Recuerden poner a grabar la clase

Clase 9

DOM III

Diplomatura UNTREF

Temario

DOM:

- QuerySelector y QueryselectorAll
- Activacion de eventos en elementos HTML dinamicos
- El metodo createElement
  - append()
  - appendChild()
  - removeElement

QuerySelector y QuerySelectorAll

Repasemos DOM

Cuando navegamos por Internet y visitamos páginas web, el contenido que vemos en el navegador está construido utilizando lenguajes de marcado como **HTML (HyperText Markup Language)**.

HTML es como el esqueleto de una página, ya que define la estructura básica del contenido, como párrafos, títulos, imágenes, enlaces, etc. Estos elementos se organizan en una jerarquía de nodos, formando lo que se conoce como el **Document Object Model o DOM.**

Repasemos DOM

El **DOM** es una representación estructurada y manipulable del contenido de una página web. Es como un árbol en el que los elementos HTML son nodos conectados entre sí.

Cada nodo representa un elemento en la página, como una etiqueta o <p>, y puede contener otros nodos (elementos) como hijos, padres o hermanos.

Repasemos DOM

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

En JavaScript, podemos interactuar con el DOM para acceder a elementos específicos y realizar acciones sobre ellos. Aquí es donde entran en juego los métodos **querySelector y querySelectorAll**.

QuerySelector

El **método querySelector** es una función que nos permite seleccionar un solo elemento dentro del **DOM** utilizando un selector **CSS**. Un selector **CSS** es una cadena de texto que describe cómo se ven los elementos en la página web. Puede basarse en el tipo de elemento, **clases**, **ID**, **atributos** o su estructura en el árbol del **DOM**.

QuerySelector

QuerySelector

querySelector es un método que pertenece al objeto document en JavaScript y se utiliza para seleccionar un solo elemento del DOM (Document Object Model) basado en un selector CSS especificado.

Sintaxis QuerySelector

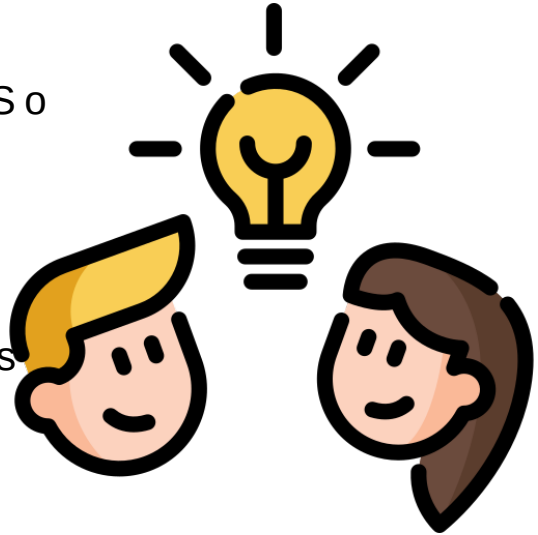
Donde selector es una cadena de texto que representa el selector CSS que se utilizará para identificar el elemento deseado.

```
document.querySelector(selector);
```

Sintaxis QuerySelector

Selector CSS: El selector CSS puede ser cualquier tipo de selector válido, similar al que usarías en una hoja de estilo CSS o en JavaScript.

Puede ser un nombre de etiqueta **(p, div, h1, etc.)**, una clase (.mi-clase), un ID (#mi-id), un atributo, o incluso selectores más complejos utilizando combinaciones de estos elementos.



Valor de retorno: querySelector devuelve el primer elemento que coincida con el selector proporcionado.

Si no se encuentra ningún elemento que coincida, el valor de retorno es null.

QuerySelector

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

En este ejemplo, querySelector selecciona el primer párrafo dentro del elemento con el ID "miDiv" y que también tiene la clase "miClase". Luego, se modifica el contenido de ese párrafo utilizando textContent.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de querySelector</title>
</head>
<body>
    <div id="miDiv">
        <p class="miClase">Este es el primer párrafo.</p>
        <p>Este es el segundo párrafo.</p>
    </div>
    <script>
        // Seleccionamos el primer párrafo utilizando querySelector
        const miParrafo = document.querySelector('#miDiv
p.miClase');

