




 arielZarate ...	27 days ago 
..	
 demo	27 days ago
 homework	27 days ago
 README.json	27 days ago
 README.md	27 days ago

 README.md



Hacé click acá para dejar tu feedback sobre esta clase.



Hacé click acá completar el quizz teórico de esta lecture.

# DOM

## Introducción al DOM

En esta Lesson se verán los siguientes temas:

- DOM
- script
- document
- document Selectors
- Element Methods
- Event Handlers

El nombre `DOM` proviene de sus siglas en ingles de 'Document Object Model'. Cuando un navegador carga una página web, toma todo el contenido HTML y genera un modelo para dicho contenido. Utilizando código Javascript podemos acceder y manipular ese modelo ya sea agregando o quitando elementos, cambiando sus atributos y también modificando sus estilos.

## El elemento *script*

---

Es posible inyectar código Javascript dentro de una página HTML utilizando el elemento `script`. Para ello existen dos formas distintas de realizarlo:

1. Insertando la etiqueta `<script>` en el `<head>` de la página HTML:

```
<html>
  <head>
    <script>
      // Acá es donde irá el código Javascript
      alert('Inyectando código Javascript');
    </script>
  </head>
</html>
```

2. Utilizar la etiqueta `<script>` para inyectar código Javascript contenido en un archivo externo en nuestra página HTML:

```
<html>
  <head>
    <script type="text/javascript" src="./index.js" async></script>
  </head>
</html>
```

*Nota: el atributo `type` debe colocarse como "text/javascript" y en el `src` se debe indicar la ubicación del archivo con código Javascript que queremos inyectar. Es posible también incluir la palabra `async` al final de la etiqueta para indicarle al navegador que debe cargar dicho script de forma asincrónica*

# Document

---

Mediante la ejecución de Javascript tenemos la posibilidad de acceder a un objeto global denominado `document` que contiene las propiedades del DOM y métodos de su prototipo que nos permiten acceder a los elementos del DOM y manipularlos.

En el motor de JS que se ejecuta en el browser, el objeto global es `document`, es decir que `this` apunta a `document` cuando lo usamos en el contexto global.

## Selectors

---

Cuando el Browser parsea el documento HTML, crea una estructura de árbol (el DOM), pensemos que el DOM es un modelo de la página en forma de objetos. JavaScript no sabe cómo trabajar con HTML, pero sí con objetos. Por lo tanto, cada elemento html que esté en el dom, podremos usarlo como un objeto, que tendrá sus propiedades y métodos. Dentro de todos los métodos que contiene `document` en su prototipo los más útiles son los **selectores**. Los **selectores** nos permitirán buscar y recuperar un elemento del DOM. (como cuando buscábamos un elemento en un árbol de búsqueda), sólo que ahora el elemento retornado es un objeto JS que representa una entidad HTML.

Los principales 5 selectores son los siguientes:

### getElementsByClassName

`document.getElementsByClassName` se encarga de encontrar elementos en función del nombre de su clase. Devuelve un array conteniendo los objetos que coincidieron con la búsqueda que puede ser iterado. Debemos brindarle como parámetro un string con el nombre de la clase que deseamos buscar. Ejemplo:

```
const divs = document.getElementsByClassName('divClass');
```

*En este ejemplo estamos buscando los elementos que contengan 'divClass' como su clase definida*

### getElementById

`document.getElementById` se encarga de encontrar un único elemento en función de su id, por lo que devolverá dicho elemento. Debemos brindarle como parámetro un string con el id del elemento que deseamos buscar. Ejemplo:

```
const div = document.getElementById('divId');
```

*En este ejemplo estamos buscando el elemento cuyo id es igual a 'divId'*

## querySelector

---

`document.querySelector` es un método que busca los elementos basándose en uno o más selectores CSS. Recordemos que es posible hacer referencia a clases utilizando un `.`, a ids con `#` y a elementos usando el nombre de su etiqueta directamente. Es recomendable utilizar sólo ids con `querySelector` ya que sólo retornará el primer elemento que coincida con el selector indicado. Ejemplo:

```
const div = document.querySelector('.divId');
```

*En este ejemplo obtendremos el primer elemento con la clase 'divId' pero si hay más elementos con dicha clase no los tendrá en cuenta*

## querySelectorAll

`document.querySelectorAll` funciona de la misma forma que `querySelector` pero en vez de devolver únicamente el primer elemento que coincida con el selector devolverá un array con todos los elementos que coincidan it returns an array like object containing all elements that match the selector. Ejemplo:

```
const divs = document.querySelectorAll('.divId');
```

*En este ejemplo obtendremos un array con todos los elementos que contengan la clase 'divId'*

