

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso

(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148

[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)

# Curso:

# Professional Webmaster



Módulo 3:

# JavaScript y maquetado avanzado

Unidad 4:

## JavaScript (parte 2)



## Presentación

En esta unidad vemos cómo podemos interactuar con los elementos del DOM con JavaScript.



## Objetivos

### Que los participantes logren...

- Conocer el DOM para poder acceder, añadir y cambiar elementos del documento.
- Manipular las propiedades y funciones de los objetos.
- Conocer las diferentes formas de uso de eventos.



## Bloques temáticos

1. DOM.
2. Eventos.

# 1.DOM

El DOM (Document Object Model en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

El DOM surgió a partir de la implementación de JavaScript en los navegadores. A esta primera versión también se la conoce como DOM o "Legacy DOM". Hoy en día el grupo WHATWG es el encargado de actualizar el estándar de DOM.

## Selectores de elementos

Los selectores api proveen métodos que hacen más fácil y rápido devolver elementos del nodo Element del DOM mediante emparejamiento de un conjunto de selectores. Esto es mucho más rápido que las técnicas anteriores, donde fuera necesario, por ejemplo usar un loop en un código JavaScript para localizar el ítem específico que quisieras encontrar.

### **`document.querySelector()`**

Devuelve la primera coincidencia del (elemento) Element nodo dentro de las subramas del nodo. Sino se encuentra un nodo coincidente, se devuelve null .

### **`document.querySelectorAll()`**

Devuelve un listado de nodos NodeList conteniendo todos los elementos del nodo coincidentes( Element) dentro de las subramas del nodo, o Devuelve un Listado de Nodos vacío NodeList sino se encuentran coincidencias.

Estos métodos aceptan uno o más selectores separados por comas entre cada selector para determinar qué elemento o elementos deben ser devueltos. por ejemplo para seleccionar todos los elementos (p) del párrafo en un documento donde la clase CSS sea tanto warning or note, puedes hacer lo siguiente:

```
const especial = document.querySelectorAll('p.warning, p.note')
```

También por usar query para etiquetas id. Por ejemplo:

```
const destacados = document.querySelector('#principal, #bienvenidos, #notas')
```

Luego de ejecutar el código de arriba, la variable contiene el primer elemento del documento, su ID puede ser uno de los siguientes: principal, bienvenidos, o notas.

Podes usar cualquier selector CSS con los métodos **querySelector()** y **querySelectorAll()**.

### **document.getElementById()**

Este método permite seleccionar una elemento del DOM usando el atributo id del mismo.



```
const mensaje = document.getElementById('mensaje')
```

### **document.getElementsByClassName()**

Este método devuelve un array o lista de elementos que contengan la clase que se especifique como parámetro.



```
const items = document.getElementsByClassName('item')
```

### **document.getElementsByTagName()**

Este método devuelve un array o lista de elementos cuya etiqueta html sea la que se especifique como parámetro.



```
const parrafos = document.getElementsByTagName('p')
```

## Crear elementos y agregarlos al DOM

Podemos crear cualquier tipo de elemento usando el método **document.createElement()**. Este método recibe como parámetro el nombre de la etiqueta del elemento que deseamos crear, por ejemplo li.

Una vez creado el elemento, este no será visible al usuario hasta que lo agreguemos explícitamente al DOM, para ello, los elementos o nodos del DOM cuentan con un método llamado **appendChild()** que recibe como parámetro el elemento que se desee agregar.



```
<body>
  <div id="div1">El texto siguiente es dinámico</div>

  <script>
    function agregarElemento() {
      var newDiv = document.createElement('div');
      newDiv.innerHTML = 'Hola!';

      document.body.appendChild(newDiv);
    }
    agregarElemento();
  </script>
</body>
```

## Eliminar elementos del DOM

Para eliminar elementos del DOM solo debemos obtener una referencia al elemento usando cualquiera de los métodos de selección que mencionamos previamente, y llamar a al método **remove()**.

```
<div id="div1">Este es el div1</div>
<div id="div2">Este es el div2</div>
<div id="div3">Este es el div3</div>

<script>
  const elemento = document.getElementById('div2')
  elemento.remove();
</script>
```

## innerHTML e innerText

Las propiedades de los elementos del DOM `innerHTML` e `innerText` hacen referencia al contenido de los nodos. Mientras que `innerText` **permite insertar** y manipular **contenido sencillo** dentro de los nodos, `innerHTML` nos permite **incluir código HTML** más complejo sin la necesidad de crear los elementos a mano.

```
<body>
  <p id="demo">Este elemnto tiene espacio extra y contiene
  <span>una etiqueta span</span></p>
  <script>

    function getHTML() {
      alert(document.getElementById('demo').innerHTML);
    }

    function getInnerText() {
      alert(document.getElementById('demo').innerText);
    }
  </script>
</body>
```

## Atributos de los elementos

Podemos acceder a los atributos de cualquier elemento que hayamos seleccionado simplemente usando el nombre del atributo como propiedad del nodo, Por ejemplo



```
const miLink = document.getElementsByTagName('a')[0];  
alert(miLink.href)
```

De esta forma veremos el mensaje de alerta con el contenido del atributo **href** de nuestro link. Del mismo modo podemos escribir la propiedad y cambiar su valor según necesitemos.

## Estilos de los elementos

Los estilos CSS se pueden manipular mediante javascript accediendo a la propiedad **style** que tienen todos los elementos o nodos del DOM. Hay que tener en cuenta que las propiedades cuyo nombre lleva un guión en su nombre (estilo llamado kebab-case) cómo font-size se convierten al estilo camelCase, resultando en fontSize.




```
const intro = document.getElementById('intro')  
intro.style.backgroundColor = '#ff00ff'
```

## 2. Eventos

Javascript permite responder a eventos en el DOM mediante el uso de funciones. Existen 3 formas distintas de registrar gestores de eventos para un elemento del DOM:


### EventTarget.addEventListener



```
myButton.addEventListener('click', function(){  
  alert('Hello world');  
}, false);
```

Esta es la forma recomendada y la más moderna de las 3. Permite registrar más de un listener por evento, lo que brinda una mayor flexibilidad.


### Atributo HTML



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

Esta forma es la menos recomendada porque incrementa el tamaño del HTML y dificulta su lectura, además de que resulta muy complejo incluir múltiples instrucciones.

## Propiedad del elemento DOM



```
myButton.onclick = function(event){  
    alert('Hello world');  
};
```

Este método solo permite registrar un listener por evento.



## Bibliografía utilizada y sugerida

### Libros y otros manuscritos:

**Eguíluz Pérez, Javier.** Introducción a JavaScript. España: [www.librosweb.es](http://www.librosweb.es) 2008.