        // Modificamos el contenido del primer párrafo seleccionado
        miParrafo.textContent = '¡Hola! Este texto ha sido
modificado por JavaScript.';
    </script>
</body>
</html>
```

QuerySelectorAll

QuerySelectorAll

querySelectorAll es un método del objeto document en JavaScript que se utiliza para seleccionar múltiples elementos del **DOM** basados en un selector CSS especificado.

A diferencia de **querySelector**, que devuelve solo el primer elemento coincidente, querySelectorAll devuelve una lista (NodeList) que contiene todos los elementos que coinciden con el selector.

Sintaxis QuerySelectorAll

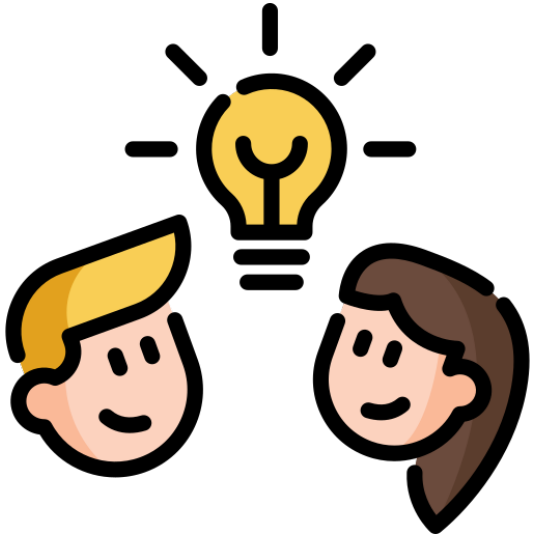
Al igual que con **querySelector**, selector es la cadena de texto que representa el selector CSS que se utilizará para identificar los elementos deseados.

```
document.querySelectorAll(selector);
```

Sintaxis QuerySelectorAll

Selector CSS: El selector CSS para querySelectorAll puede ser cualquier tipo de selector válido, similar a querySelector.

Valor de retorno: querySelectorAll devuelve una lista (NodeList) de todos los elementos que coinciden con el selector proporcionado. Si no se encuentran elementos que coincidan, la lista estará vacía, pero nunca será null.



Sintaxis QuerySelectorAll

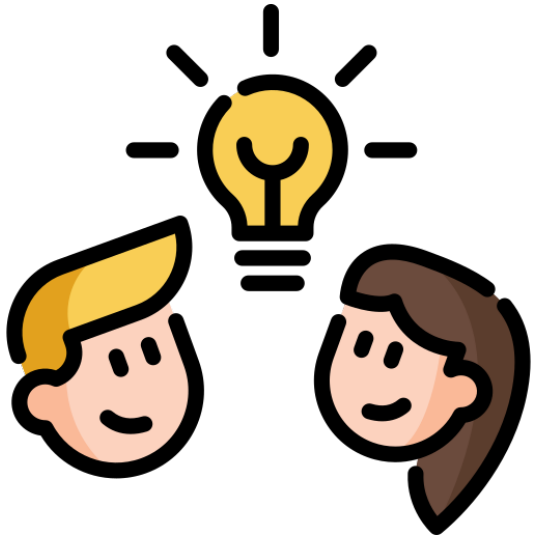
En este ejemplo, **querySelectorAll** selecciona todos los elementos <li> que están dentro de una lista desordenada <ul>.

Luego, utilizamos un bucle forEach para recorrer la lista de elementos seleccionados y modificar su contenido.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo de querySelectorAll</title>
</head>
<body>
    <ul>
        <li>Elemento 1</li>
        <li>Elemento 2</li>
        <li>Elemento 3</li>
    </ul>
    <script>
        // Seleccionar todos los elementos li dentro de la lista ul utilizando querySelectorAll
        const listaElementos = document.querySelectorAll('ul li');

        // Iterar sobre la lista de elementos y modificarlos
        listaElementos.forEach((elemento, index) => {
            elemento.textContent = `Elemento ${index + 1} ha sido modificado`;
        });
    </script>
</body>
</html>
```

Resumen

En resumen, tanto **querySelector** como **querySelectorAll** son métodos útiles para seleccionar y manipular elementos del **DOM** en JavaScript. Mientras que **querySelector** se utiliza para seleccionar un solo elemento, **querySelectorAll** devuelve una lista de elementos que coinciden con el selector especificado.