## createElement

En el caso de que queramos crear un elemento para agregarlo al DOM podemos utilizar `document.createElement`. Este método recibe como parámetro un string indicando el tipo de elemento que deseamos crear y devuelve un elemento vacío de dicho tipo. Ejemplo:

```
const newDiv = document.createElement('div');
```

*En este ejemplo estamos creando un nuevo elemento 'div' vacío*

## Element Methods

---

Una vez seleccionado el elemento podemos utilizar distintos métodos y propiedades para modificarlo como por ejemplo cambiar su estilo, cambiar de atributos, agregar/eliminar elementos anidados, agregar/eliminar `event listeners`. Algunos de los métodos más comunes son:

### .innerHTML

Con el método `innerHTML` podemos acceder a la información que se encuentra entre las etiquetas de apertura y cierre de un elemento HTML tanto para simplemente lectura o para su modificación. Ejemplo:

```
const p = document.querySelector('#pId');  
console.log(p.innerHTML) // Va a imprimir el texto dentro del párrafo con el id '  
  
p.innerHTML = 'Nuevo texto'; // Acá estamos modificando el texto del párrafo menc  
  
console.log(p.innerHTML); // Va a imprimir el nuevo texto que le seteamos, es dec
```

## `.[attribute]` y `.setAttribute`

Podemos llamar al método `.setAttribute` para agregar un atributo a un elemento o sobrescribirlo en el caso de que ya se encuentre definido. Otra forma equivalente de realizarlo pero más corta sería llamando a `.[nombre del atributo] = [nuevo valor]`. Ejemplo:

```
const a = document.querySelector('#linkHenry'); // Obtengo el elemento a cuyo id  
  
a.setAttribute('href', 'https://www.soyhenry.com/'); // Seteo el atributo href de  
  
a.href = 'https://www.soyhenry.com/'; // Equivalente al anterior pero más corto
```

## `.style`

Podemos modificar el estilo de un elemento utilizando `.style`. Cabe mencionar que con esto no estamos accediendo al estilo CSS sino que lo que estamos haciendo es agregarle la propiedad `style` dentro de la etiqueta HTML. Ejemplo:

```
const div = document.querySelector('#divId');  
  
div.style.height = '300px'; // Le damos una altura de 300 pixeles al div cuyo id  
div.style.background = 'red'; // Le seteamos el color de fondo en rojo a dicho di
```

## `.className` y `.id`

Podemos utilizar `.className` y `.id` para acceder y modificar las clases o ids de los elementos. Esto es útil cuando ya tenemos definido en los estilos CSS un estilo en particular asociado a una clase o id y queremos simplemente modificando la clase o id del elemento cambiar su estilo sin tener que modificar propiedad por propiedad. Ejemplo:

```
const div = document.querySelector('#divId');

console.log(div.id); // Utilizando ',id' accedemos al nombre de su id, en este ca
div.className = 'nuevaClase'; // Le seteamos la clase 'nuevaClase'
div.id = 'nuevoId'; // Le seteamos el id 'nuevoId'
```

## .appendChild

Es posible agregar elementos directamente al DOM utilizando `.appendChild` sobre el elemento que queremos que sea su padre. Ejemplo:

```
const body = document.querySelector('body');
const newDiv = document.createElement('div'); // Creamos un nuevo elemento div

body.appendChild(newDiv); // Agregamos el div recién creado dentro del body de l
```

## Event Listeners

Un `Event Listener` es una función que se ejecuta luego de que ocurra un determinado evento. Existen diferentes tipos de eventos, entre ellos se encuentran: un click, un desplazamiento del mouse por encima del elemento, el apretado de una tecla, etc. Para ver la lista completa pueden consultar el siguiente [link](#)

### Click

El evento más común es el de 'click' y en particular es el único que posee la propiedad `.onclick` para asignarle una función que será ejecutada al clickear el componente indicado. Ejemplo:

```
const div = document.querySelector('#divId');
div.onclick = function() {
  console.log('clickeado');
};
```

*En este ejemplo lo que estamos haciendo es indicarle que cuando se clickee el div cuyo id es 'divId' se ejecute la función ahí definida que lo único que hará en este caso es escribir por consola "clickeado"*

## addEventListener y otros eventos

`.addEventListener` es un método que recibe como primer parámetro el tipo de evento que va a estar esperando y como segundo parámetro una función callback que es la que va a ejecutarse cuando ocurra dicho evento. Nota: es mejor utilizar `addEventListener` en todos los casos incluyendo los clicks. Ejemplo:

```
const div = document.querySelector('#divId');
div.addEventListener('mouseenter', function() {
  console.log('El mouse entró!');
});
```

*En este ejemplo lo que estamos haciendo es indicarle que cuando el mouse ingrese al div cuyo id es 'divId' se ejecute la función ahí definida que lo único que hará en este caso es escribir por consola "El mouse entró!"*

## Homework

---

Completa la tarea descrita en el archivo [README](#)

---