



FULL STACK

**Comenzamos en unos
minutos**

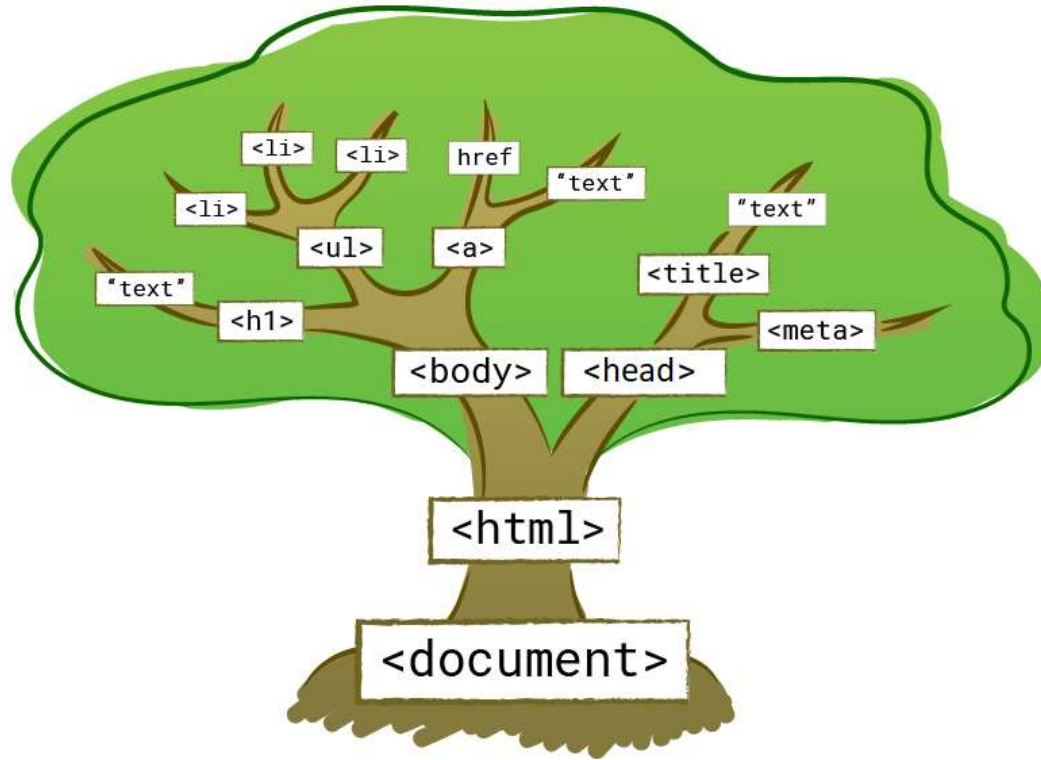
ACADEMY
by NUMEN

JavaScript: Clase 7

Introducción al DOM

HTML con JavaScript

¿Se acuerdan del DOM?



DOM API

El objeto windows que vimos la clase pasada posee una API que nos permite interactuar con los elementos del DOM.

```
console.log(window.document)
```



```
▼ #document  
  <!DOCTYPE html>  
  <html lang="en">  
    ► <head>...</head>  
    ► <body>...</body>  
  </html>
```

DOM API

Si quisiéramos acceder a cada parte del documento, simplemente utilizamos la notación del punto de esta forma:

```
console.log(document.doctype)
```

```
console.log(document.documentElement)
```



```
<!DOCTYPE html>
```



```
<html lang="en">  
  > <head>...</head>  
  > <body>...</body>  
</html>
```

```
console.log(document.head)
```

```
console.log(document.body)
```



```
> <head>...</head>
```



```
> <body>...</body>
```

Manejo del DOM

Para poder interactuar con el DOM, Javascript utiliza una serie de métodos que rastrean selectores. Estos selectores pueden ser desde etiquetas HTML hasta valores de ciertos atributos.

Recorramos estos selectores uno por uno.



Selectores: Por ID

Finalmente, y no menos importante, está la posibilidad de manipular el DOM por medio del atributo ID o selector único.

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



```
console.log(document.getElementById("titulo"))
```




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

Selectores: Query Selector

Durante mucho tiempo se han usado los métodos para conectar por nombre, nombre de clase y nombre de etiqueta. Sin embargo, en las últimas versiones de Javascript, han aparecido dos nuevos métodos que reemplazan de manera más eficiente a estos.

