ThePower business school

# 🇪🇸 DOM JS 👨‍💻🤖

El DOM da una representación del documento como un grupo de nodos y objetos estructurados que tienen propiedades y métodos. En resumen, es la representación de la página web en la memoria del navegador, a la que podemos acceder a través de JavaScript. El DOM es un árbol donde cada nodo es un objeto con todas sus propiedades y métodos que nos permiten modificarlo.

### Enlazar Javascript con HTML

Para poder leer, modificar, eliminar o crear elementos del DOM tenemos que enlazar nuestro fichero html con javascript. Para ello lo hacemos con la etiqueta script dentro de nuestro head.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="name_file.js" defer></script>
    <title>Enlazar JS :)</title>
</head>
```

Usamos el atributo **defer** para la carga de nuestro fichero una vez construido el DOM.

### Acceso a elementos del DOM

Desde nuestro fichero javascript podemos acceder al árbol del DOM generado en nuestro fichero html y recuperar los elementos.

```
// Obtiene un elemento por id
document.getElementById('someid');

// Obtinee una lista con los elementos que tienen esa clase
document.getElementsByClassName('someclass');

// Obtiene una HTMLCollection con los todos los elementos 'li'
document.getElementsByTagName('li');

// Devuelve el primer elemento del documento que cumpla la selección (la notación es como en CSS)
document.querySelector('.someclass');

// Devuelve una lista de elementos que cumplen con la selección (notación como en CSS)
document.querySelectorAll('div.note, div.alert');
```

### Acceso de hijos a padres

A través de la referencia de padre o hijo podemos acceder a un elemento si es que este no tiene la posibilidad de obtención directa.

```
// Obtener los hijos de un elemento
let elem = document.getElementById('someid');
let hijos = elem.childNodes;

// Su nodo padre
let padre = elem.parentNode;
```

### Crear nuevos elementos

En javascript podemos crear nuevos elementos y posteriormente agregarlos de manera dinámica.

```
// Para crear elementos llamamos a createElement con el nombre del elemento
let nuevoH1 = document.createElement('h1');
let nuevoParrafo = document.createElement('p');

// Crear nodos de texto para un elemento
let textoH1 = document.createTextNode('Hola mundo!');
let textoParrafo = document.createTextNode('lorem ipsum...');

// Añadir el texto a los elementos
nuevoH1.appendChild(textoH1);
nuevoParrafo.appendChild(textoParrafo);

// también podemos asignar directamente el valor a la propiedad innerHTML
nuevoH1.innerHTML = textoH1
nuevoParrafo.innerHTML = textoParrafo

// los elementos estarían listos para añadirlos al DOM, ahora mismo solo existen en memoria, pero no serán visibles hasta que no los añadam
```

### Añadir elementos al DOM

Una vez creados podemos agregarlos al árbol del DOM.

```
// seleccionamos un elemento
let cabecera = document.getElementById('cabecera');

// Añadir elementos hijos a un elemento
cabecera.appendChild(nuevoH1);
cabecera.appendChild(nuevoParrafo);

// También podemos añadir elementos ANTES del elemento seleccionado

// Tomamos el padre
let padre = cabecera.parentNode;

// Insertamos el h1 antes de la cabecera
padre.insertBefore(nuevoH1, cabecera);
```

También podemos añadir directamente un trozo de HTML antes o después de un elemento del DOM, supongamos que tenemos estos elementos en la página.

```
<div id='box1'>
  <p>aquí algo de texto</p>
</div>
<div id='box2'>
  <p>otro parrafo bla bla bla</p>
</div>
```

Podemos hacer.

```
let box2 = document.getElementById('box2');
box2.insertAdjacentHTML('beforebegin', '<div><p>un parrafo nuevo.</p></div>');

// beforebegin - El nuevo HTML es insertado justo antes del elemento, a la misma altura (hermano).
// afterbegin - El nuevo HTML se inserta dentro del elemento, antes del primer hijo.
// beforeend - El nuevo HTML se inserta dentro del elemento, después del último hijo.
// afterend - El nuevo HTML es insertado justo después del elemento, a la misma altura (hermano).
```

### Manipular clases

Además de trabajar sobre elementos también podemos atacar a sus atributos de clase.

```
// Tomamos un elemento
var cabecera = document.getElementById('cabecera');

// elimina una clase del elemento
cabecera.classList.remove('foo');

// Añade una clase si no existe
cabecera.classList.add('otra');

// añade o elimina varias clases a la vez
cabecera.classList.add('foo', 'bar');
cabecera.classList.remove('foo', 'bar');

// Si la clase existe la elimina, si no existe, la crea
cabecera.classList.toggle('visible');

// Devuelve true si el elemento contiene esa clase
cabecera.classList.contains('foo');
```

### DOM templates

Además de esto podemos realizar pequeños templates en javascript e ir añadiendo o quitando dichos elementos a nuestro gusto o necesidades.

```
let title = `<h1>Hello</h1>`
```

Algo más complejo o elaborado sería.

```
let name = 'Alberto';
let job = 'Frontend Developer';

// old school
console.log('my name is ' + name + 'and my job is ' + job);
// new wave
console.log(`my name is ${name} and my job is ${job}`);

// Add to HTML
const contentApp = document.querySelector('#nameSelector');
let html = `<ul>
  <li>name: ${name}</li>
  <li>job: ${job}</li>
</ul>`;

contentApp.innerHTML = html;
```