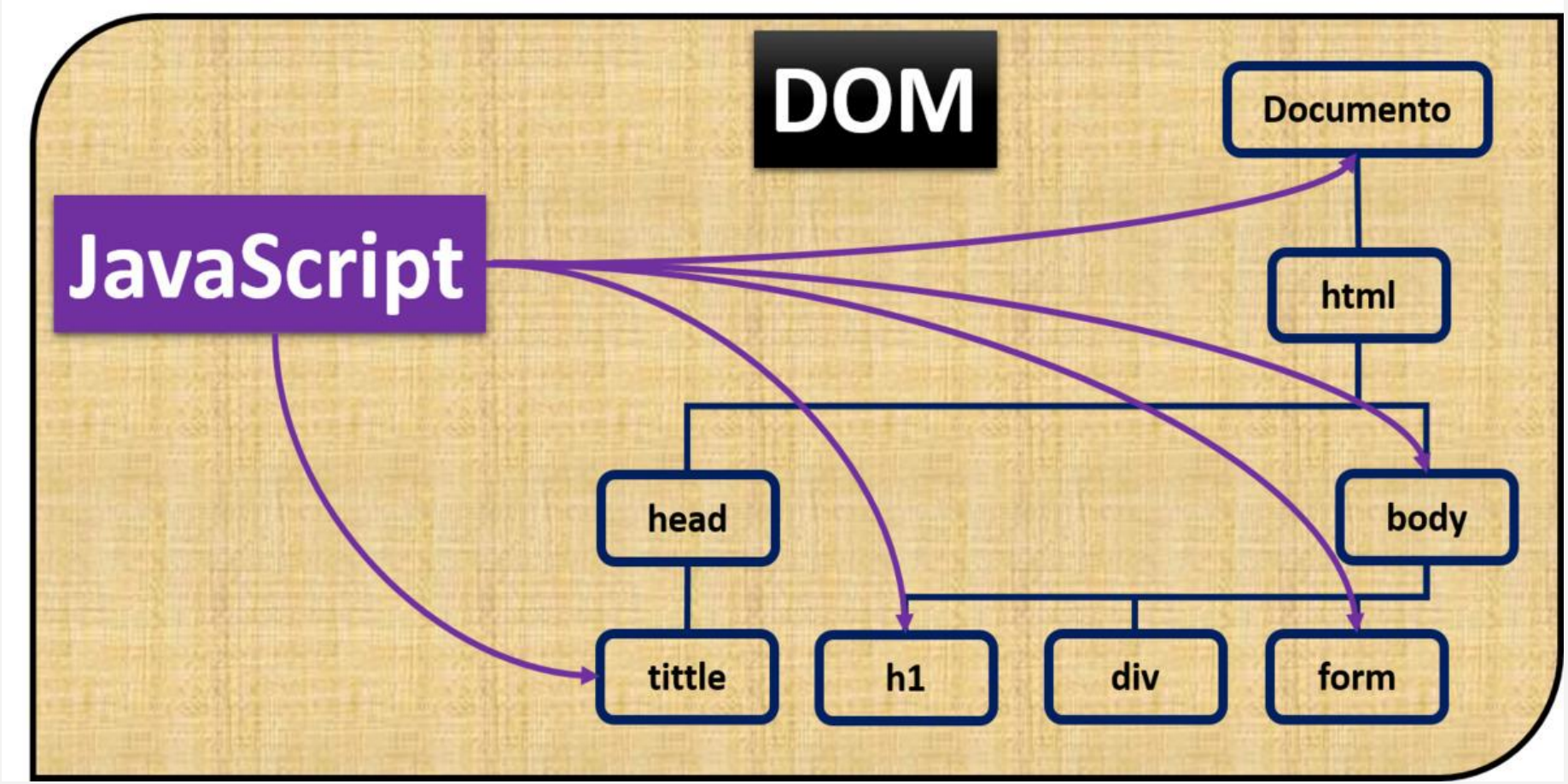
UNAHUR
UNIVERSIDAD NACIONAL
DE HURLINGHAM

Manipulación del DOM

CÁTEDRA: CONSTRUCCIÓN DE INTERFACES DE USUARIO

Manipulación del DOM



Manipulación del DOM

La **manipulación del DOM** (Document Object Model) en JavaScript se refiere al proceso de **acceder y modificar los elementos de una página web** (como párrafos, botones, imágenes, etc.) a través de código JavaScript.

