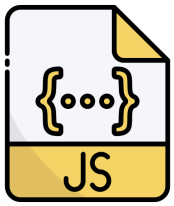
Clase 9

Sitio: Talento Tech
Curso: Desarrollo Web 2_11
Libro: Clase 9

Imprimido por: Ailen Callizaya
Día: martes, 19 de noviembre de 2024, 15:09

Tabla de contenidos

1. □ Clase N° 9



¿Qué vamos a trabajar en esta clase?

- Eventos
- Promesas

Al final de esta clase serás capaz de:

- Identificar y utilizar algunos de los eventos más comunes, como click, mouseover, keydown, submit y load.
- Implementar EventListeners para escuchar y manejar las interacciones del usuario en tu sitio web.
- Entender el concepto de Promesas en JavaScript y cómo se utilizan para manejar operaciones asíncronas de manera más legible y manejable.

2. Eventos

Los **eventos** en JavaScript son **acciones o sucesos que ocurren en una página web**, como la interacción del usuario con elementos HTML, como hacer clic en un botón, mover el mouse sobre una imagen, presionar una tecla, entre otros. Estos eventos activan la ejecución de código JavaScript para responder de manera específica a la acción realizada.

Algunos de estos eventos son:

- **Click:** Ocurre cuando se hace clic en un elemento.
- **Mouseover:** Ocurre cuando el ratón se mueve sobre un elemento.
- **Keydown:** Ocurre cuando una tecla del teclado es presionada.
- **Submit:** Ocurre cuando se envía un formulario.
- **Load:** Ocurre cuando se carga completamente un documento o un recurso.

Para saber cuándo el usuario realiza una interacción con el sitio web, es decir, escuchar las interacciones, se utiliza EventListener.

Importante: Para seleccionar un solo elemento utilizamos **getElementById** si tenemos un conjunto de elementos(Array) usamos **getElementsByTagName**, **getElementsByClassName**.

Este último, debemos utilizar el for para recorrerlo y utilizar cada elemento dentro del array.

2.1. Evento Click

Vamos a seleccionar con **getElementById** y hacer que se ejecute un evento.

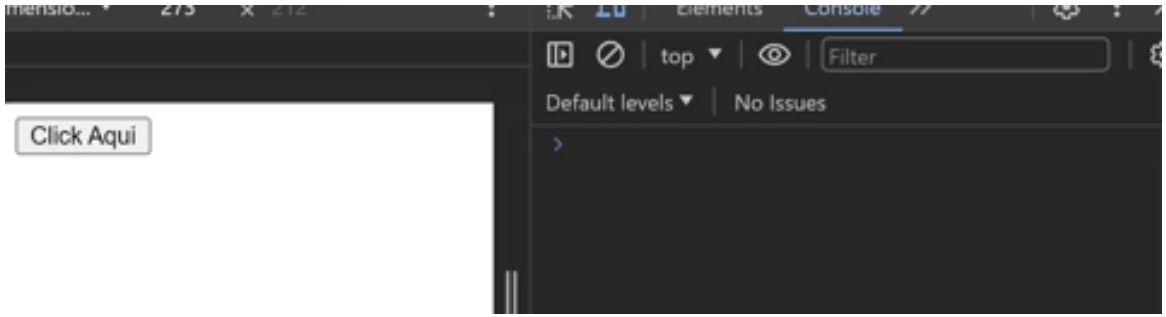
HTML

```
<button id="miElemento">Click Aqui</button>
```

JS

```
// Seleccion un solo Elemento [getElementByID]
let miElemento = document.getElementById("miElemento");
// Añadimos el Evento (tipoDeEvento, funcion que se realiza)
miElemento.addEventListener("click", function()) {
    console.log("Se hizo clic en el elemento.");
});
```

Este es el resultado:



Vamos a seleccionar con **getElementsByClassName**, también podemos realizarlo con **getElementsByTagName**. Esto hace que se ejecute un evento para cada elemento del array.

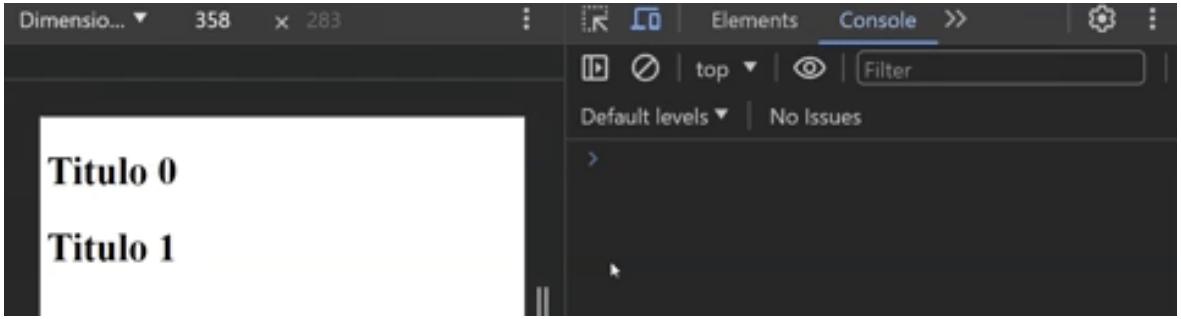
HTML

```
<h2 class="misTitulos">Titulo 1 </h2>
<div>
   <h2 class="misTitulos">Tirulo 2</h2>
</div>
```

JS

```
// Seleccion Conjunto de Elementos [getElementsByTagName, getElementsByClassName]
let titulos = document.getElementsByClassName("misTirulos");
//Como es un Array, lo recorremos con el for
for (let i = 0; i < titulos.length; i++) {
   // Añadimos el Evento (tipoDeEvento, funcion que se realiza)
   titulos[i].addEventListener("click", function() {
         console.log("Se hizo clic en el elemento." + i );
   });
}
```

Este es el resultado:


Dimensio...
358
x
283
Elements
Console
top
Filter
Default levels
No Issues
Titulo 0
Titulo 1

2.2. Evento MouseOver y MouseOut

Es parecido a la función hover de pseudoclases.

