



FULL STACK

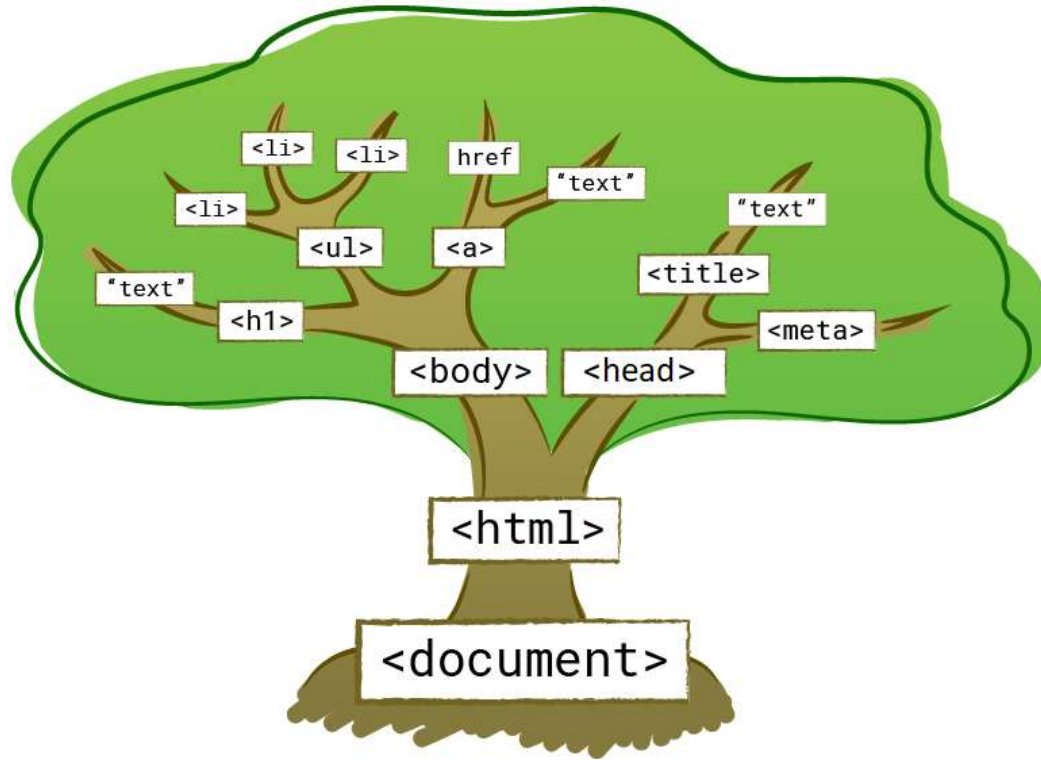
**Comenzamos en unos
minutos**

ACADEMY
by NUMEN

JavaScript: Clase 5

Introducción al DOM

¿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 nombre de etiqueta

El primero de los selectores que veremos es el selector por nombre de etiqueta. Este nos permite conectar Javascript a una o varias etiquetas de un mismo tipo.

```
<!-- index.html -->
<ul>
  <li>Fuego</li>
  <li>Tierra</li>
  <li>Agua</li>
  <li>Aire</li>
</ul>
```



```
console.log(document.getElementsByTagName('li'))
```



```
▼ HTMLCollection(4) [li, li, li, li]
  ► 0: li
  ► 1: li
  ► 2: li
  ► 3: li
    length: 4
  ► [[Prototype]]: HTMLCollection
```


Selectores: Por nombre de clase

Este selector nos permite conectar a Javascript con el DOM a través del valor que le ingresemos al atributo **class** de determinada etiqueta.

```
<!-- index.html -->
<ul>
  <li class="item">Fuego</li>
  <li class="item">Tierra</li>
  <li class="item">Agua</li>
  <li class="item">Aire</li>
</ul>
```

```
console.log(document.getElementsByClassName("item"))
```

```
HTMLCollection(4) [li.item, li.item,
li.item, li.item] i
  ▶ 0: li.item
  ▶ 1: li.item
  ▶ 2: li.item
  ▶ 3: li.item
    length: 4
  ▶ [[Prototype]]: HTMLCollection
```