La **manipulación del DOM** es una parte fundamental de JavaScript cuando trabajás con páginas web. Te permite hacer que el sitio **responda al usuario** sin tener que recargar la página.

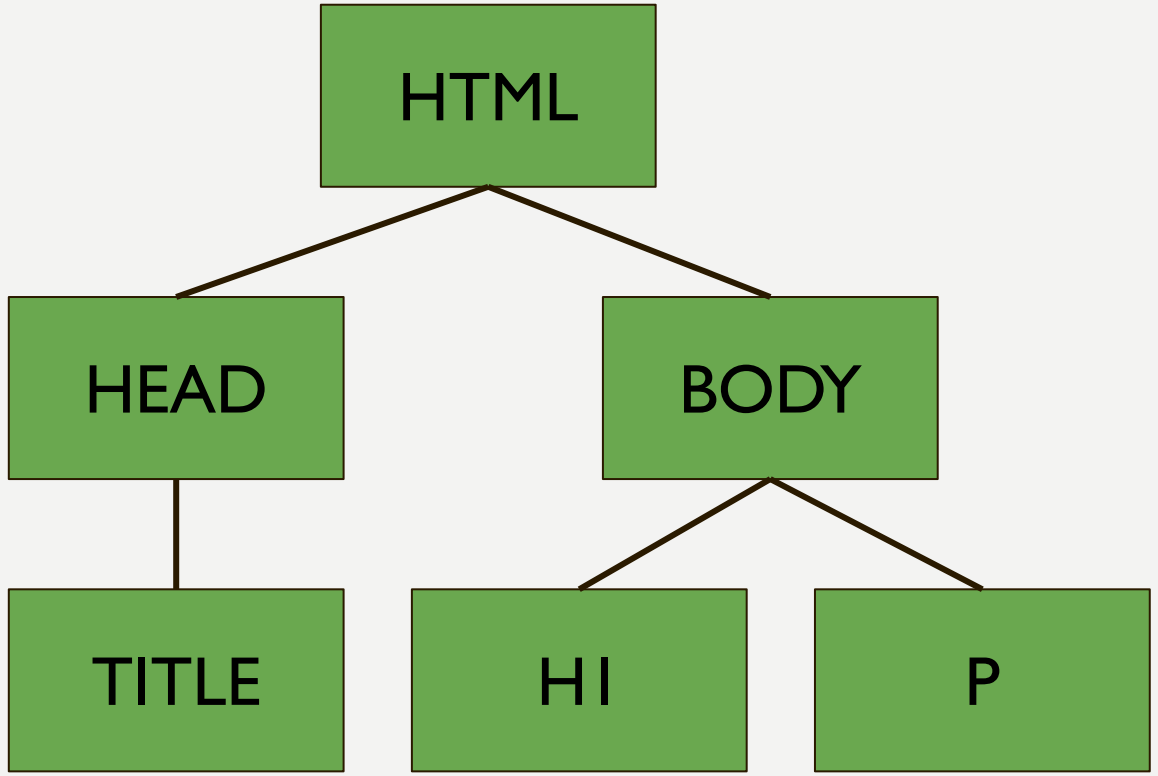
¿Qué es el DOM?

DOM es una interfaz de programación que nos permite crear, cambiar, o remover elementos del documento HTML.

Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina **árbol DOM**.

Ejemplo

```
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de DOM</title>
</head>
<body>
    <h1>Hola Mundo</h1>
    <p>Este es un párrafo.</p>
</body>
</html>
```



DOM (Document Object Model)

Con el DOM, se puede **acceder y manipular cualquier parte del documento** mediante JavaScript. Esto incluye cambiar el contenido, los estilos, los atributos, y también agregar o eliminar elementos.

El DOM también permite gestionar eventos que ocurren en el documento, como clics, movimientos del ratón, teclas presionadas, etc. Estos eventos pueden ser manejados con funciones en JavaScript.

DOM (Document Object Model)

- El DOM es una representación en forma de árbol de un documento HTML.
- Cada elemento en HTML es un nodo en el árbol del DOM.
- JavaScript puede acceder y manipular el DOM para cambiar el contenido y la estructura de una página web.

¿Qué permite hacer JavaScript con el DOM?