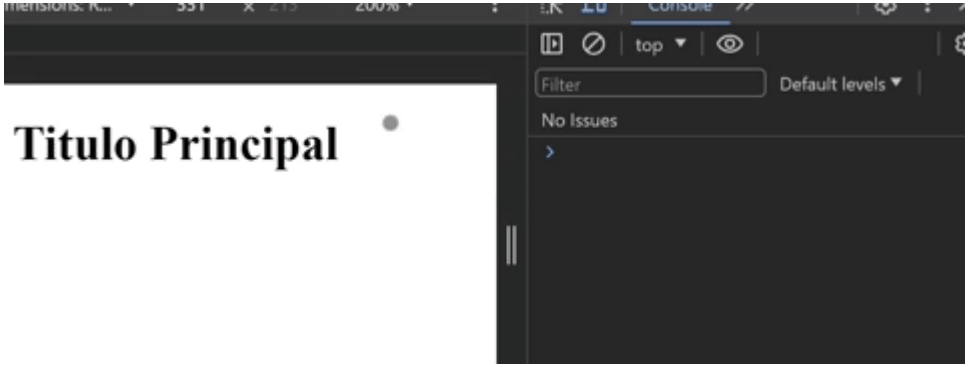
HTML

```
<h1 id="titulo">Titulo Principal</h1>
```

JS

```
let miTitulo = document.getElementById("titulo");
miTitulo.addEventListener('mouseover', function() {
    miTitulo.style.backgroundColor = 'ligthcoral' ;
});
miTirulo.addEventListener('mouseout', function() {
    miTitulo.style.backgroundColor = 'lightblue';
});
```

Este es resultado:


Titulo Principal
top
Filter
Default levels
No Issues

3. Promesas

En JavaScript, las Promesas son como promesas en la vida real. Imagina que le pides a un amigo que haga algo por ti. Esa tarea podría llevar tiempo, y no quieres quedarte esperando. Entonces, tu amigo te hace una promesa: te dará un resultado (éxito o fallo) en algún momento.

En términos más técnicos, las Promesas en JavaScript son útiles cuando hacemos cosas que pueden tomar un tiempo, como cargar datos desde una base de datos o realizar una solicitud a un servidor. Te permiten decir "haz esto y avísame cuando hayas terminado, ya sea bien o mal".

Lo mejor de las Promesas es que hacen que el código sea más fácil de entender, especialmente cuando tienes muchas tareas que ocurren una tras otra. Evitan que tu código se vea como un "infierno de devolución de llamada" (callback hell), haciendo que la lógica asíncrona sea más legible y manejable.

La estructura básica de una Promesa se compone de tres estados:

- **Pending (pendiente):** Estado inicial, la Promesa está en curso.
- **Fulfilled (cumplida)**: La operación se completó exitosamente.
- **Rejected (rechazada):** La operación ha fallado.

```
// Crear una Promesa
let miPromesa = new Promise(function(resolve, reject) {
    // Simular una tarea que toma tiempo (por ejemplo, una solicitud a un servidor)
    setTimeout(function() {
        let exito = true; // Cambiar a false para simular un fallo
        if (exito) {
```

```
            resolve("La terea fue completada con éxito");
        } else {
            reject("Hubo un problema al completar la terea");
        }
    }, 2000); // Simulamos una tarea que toma 2 segundos
});
// Utilizar la Promesa
miPromesa.then(function(resultado) {
    console.log("Éxito:", resultado);
})
.catch(function(error) {
   console.log("Error:", error);
});
```

En este ejemplo, se crea una promesa que simula una operación asíncrona (en este caso, un temporizador). Si la operación es exitosa, se llama a **resolve** con el resultado; si falla, se llama a **reject** con un mensaje de error. Luego, se utiliza el método .then para manejar el caso de éxito y el método **.catch** para manejar el caso de error.

Las Promesas son una parte integral de la programación asíncrona en JavaScript y se utilizan ampliamente, especialmente en el contexto de solicitudes a servidores, manipulación de archivos y otras operaciones asíncronas.

4. ? Pregunta de reflexión

En el contexto de las Promesas;

¿Cómo crees que facilitan el manejo de operaciones asíncronas en comparación con el uso de callbacks?

¿Puedes mencionar una situación práctica donde sería beneficioso utilizar eventos en JavaScript para mejorar la interactividad de una página web?

5. Desafío #9

Manipular el **DOM** implica modificar y acceder a elementos del DOM, o crear nuevos con **createElement**.

Manipulando el DOM:

- Crear un botón con **createElement**
- Sumarle al botón creado el ID '**btnCarrito**'.
- Sumar con .**innetHTML** el texto que tendrá el botón en su interior.

Ejemplo: "Agregar al carrito".

- Luego, escuchar con **addEventListener** el evento "**click**" sobre el botón generado, asegurando que la función que vas a crear dispare una salida en un elemento HTML cuando se presione. Esto lo podrás lograr creando un nuevo elemento **<h3>** con **createElement** y modificando texto del botón con **.innterHTML**

Ejemplo: "Agregado"

Una vez hecho esto; solo queda agregar el botón generado con **.apprendChild**

TIP: El evento puede asociarse con **addEventListener**, o la propiedad **onclick**.