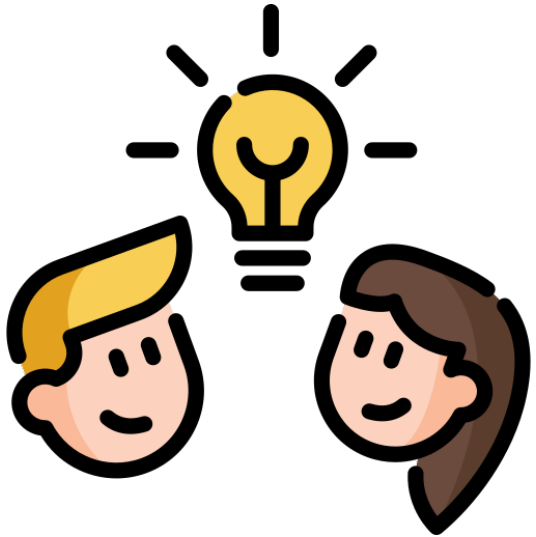


Estos métodos son herramientas poderosas para trabajar con el contenido y la estructura de una página web de manera dinámica y programática.

Activación de eventos en HTML

Activación de eventos en HTML

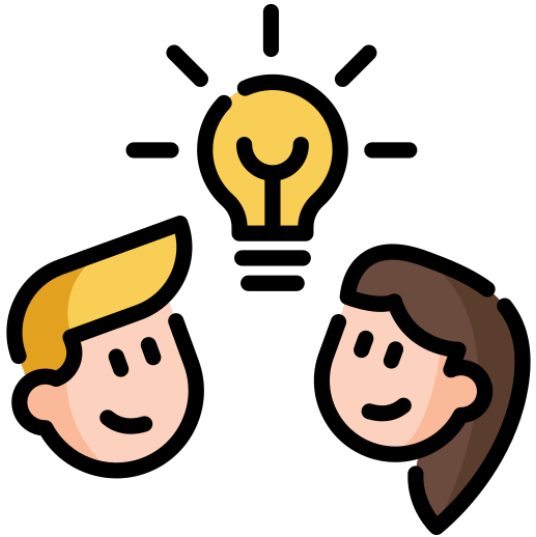
La activación de eventos en HTML dinámicos a través del DOM se refiere a la capacidad de interactuar con elementos que se han agregado, creado o modificado en una página web en tiempo de ejecución.



Cuando trabajamos con JavaScript, podemos crear contenido HTML de manera dinámica, ya sea agregando nuevos elementos, cambiando su contenido o modificando sus atributos. El DOM nos permite acceder a estos elementos dinámicos y realizar acciones sobre ellos.

Activación de eventos en HTML

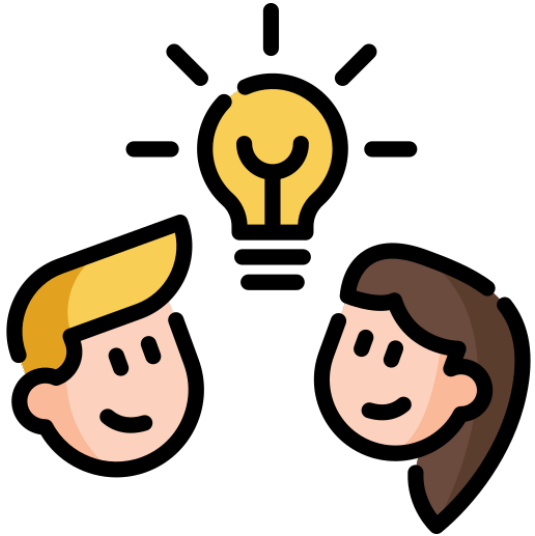
La activación de eventos en HTML dinámicos a través del DOM se refiere a la capacidad de interactuar con elementos que se han agregado, creado o modificado en una página web en tiempo de ejecución.



Cuando trabajamos con JavaScript, podemos crear contenido HTML de manera dinámica, ya sea agregando nuevos elementos, cambiando su contenido o modificando sus atributos. El DOM nos permite acceder a estos elementos dinámicos y realizar acciones sobre ellos.