Un de ellos es el `querySelector()`. Veamos que hace:



```
<section id="seccion">
  <h2 class="titulo">Un Titulo</h2>
</section>
```

```
console.log(document.querySelector("h2"))
console.log(document.querySelector("#seccion"))
console.log(document.querySelector(".titulo"))
```



```
practica.js:1
<h2 class="titulo">Un Titulo</h2>
```

```
practica.js:2
▶ <section id="seccion">...</section>
```

```
practica.js:3
<h2 class="titulo">Un Titulo</h2>
```

Selectores: Query Selector All

`querySelectorAll()` es similar a `querySelector()` con la diferencia que nos trae todos los elementos del mismo un mismo tipo. Vamos...

```
<ul>
  <li>Fuego</li>
  <li>Tierra</li>
  <li>Agua</li>
  <li>Aire</li>
</ul>
```

```
console.log(document.querySelectorAll("li"))
```

```
▼ NodeList(4) [li, li, li, li] ⓘ
  ▶ 0: li
  ▶ 1: li
  ▶ 2: li
  ▶ 3: li
    length: 4
  ▶ [[Prototype]]: NodeList
```

Métodos para atributos: get

El método `getAttribute()` nos devuelve el valor de un atributo. Veámoslo:



```
console.log(document.querySelector("ul").getAttribute("name"))
```

```
<ul name="lista">  
  <li>Fuego</li>  
  <li>Tierra</li>  
  <li>Agua</li>  
  <li>Aire</li>  
</ul>
```




```
lista
```

Métodos para atributos: set

El método `setAttribute()` nos permite agregar un atributo y su valor a una etiqueta. Veámoslo:

```
<a href="#">Un enlace</a>
```



```
console.log(document.querySelector("a").setAttribute("target", "_blank"))
```



```
<html lang="en">
  <head>...</head>
  <body>
    <a id="enlace" href="#" target="_blank">Un enlace</a>
    <script src="practica.js"></script>
    <!-- Code injected by live-server -->
    <script type="text/javascript">...</script>
  </body>
</html>
```

Texto y HTML

Javascript nos permite manipular el DOM a través de ciertas propiedades. `textContent` sirve para agregar contenido a una etiqueta, `innerHTML` para insertar etiquetas con contenido a una etiqueta y `outerHTML` para reemplazar una etiqueta por etiquetas con contenido.

```
<article id="articulo">  
  
</article>
```



```
const $articulo = document.getElementById('articulo')
```

```
$articulo.textContent = texto
```



```
$articulo.innerHTML = texto
```



```
$articulo.outerHTML = texto
```



```
let texto = `  
<p>  
    Lorem ipsum dolor <b>sit amet</b>  
consectetur  
    <mark>adipisicing elit</mark>.  
Quisquam, nemo.  
</p>  
`
```



Creando Elementos del DOM

Otra funcionalidad que nos provee Javascript es la de crear Elements en el DOM. Para esto disponemos de los métodos `createElement()` y `appendChild()`. Veamos como funciona:

```
<section class="cards">  
  
</section>
```



```
const $cards = document.querySelector(".cards"),  
    $figure = document.createElement("figure"),  
    $img = document.createElement("img"),  
    $figcaption = document.createElement("figcaption"),  
    $figcaptionText = document.createTextNode("Hola Mundo")
```

```
$img.setAttribute("src",  
    "http://pm1.narvii.com/6378/71077675a874957c3  
    8a660206fe3ca672b1569f4_00.jpg");  
$img.setAttribute("alt", "yoda");  
$img.setAttribute("width", "150");  
$figcaption.appendChild($figcaptionText);
```

```
$cards.appendChild($figure);  
$figure.appendChild($img);  
$figure.appendChild($figcaption);
```



Creando HTML dinamico

Javascript nos permite generar HTML de manera dinámica iterando sobre arreglos que contengan información. Para esto usaremos alguno de los métodos callbacks antes vistos.

```
const estaciones = ["Verano", "Otoño", "Invierno",  
"Primavera"],  
    $ul = document.createElement("ul")  
  
document.write("<h3>Estaciones del Año</h3>");  
document.body.appendChild($ul)
```



Estaciones del Año

- Verano
- Otoño
- Invierno
- Primavera



```
estaciones.forEach(el => {  
    const $li = document.createElement("li");  
    $li.textContent = el;  
    $ul.appendChild($li);  
})
```