- **Seleccionar elementos** del HTML (getElementById, querySelector, etc.).
- **Modificar contenido o atributos** (element.textContent, element.setAttribute, etc.).
- **Cambiar estilos** (element.style).
- **Crear o eliminar elementos** (createElement, appendChild, remove).
- **Responder a eventos** como clics o teclas (addEventListener).

Selección de Elementos del DOM

Métodos de Selección:

- document.getElementById()
- document.getElementsByClassName()
- document.getElementsByTagName()
- document.querySelector()
- document.querySelectorAll()

¿Que es document?

En JavaScript, document es un objeto que representa el **documento HTML cargado en el navegador.** Es parte del **DOM** (Document Object Model).

Con document, podés acceder y modificar elementos del HTML, agregar contenido, cambiar estilos, manejar eventos, y mucho más.

Métodos de Selección

document.getElementById()

Descripción: Selecciona un elemento del DOM por su atributo id.

Uso: Devuelve el primer elemento con el id especificado.

Ejemplo:

```
const element = document.getElementById('miElemento');
```

Métodos de Selección

document.getElementsByClassName()

Descripción: Selecciona todos los elementos que tienen una clase específica.

Uso: Devuelve una colección en vivo (HTMLCollection) de todos los elementos que tienen todas las clases especificadas.

Ejemplo:

```
const elements = document.getElementsByClassName('miClase');
```

Convertir el HTMLCollection a un array

Aunque HTMLCollection parece un array, no lo es.

Puedes hacer esto usando el método **Array.from()** para convertir el HTMLCollection a un array.

Ejemplo:

```
// Seleccionamos todos los elementos con la clase "mi-clase"
let elementos = document.getElementsByClassName('mi-clase')
// Convertimos el HTMLCollection a un array
let elementosArray = Array.from(elementos)
// Aplicamos el método map para obtener los textos de cada elemento
let textos = elementosArray.map(elemento => elemento.textContent)
```

Métodos de Selección

document.getElementsByTagName()

Descripción: Selecciona todos los elementos con un nombre de etiqueta específico.

Uso: Devuelve una colección en vivo de todos los elementos con el nombre de etiqueta dado.

Ejemplo:

```
const elements = document.getElementsByTagName('div');
```

Métodos de Selección

document.querySelector()

Descripción: Selecciona el primer elemento que coincide con un selector CSS especificado.

Uso: Devuelve el primer elemento en el documento que coincide con el grupo de selectores especificado.

Ejemplo:

```
const element = document.querySelector('.miClase #miElemento');
```

Métodos de Selección

document.querySelectorAll()

Descripción: Selecciona todos los elementos que coinciden con un selector CSS especificado.

Uso: Devuelve una NodeList estática de todos los elementos en el documento que coinciden con el grupo de selectores especificado.

Ejemplo:

```
const elements = document.querySelectorAll('div.miClase');
```

Resumen

document.getElementById(): Ideal para seleccionar un único elemento por su id.

document.getElementsByClassName(): Útil para seleccionar múltiples elementos que comparten la misma clase.

document.getElementsByTagName(): Utilizado para seleccionar todos los elementos con un nombre de etiqueta específico.

document.querySelector(): Perfecto para seleccionar el primer elemento que coincide con un selector CSS complejo.

document.querySelectorAll(): Utilizado para seleccionar todos los elementos que coinciden con un selector CSS complejo, devolviendo una NodeList estática.

Código de Ejemplo:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Manipulación del DOM</title>
</head>
<body>
    <h1 id="titulo">Hola Mundo</h1>
    <p class="parrafo">Este es un párrafo.</p>
    <button id="boton">Haz clic aquí</button>
    <script src="app.js"></script>
</body>
</html>
```

Código de Ejemplo:

```
// Selección de elementos
const titulo = document.getElementById('titulo');
const parrafos = document.getElementsByClassName('parrafo');
const boton = document.querySelector('#boton');
```

Principales Tipos de Manipulación del DOM

Modificación de Contenido:

Cambiar el texto, HTML o atributos de un elemento.

Ejemplo:

```
// Cambiar el texto de un elemento
const element = document.getElementById('miElemento');
element.textContent = 'Nuevo Texto';

element.innerHTML = '<span>Nuevo Texto</span>';
```

textContent

La propiedad **textContent** se utiliza para obtener o establecer el contenido textual de un elemento. Esto incluye solo el texto dentro del elemento y sus hijos, excluyendo cualquier etiqueta HTML.

Permite insertar solo texto, sin interpretar ninguna estructura HTML.

Ejemplo:

```
// Obtener el contenido de texto de un elemento
let textContent = document.getElementById("miElemento").textContent;

// Establecer el contenido de texto de un elemento
document.getElementById("miElemento").textContent = "Nuevo contenido de texto";
```

innerHTML

La propiedad **innerHTML** se utiliza para obtener o establecer el contenido HTML de un elemento. Esto incluye cualquier elemento HTML y su contenido.

Permite insertar HTML, incluyendo etiquetas HTML y su estructura, dentro de un elemento.

Ejemplo:

```
// Obtener el contenido HTML de un elemento
let htmlContent = document.getElementById("miElemento").innerHTML;

// Establecer el contenido HTML de un elemento
document.getElementById("miElemento").innerHTML = "<p>Nuevo contenido <strong>HTML</strong></p>";
```

Principales Tipos de Manipulación del DOM

Cambio de Estilos:

Modificar los estilos CSS de un elemento.

Ejemplo:

```
// Cambiar el color del texto
element.style.color = 'red';
```

Principales Tipos de Manipulación del DOM

Gestión de Clases:

Añadir, quitar o alternar clases CSS de un elemento.

Ejemplo:

```
// Añadir una clase CSS
element.classList.add('miClase');

// Quitar una clase CSS
element.classList.remove('miClase');
```

Principales Tipos de Manipulación del DOM

Manipulación de Atributos:

Modificar los atributos de un elemento.

Ejemplo:

```
// Establecer un atributo
element.setAttribute('title', 'Este es un título');

// Obtener un atributo
const title = element.getAttribute('title');
```

Principales Tipos de Manipulación del DOM

Creación y Eliminación de Elementos:
Crear nuevos elementos y añadirlos al documento, o eliminar elementos existentes.

Ejemplo:

```
// Crear un nuevo párrafo
const nuevoParrafo = document.createElement('p');
nuevoParrafo.textContent = 'Este es un nuevo párrafo.';
document.body.appendChild(nuevoParrafo);

// Eliminar un elemento
const eliminarElemento = document.getElementById('elementoAEliminar');
eliminarElemento.remove();
```

¿Por qué es importante?

La **manipulación del DOM** te permite:

- Hacer que la página **reaccione en tiempo real.**
- Crear **interfaces interactivas** (formularios, menús, sliders, etc.).
- Evitar recargas de página innecesarias.
- Crear una mejor experiencia de usuario.

Beneficios de la Manipulación del DOM

- **Interactividad:** Permite crear interfaces de usuario dinámicas que pueden responder a las acciones del usuario en tiempo real.
- **Actualizaciones Dinámicas:** Posibilita la actualización de contenido sin necesidad de recargar la página completa.
- **Experiencia del Usuario:** Mejora la experiencia del usuario al permitir interfaces más fluidas y reactivas.

Ejercicio

En un archivo HTML se encuentra una lista de productos con sus precios representada con una lista no ordenada (<ul>), donde cada ítem (<li>) incluye el nombre del producto y su precio actual.

Tu tarea es escribir un script en JavaScript que:

1. Reciba un array de objetos llamado nuevosPrecios, donde cada objeto tiene dos propiedades: producto (nombre del producto) y precio (nuevo precio).
2. Busque en el DOM cada ítem de la lista que coincida con el nombre del producto.
3. Reemplace el precio mostrado en la página por el nuevo precio que corresponde a ese producto.

Ejercicio

HTML:

```
<ul id="lista-productos">
    <li>Pan - $100</li>
    <li>Leche - $200</li>
    <li>Queso - $300</li>
</ul>
```

Javascript:

```
const nuevosPrecios = [
    { producto: "Pan", precio: 120 },
    { producto: "Leche", precio: 220 },
    { producto: "Queso", precio: 330 }
];
```

Ejercicio

No se debe modificar el HTML manualmente: todos los cambios deben realizarse mediante JavaScript.

Ayuda

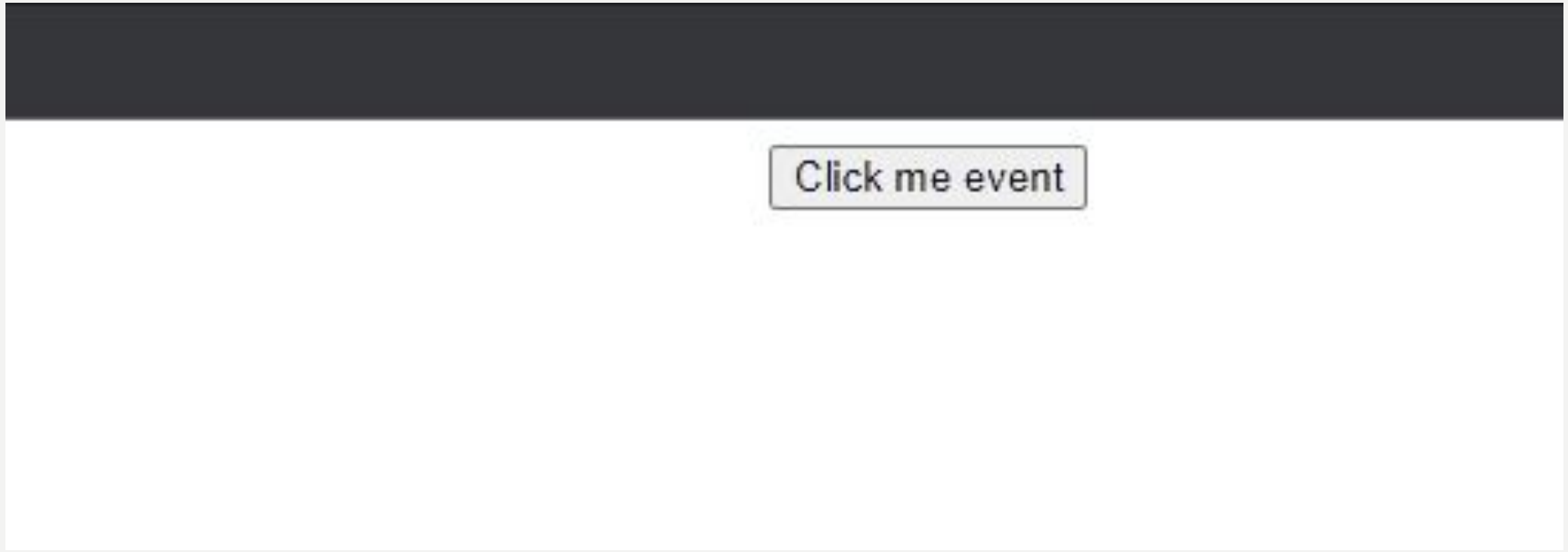
El método .split() en JavaScript se utiliza para dividir una cadena de texto en un array de partes, utilizando un separador que vos le indiques.

Ejemplo:

```
const frase = "Hola cómo estás";
const palabras = frase.split(" ");
console.log(palabras); // ["Hola", "cómo", "estás"]
```

Eventos

Eventos



Eventos

Un evento es una **acción o suceso** que ocurre en el navegador, como un clic, la carga de una página, el envío de un formulario, etc.

Estas acciones pueden ser realizadas por el usuario (como hacer clic, mover el mouse, presionar una tecla), por el navegador (como cargar una página) o por el sistema (como un temporizador que se agota).

JavaScript utiliza un modelo de eventos basado en el DOM, donde los eventos pueden ser capturados y manejados en diferentes fases.

Manejar eventos

En JavaScript, **hay tres formas principales de manejar eventos**:

- Mediante atributo HTML (forma antigua pero útil en ejemplos simples)
- Usando propiedades del DOM
- Con addEventListener (la más recomendada)

¿Qué es manejar un evento con atributo HTML?

Significa agregar el **nombre del evento** directamente en la **etiqueta HTML**, seguido de código JavaScript como valor.

Sintaxis general:

<etiqueta evento="código JavaScript">

Por ejemplo:

```
<button onclick="alert('¡Hola!')">Saludar</button>
```

¿Qué significa manejar un evento con propiedad del DOM?

Significa acceder al elemento HTML desde JavaScript y **asignar una función** a una propiedad que representa el evento, como onclick, onchange, oninput, etc.

Sintaxis general:

```
elemento.evento = función;
```

Ejemplo:

```
<button id="btn">Haz clic</button>

<script>
    const btn = document.getElementById('btn');
    btn.onclick = function() {
        alert('¡Clic detectado con propiedad del DOM!');
    };
</script>
```

¿Qué está pasando aquí?

- document.getElementById('btn'): selecciona el botón del DOM.
- btn.onclick = function() { ... }: se asigna una función al evento onclick.
- Cuando el usuario hace clic, se ejecuta el código dentro de la función.

¿Qué es addEventListener()?

Es un método que permite **escuchar un evento** en un elemento HTML y **ejecutar una función cuando ocurre ese evento.**

Sintaxis general:

```
elemento.addEventListener('tipoDeEvento', función);
```

Agregar Manejadores de Eventos

addEventListener(): Método moderno para agregar manejadores de eventos.

Ejemplo:

```
const boton = document.getElementById('miBoton');
boton.addEventListener('click', function() {
    alert('¡Botón clickeado!');
});
```

Formas de definirlos

Con función Anónima y asignando el botón a una variable

```
let boton = document.getElementById('miBoton')
boton.addEventListener('click', function() {
    alert('¡Botón clickeado con variable!');
});
```

Con función Anónima:

```
document.getElementById('miBoton').addEventListener('click', function() {
    alert('¡Botón clickeado!');
});
```

Formas de definirlos

Con función Nombrada:

```
let boton=document.getElementById('miBoton')
boton.addEventListener('click', MostrarMsj);

function MostrarMsj(){
    alert('¡Botón clickeado con función nombrada!');
};
```

Arrow Function:

```
document.getElementById('miBoton').addEventListener('click', () => {
    alert('¡Botón clickeado con arrow function!');
});
```

¿Cuál conviene usar?

Depende de lo que necesito hacer. Si voy a reutilizar me conviene definir la función nombrada.

Ejemplo: tengo dos botones, al hacer click en uno se pinta el fondo de rojo y al hacer click en el otro se pinta el fondo de verde.

```
<body>
    <button id="rojoBtn">Pintar Fondo Rojo</button>
    <button id="verdeBtn">Pintar Fondo Verde</button>
</body>
```

```
function cambiarColor(color) {
    document.body.style.backgroundColor = color;
}
```

¿Cómo le asigno el evento a cada botón reutilizando la función cambiarColor?

¿Cuál conviene usar?

Puedo hacerlo con una función nombrada, ya que indicamos que eso sería lo ideal si necesito reutiliza:

```
rojoBtn.addEventListener("click", cambiarColor("red"));
verdeBtn.addEventListener("click", cambiarColor("green"));
```

¿Pero qué sucede?

La función se ejecuta en el momento que estoy asignando el evento al botón. Eso sucede porque los paréntesis indican que debe ejecutarse. Si revisamos el primer ejemplo que hicimos con funcion nombrada:

```
boton.addEventListener('click', MostrarMsj);
```

Y le agregamos los paréntesis a la función:

```
boton.addEventListener('click', MostrarMsj());
```

También el mensaje se muestra ANTES que apretemos el botón

¿Cómo se soluciona?

Con funciones anónimas o arrow function:

```
rojoBtn.addEventListener("click", () => cambiarColor("red"));
verdeBtn.addEventListener("click", function(){
        cambiarColor("green");
});
```

Si necesito pasar parametros debo utilizar alguna de estas formas, en las cuales indico por referencia cuál es la función que quiero que se ejecute cuando ocurra el evento.

Este caso también paso la referencia a la función que quiero que se ejecute, pero no puedo pasarle parámetros

```
boton.addEventListener('click', MostrarMsj);
```

Tipos de Eventos Comunes - Ratón

Eventos de Ratón:

- **click:** Ocurre cuando se hace clic en un elemento.
- **dblclick:** Ocurre cuando se hace doble clic en un elemento.
- **mouseover:** Ocurre cuando el ratón se mueve sobre un elemento.
- **mouseout:** Ocurre cuando el ratón se mueve fuera de un elemento.
- **mousemove:** Ocurre cuando el ratón se mueve dentro de un elemento.

Tipos de Eventos Comunes - Ratón

Ejemplo de Eventos de Ratón:

```
const boton = document.getElementById('miBoton');

boton.addEventListener('mouseover', function() {
    boton.style.backgroundColor = 'yellow';
});
```

¿Qué pasa cuando paso con el mouse sobre el botón?
¿Y cuando saco el mouse de encima del botón?

Tipos de Eventos Comunes

Ejemplo de Eventos de Ratón:

```
const boton = document.getElementById('miBoton');

boton.addEventListener('mouseover', function() {
    boton.style.backgroundColor = 'yellow';
});

boton.addEventListener('mouseout', function() {
    boton.style.backgroundColor = '';
});
```

Tipos de Eventos Comunes - Teclado

Eventos de Teclado:

- **keydown:** Ocurre cuando se presiona una tecla.
- **keyup:** Ocurre cuando se libera una tecla.
- **keypress:** Ocurre cuando una tecla es presionada y soltada (obsoleto).

Tipos de Eventos Comunes

Ejemplo de Eventos de Teclado:

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

Objeto "**Event**": contiene información sobre el evento que acaba de ocurrir. Se puede pasar como parámetro a la función que se ejecute cuando suceda el evento, para poder leer/utilizar sus propiedades.
Algunos ejemplos:
event.key → el carácter de la tecla presionada.
event.target → qué elemento del DOM fue clickeado.
event.clientX/Y → Posición del mouse (coordenadas)

Tipos de Eventos Comunes

Ejemplo de Eventos de Teclado:

```
document.addEventListener('keydown', function(prueba) {
    console.log(`Tecla presionada: ${prueba.key}`);
});
```

¿Funciona si le cambio el nombre a “event”?

Tipos de Eventos Comunes

Eventos de Formulario:

- **submit:** Ocurre cuando se envía un formulario.
- **change:** Ocurre cuando el valor de un elemento de formulario cambia.
- **focus:** Ocurre cuando un elemento gana el foco.
- **blur:** Ocurre cuando un elemento pierde el foco.

Ejercicio

Cambiar el color de fondo de un botón al hacer clic.

Resolución

```
const boton = document.getElementById('miBoton');
boton.addEventListener('click', function() {
    boton.style.backgroundColor = 'blue';
});
```

Ejercicio

Crea un botón que, al ser presionado, cambie el contenido de un párrafo.

Resolución

HTML:

```
<p id="myParagraph">Este es el texto original.</p>
<button id="myButton">Cambiar Texto</button>
```

Javascript:

```
document.getElementById('myButton').addEventListener('click', function() {
    document.getElementById('myParagraph').textContent = 'Este es el texto cambiado.';
});
```

Ejercicio

Crea un botón que, al ser presionado, cambie el color de fondo de un div.

Resolución

HTML:

```
<div id="myDiv" style="width: 200px; height: 200px; background-color:
lightblue;"></div>
<button id="styleButton">Cambiar Color</button>
```

Javascript:

```
document.getElementById('styleButton').addEventListener('click', function() {
    document.getElementById('myDiv').style.backgroundColor = 'lightgreen';
});
```

Ejercicio

Crear una lista de elementos que se añaden al hacer clic en un botón.

Resolución

HTML

```
<button id="agregarItem">Agregar Item</button>
<ul id="lista"></ul>
```

Javascript

```
const boton = document.getElementById('agregarItem');
const lista = document.getElementById('lista');

boton.addEventListener('click', function() {
    const nuevoItem = document.createElement('li');
    nuevoItem.textContent = 'Nuevo Item';
    lista.appendChild(nuevoItem);
});
```

Ejercicio

Crea un botón que, al ser presionado, incremente un contador y muestre el número de veces que se ha presionado el botón.

Resolución

HTML:

```
<button id="clickButton">Haz clic aquí</button>
<p>Has hecho clic <span id="clickCount">0</span> veces.</p>
```

Javascript:

```
let count = 0;

document.getElementById('clickButton').addEventListener('click', function() {
    count++;
    document.getElementById('clickCount').textContent = count;
});
```

Ejercicio

Crea un botón que, al ser presionado, oculte o muestre un párrafo.

Resolución

HTML:

```
<p id="toggleParagraph">Este párrafo puede ser ocultado o mostrado.</p>
<button id="toggleButton">Mostrar/Ocultar Párrafo</button>
```

Javascript:

```
document.getElementById('toggleButton').addEventListener('click', function() {
    var paragraph = document.getElementById('toggleParagraph');
    if (paragraph.style.display === 'none') {
        paragraph.style.display = 'block';
    } else {
        paragraph.style.display = 'none';
    }
});
```