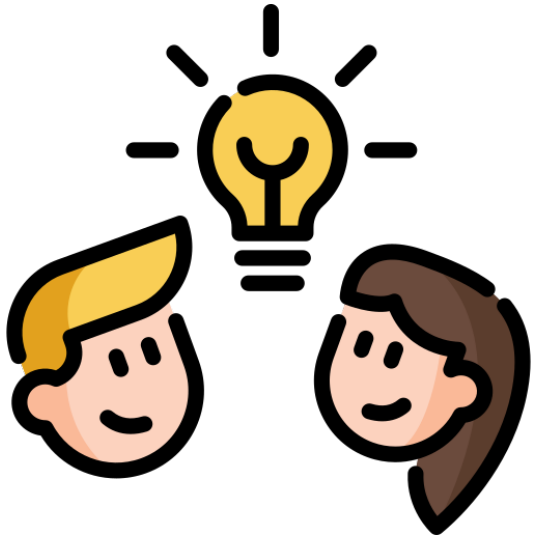
Activación de eventos en HTML

es posible que los eventos asociados a ellos no estén disponibles de inmediato. Para resolver este problema, se utilizan conceptos como la **propagación de eventos** y la **delegación de eventos.**



Propagación de eventos

La propagación de eventos es un comportamiento del DOM donde un evento se propaga desde el elemento objetivo hasta el elemento raíz del documento. Esto significa que si se hace clic en un elemento hijo, el evento se propagará a sus elementos padres a lo largo de la jerarquía del DOM.



Propagación de eventos

Cuando se agregan elementos dinámicos al DOM, pueden no tener eventos asociados de forma predeterminada. Para activar eventos en estos elementos, se puede utilizar la propagación de eventos junto con el método addEventListener.

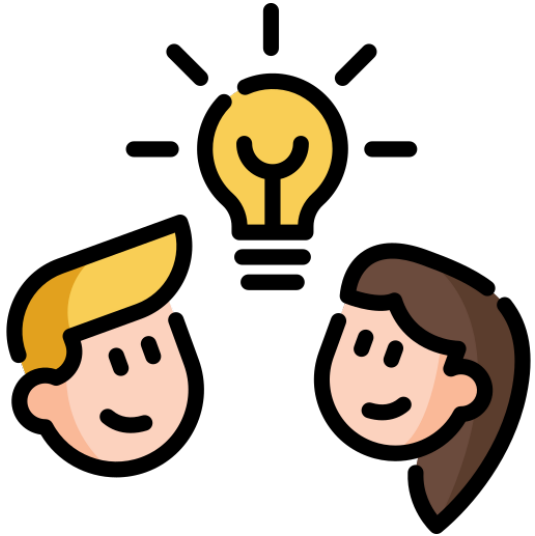
```html
<!DOCTYPE html>
<html>
<head>
    <title>Eventos en elementos dinámicos</title>
</head>
<body>
    <button id="boton-agregar">Agregar elemento</button>
    <div id="contenedor-dinamico"></div>
    <script>
        // Obtener el botón y el contenedor donde se agregarán los elementos dinámicos
        const botonAgregar = document.getElementById('boton-agregar');
        const contenedorDinamico = document.getElementById('contenedor-dinamico');

        // Función para crear y agregar un nuevo elemento dinámico al contenedor
        function agregarElemento() {
            const nuevoElemento = document.createElement('p');
            nuevoElemento.textContent = '¡Soy un nuevo elemento!';
            contenedorDinamico.appendChild(nuevoElemento);
        }

        // Agregar un event listener al botón para activar la función al hacer clic
        botonAgregar.addEventListener('click', agregarElemento);

        // Agregar un event listener al contenedor para activar el evento en los elementos dinámicos
        contenedorDinamico.addEventListener('click', function(event) {
            // Verificar si el clic se realizó en un elemento <p> dentro del contenedor
            if (event.target.tagName === 'P') {
                // Hacer algo cuando se haga clic en el elemento <p> dinámico
                console.log('¡Se hizo clic en un elemento dinámico!');
            }
        });
    </script>
</body>
</html>
```

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Delegación de eventos

La **delegación de eventos** es un patrón que aprovecha la propagación de eventos para asignar un evento a un elemento padre en lugar de a los elementos hijos individuales.

Esto es útil cuando tienes muchos elementos similares que se generan dinámicamente y no quieres agregar eventos a cada uno por separado.