Selectores: Por nombre

Así como podemos conectarnos a través del atributo `class`, otra forma que nos provee Javascript es conectarnos por medio del atributo **name**. Esto es especialmente importante a la hora de manipular formularios.

```
<!-- index.html -->
<form>
  <input type="text" name="name">
  <input type="number" name="phone">
  <input type="email" name="email">
</form>
```

```
console.log(document.getElementsByName("name"))
console.log(document.getElementsByName("phone"))
console.log(document.getElementsByName("email"))
```

```
▶ NodeList [input]
▶ NodeList [input]
▶ NodeList [input]
```

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>  
`
```



Recorriendo el DOM (Traversing)

Javascript posee propiedades que nos permiten recorrer los elementos del DOM. Aquí veremos las más importantes que son `children` y `parentElement`.

```
<section>
  <ul id="estaciones">
    <li>Verano</li>
    <li>Otoño</li>
    <li>Invierno</li>

    <li>Primavera</li>
  </ul>
</section>
```



```
var $estaciones = document.getElementById('estaciones')
```

```
console.log($estaciones.children)
```

```
console.log($estaciones.children[2])
```

```
console.log($estaciones.parentElement)
```



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);
```


Templates (parte 1)

Las etiquetas `<template>` nos permiten agregar contenido HTML que no se renderiza en el DOM.

```
<!-- index.html -->
<section class="cards">
  <template id="template-card">
    <figure class="card">
      <img>
      <figcaption></figcaption>
    </figure>
  </template>
</section>
```



```
// Archivo JS
const $cards = document.querySelector(".cards")
  $template = document.getElementById("template-
card").content,
  $fragment = document.createDocumentFragment(),
```



Templates (parte 2)

```
var cardContent = [  
  {  
    title: "Yoda",  
    img: "http://pm1.narvii.com/6378/71077675a874957c38a660206fe3ca672b1569f4_00.jpg",  
  },  
  {  
    title: "Luke",  
    img: "https://culturawarsie.files.wordpress.com/2018/11/2vlstesb6.jpg?w=816",  
  },  
  {  
    title: "Darth Vader",  
    img: "https://upload.wikimedia.org/wikipedia/commons/5/5b/Darth_Vader_%28starry_background%29.jpg",  
  },  
  {  
    title: "Han Solo",  
    img: "https://pm1.narvii.com/6214/ab62819f0f71b135b6ed497cf9cbd1f2d83d1f58_hq.jpg",  
  }  
]
```



Templates (parte 3)

```
cardContent.forEach((el) => {  
    $template.querySelector("img").setAttribute("src", el.img);  
    $template.querySelector("img").setAttribute("alt", el.title);  
    $template.querySelector("img").setAttribute("width", "150")  
  
    $template.querySelector("figcaption").textContent = el.title;  
  
    let $clone = document.importNode($template, true);  
    $fragment.appendChild($clone);  
});
```



```
$cards.appendChild($fragment);
```

Insertando Nodos

Para insertar nuevos nodos podemos usar el método `appendChild()` como vimos en ejemplos anteriores. Pero si quisiéramos insertar el nodo en la posición previa a la de otro nodo, podemos utilizar `insertBefore()`

```
/*selector*/.insertBefore(/*Referencia al contenedor*/, /*Referencia al nodo hijo */)
```


Reemplazando Nodos

Así como podemos insertar un método también podemos reemplazar uno existente. Para ello disponemos del método `replaceChild()`.

```
/*selector*/.replaceChild(/*Referencia al contenedor*/, /*selector.children[2] */)
```

Eliminando Nodos

Además de insertar y reemplazar, obviamente deberíamos poder eliminar un nodo existente. Para ello Javascript nos brinda el metodo `removeChild()`.

```
/*selector*/.removeChild(/*selector.children*/)
```

Clonando Nodos

Y como no podía faltar, en caso de necesitar replicar un nodo existente podemos utilizar el método `cloneNode()`.

```
/*selector*/.cloneNode(true)
```

Si le pasamos `true` como valor, clona la etiqueta que posee el selector junto con todo el contenido que esta posea, es decir, incluyendo sus elementos y nodos hijos.

Si por el contrario le pasamos `false`, solo clonara la etiqueta que posee el selector, ignorando su contenido.

ACADEMY
by NUMEN