Creando Fragmentos

Un **fragmento** es “trozo” del DOM que puede contener nodos. La idea del uso de los fragmentos es evitar que con cada cambio que se realice en el DOM, este se renderice completamente. En su lugar, solo se renderiza el fragmento contenedor del nodo en cuestión.

```
const estaciones = ["Verano", "Otoño", "Invierno",  
"Primavera"],  
  
$ul = document.createElement("ul"),  
$fragment = document.createDocumentFragment();
```

Estaciones del Año

- Verano
- Otoño
- Invierno
- Primavera

```
estaciones.forEach(el => {  
  const $li = document.createElement("li");  
  $li.textContent = el;  
  $fragment.appendChild($li);  
})
```


```
document.write("<h3>Estaciones del Año</h3>")  
$ul.appendChild($fragment);  
document.body.appendChild($ul);
```

CSS con JavaScript

Estilos con Javascript

A través de Javascript podemos acceder a estilos CSS, agregarlos, quitarlos y mucho más. A través de la propiedad style de la API del document, podemos ver toda la variedad de propiedades CSS de las que podremos disponer.

```
<!-- index.html -->  
<a href="#" class="enlace"></a>
```




```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log($enlace.style)
```

Método `getAttribute()`

Este método nos permite acceder al contenido del atributo `style` de una etiqueta HTML. Si este atributo posee varias propiedades, nos devolverá toda la colección de propiedades que posea.

```
<!-- index.html -->  
<a href="#" class="enlace" style="color: red"></a>
```



```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
console.log($enlace.getAttribute("style"))
```

Estilos por notación del punto

Otra forma en la que podríamos agregar un valor CSS es conectar con la propiedad CSS implícita de objeto `CSSStyleDeclaration`. Veámoslo:

```
<!-- index.html -->  
<a href="#" class="enlace">Esto es un enlace</a>
```

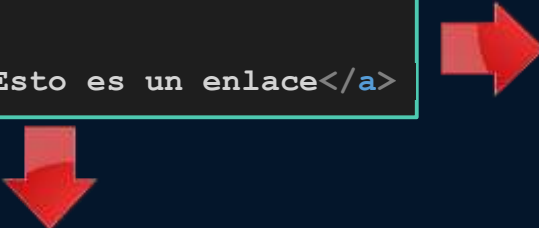


```
// index.js  
const $enlace = document.querySelector(".enlace");  
  
$enlace.style.backgroundColor = "pink"  
$enlace.style.color = "blue"  
$enlace.style.padding = "20px"
```

Agregando y removiendo clases

Lo último que nos queda por saber sobre CSS es como agregar una clase adicional a una etiqueta. Para ello utilizaremos el método `add()`. Veámoslo:

```
<!-- index.html -->
<a href="#" class="enlace">Esto es un enlace</a>
```



```
// index.js
const $enlace = document.querySelector(".enlace");
console.log($enlace.classList)

$enlace.classList.add("agregar-color")
$enlace.classList.remove("agregar-color")
```

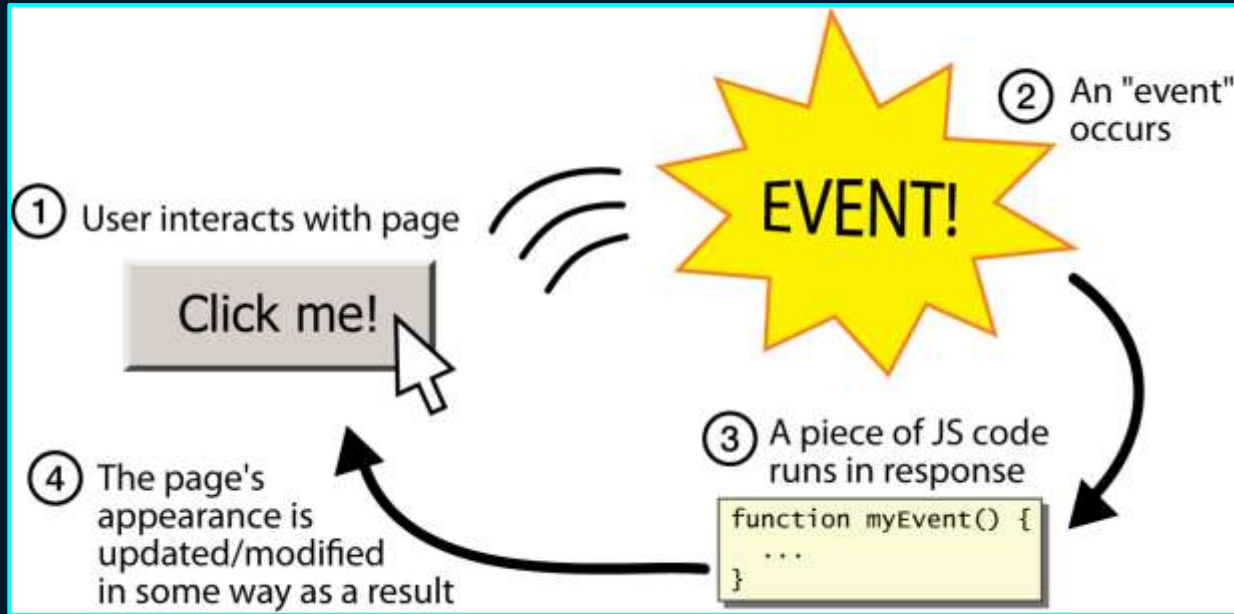
```
/* style.css */
.enlace{
    text-decoration: none;
}

.agregar-color {
    color: red;
}
```