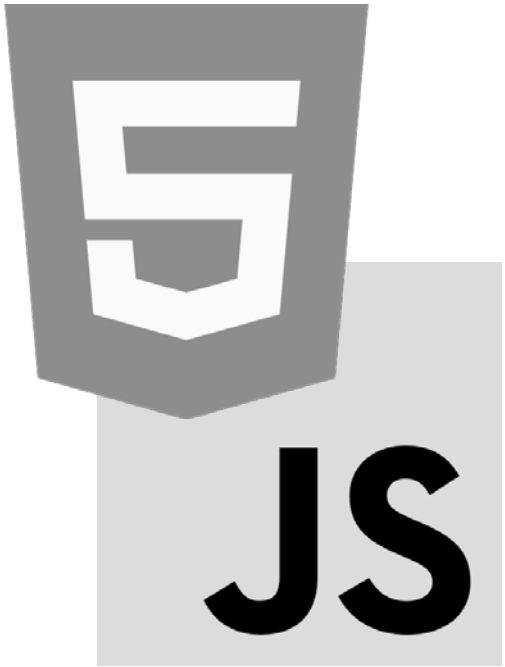


createElement

CreateElement

El método **createElement** es una función que se utiliza en JavaScript para crear un nuevo elemento HTML de forma dinámica dentro del **Document Object Model (DOM).** Este método es parte del **objeto document,** que representa el documento HTML actual cargado en el navegador.

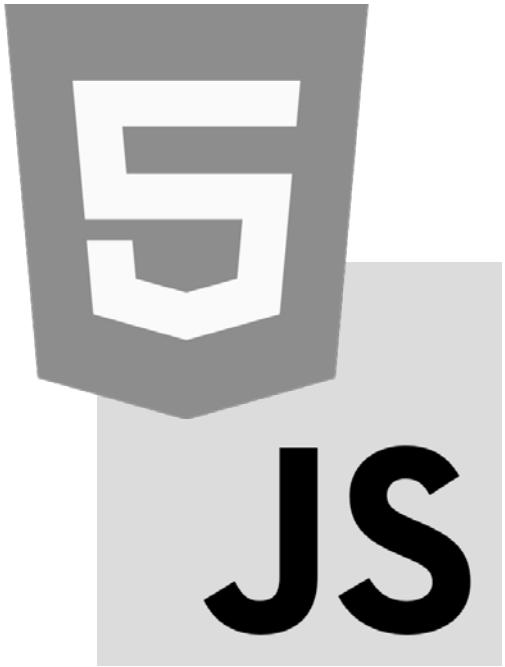
El propósito principal de **createElement** es permitir a los desarrolladores crear elementos HTML desde JavaScript y luego agregarlos a la página web para interactuar con ellos o mostrar contenido dinámico.



CreateElement

El método **createElement** es una función que se utiliza en JavaScript para crear un nuevo elemento HTML de forma dinámica dentro del **Document Object Model (DOM).** Este método es parte del **objeto document,** que representa el documento HTML actual cargado en el navegador.

El propósito principal de **createElement** es permitir a los desarrolladores crear elementos HTML desde JavaScript y luego agregarlos a la página web para interactuar con ellos o mostrar contenido dinámico.



CreateElement: append

El método append es una función poderosa que permite agregar uno o varios nodos al final de un elemento padre. Puedes pensar en un nodo como cualquier elemento dentro de un documento HTML, como etiquetas , <p>, <img>, etc. También puedes incluir texto directamente.

Imagina que tienes un elemento HTML con el id "container" en tu página web, y deseas agregar un nuevo párrafo <p> y un enlace <a> al final de este contenedor:

```
<div id="container">
  <!-- Contenido existente -->
</div>

////javaScript
const container = document.getElementById("container");

const newParagraph = document.createElement("p");
newParagraph.textContent = "Este es un nuevo párrafo.";

const newLink = document.createElement("a");
newLink.textContent = "Enlace";
newLink.href = "https://www.ejemplo.com";

container.append(newParagraph, newLink);
```

CreateElement: appendChild

El método appendChild se utiliza para agregar un solo nodo hijo al final de un elemento padre. Esto puede ser especialmente útil cuando solo deseas agregar un único elemento al final de otro.

Supongamos que tienes un elemento HTML con el id "list" y quieres agregar un nuevo elemento de la lista <li> a este elemento:

CreateElement: removeChild

El método removeChild te permite eliminar un nodo hijo específico de un elemento padre. Esto es útil cuando deseas eliminar un elemento particular del DOM.

Supongamos que tienes el mismo elemento de lista <ul> que mencionamos anteriormente y deseas eliminar el primer elemento de la lista:

```
<ul id="list">
  <li>Elemento 1</li>
  <li>Elemento 2</li>
</ul>

//////JavaScript

const list = document.getElementById("list");

const firstListItem = list.firstElementChild; //
Obtenemos el primer hijo

if (firstListItem) {
  list.removeChild(firstListItem);
}
```

¿Dudas o consultas?

¡Muchas Gracias!