Eventos en JavaScript

EVENTOS

Los eventos son un tipo de método que nos permite ejecutar funciones en determinadas situaciones, como por ejemplo, cuando el usuario interactúa con un botón.



Eventos como atributos

Una forma de manejar eventos, aunque no la recomendada, es a través de atributos de las etiquetas HTML. Veámos un ejemplo:


```
<!-- index.html -->  
<button onclick="alert('Alerta desde un evento')">Mi primer evento</button>
```

Llamando una función por atributo

En el ejemplo anterior ejecutamos una función de alerta, de esas que vienen colgadas del objeto window.

Ahora nos toca llamar una función que hayamos programado en nuestro archivo.js.

```
<!-- index.html -->  
<button onclick="holaMundo()">Mi primer evento</button>
```





```
// index.js  
function holaMundo() {  
    alert("Evento desde archivo.js")  
}
```

Eventos semánticos

Así como podíamos agregar propiedades CSS a través de la notación de punto, también es posible manejar eventos de esta forma. Veamos un ejemplo:

```
<!-- index.html -->  
<button id="evento">Mi primer evento</button>
```




```
// index.js  
function holaMundo() {  
    alert("Evento desde archivo.js")  
}  
  
const $evento = document.getElementById("evento")  
  
$evento.onclick = holaMundo;
```

```
// index.js  
const $evento = document.getElementById("evento")  
  
$evento.onclick = function () {  
    alert("Evento desde archivo.js")  
}
```

Eventos Listeners (manejadores)

El `addEventListener` es un método que nos permite ejecutar múltiples eventos. Como parámetro recibe el evento elegido y la función a ejecutar.

```
<!-- index.html -->
<button id="evento">Mi primer evento</button>
```



```
// index.js
const $evento = document.getElementById("evento")

function holaMundo() {
  alert('Hola Mundo')
}


function adiosMundo() {
  alert('Adios Mundo')
}

$evento.addEventListener("click", holaMundo)
$evento.addEventListener("click", adiosMundo)
```

Función anónima como parámetro

Además de recibir una función declarada, `addEventListener()` puede recibir como parámetro una función anónima.

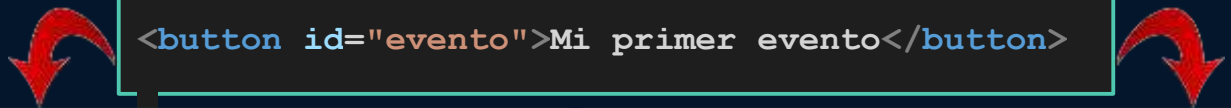
```
<!-- index.html -->  
<button id="evento">Mi primer evento</button>
```



```
// index.js  
const $evento = document.getElementById("evento")  
  
$evento.addEventListener("click", function() {  
    alert('Hola Mundo')  
}))
```

Escuchar funciones con parámetros

¿Qué pasaría si deseamos que la función que pasamos por parámetro en el `addEventListener` reciba parámetros? ¿Como lo programaríamos? Veámoslo:




```
<!-- index.html -->
<button id="evento">Mi primer evento</button>
```

```
// index.js
const $evento = document.getElementById("evento")

function holaMundo(nombre = "usuario") {
  alert(`Hola ${nombre}`)
}


$evento.addEventListener('click', holaMundo)
```



```
// index.js
const $evento = document.getElementById("evento")

function holaMundo(nombre = "usuario") {
  alert(`Hola ${nombre}`)
}

$evento.addEventListener('click', function () {
  holaMundo()
})
```



Objeto EVENT

El objeto event es un objeto predefinido de JavaScript que nos provee métodos y propiedades para gestionar los eventos.

Descripción aprenderaprogramar.com		
Tipo	Nombre	
Propiedades de control del evento	type	Devuelve el tipo de evento producido, sin el prefijo on (p.ej. click)
	target	Devuelve el elemento del DOM que disparó el evento (inicialmente)
	currentTarget	Devuelve el elemento del DOM que está disparando el evento actualmente (no necesariamente el elemento que disparó el evento, ya que puede ser un disparo debido a burbujeo)
Otras propiedades de control del evento	eventPhase (indica en qué fase de tratamiento de evento estamos, 1 captura, 2 en objetivo, 3 burbujeo), bubbles (booleana, indica si es un evento que burbujea o no), cancelable (booleana, devuelve si el evento viene seguido de una acción predeterminada que puede ser cancelada), cancelBubble (booleana, devuelve si el evento actual se propagará hacia arriba en la jerarquía del DOM o no).	
Propiedad temporal	timeStamp	Devuelve una medida de tiempo en milisegundos desde un origen temporal determinado.
Propiedades de localización del puntero del ratón	clientX, clientY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la ventana del navegador y se expresan en píxeles.
	screenX, screenY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda de la pantalla y se expresan en píxeles.
	pageX, pageY	Devuelven las coordenadas en que se encontraba el puntero del ratón cuando se disparó el evento. Las coordenadas están referidas a la esquina superior izquierda del documento, que pueden ser distintas a las de la ventana si el usuario ha hecho scroll sobre el documento.

Propiedad para detectar el botón del ratón pulsado	button	Normalmente empleado para el evento mouseup (liberación de botón del ratón) para detectar cuál ha sido el botón pulsado. Contiene un valor numérico: 0 para click normal (botón izquierdo), 1 para botón central (botón en el scroll), 2 para botón auxiliar (botón derecho).
Propiedades relacionadas con el teclado	Para determinar qué tecla ha sido pulsada	Lo estudiaremos por separado en la siguiente entrega del curso
Propiedades relacionadas con drag and drop	Algunas no estandarizadas	dataTransfer, dropEffect, effectAllowed, files, types
Otras propiedades varias	Otras (algunas no estandarizadas)	x, y, layerx, layery, offsetX, offsetY, wheelDelta, detail, relatedNode, relatedTarget, view, attrChange, attrName, newValue, prevValue, data, lastEventId, origin, source
Método	stopPropagation()	Detiene la propagación del evento
Método	preventDefault()	Cancela (si es posible) la acción de defecto que debería ocurrir después del evento (equivalente a return false para cancelar la acción).

Como podrán observar son unos cuantos, así que nos concentraremos únicamente en los esenciales, pero ustedes pueden investigar por su cuenta para que sirven los demás.

preventDefault()

Existe un método para detener la propagación de efectos por defecto que poseen algunas etiquetas HTML y que son indeseados. Este se utiliza principalmente para evitar que nuestro sitio se recargue al enviar los datos de un formulario. Veamos un sencillo ejemplo:

```
<!-- index.html -->  
<a href="https://www.google.com/?hl=es">Ir a Google</a>
```



```
// index.js  
const $a = document.querySelector("a")  
  
$a.addEventListener("click", function (e) {  
    alert(`¿A donde crees que vas?`);  
    e.preventDefault();  
})
```

Delegación de eventos

Para evitar declarar muchos `addEventListener()`, algo que podemos hacer es programar una delegación de eventos en un solo `addEventListener()` afectando directamente al `document`.

```
<!-- index.html -->
<section class="una-seccion">
  <h2>Título</h2>
  <article>
    <h3>Subtítulo</h3>
    <p>Lorem, ipsum dolor.</p>
    <a href="https://www.google.com/?hl=es">Ir a Google</a>
  </article>
</section>
```



```
// index.js
document.addEventListener("click", function (e) {
  console.log("Hiciste click en", e.target);

  if (e.target.matches(".una-seccion a")) {
    alert("A dónde crees que vas?");
    e.preventDefault()
  }
});
```

ACADEMY
by NUMEN