# DOM y su manipulación

## **¿Qué es el DOM?**

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. Estas son algunas funciones que nos permiten acceder y modificar los elmentos del DOM:

## **Acceso a elementos del DOM**

// Obtiene un elemento por id

document.getElementById('someid');

// Obtiene 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');

### **Acceder a hijos/padres de un elemento**

// Obtener los hijos de un elemento

var elem = document.getElementById('someid');

var hijos = elem.childNodes;

// Su nodo padre

var padre = elem.parentNode;

### **Crear nuevos elementos en el DOM**

// Para crear elementos llamamos a createElement con el nombre del elemento

var nuevoH1 = document.createElement('h1');

var nuevoParrafo = document.createElement('p');

// Crear nodos de texto para un elemento

var textoH1 = document.createTextNode('Hola mundo!');

var 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 añadamos a un elemento del DOM

### **Añadir elementos al DOM**

// seleccionamos un elemento

var 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

var 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:

var 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).

### **Añadir/eliminar/modificar Clases**

// 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');

document.querySelector('img'): obtiene el primer elemento indicado por parametro

document.querySelectorAll('img'): obtiene todos los elementos a partir del indicado por parametro

document.querySelector(‘small’).innerHtml = ‘<b>Hola Mundo</b>’ : coloca lo que queremos en el elemento

document.querySelector('#idDelElemento'): obtiene el elemento

document.querySelector('.nombreDeLa Clase'): obtiene el primer elemento que tenga esa clase asignada

document.querySelectorAll('.nombreDeLa Clase'): obtiene todos los elemento que tengan esa clase asignada

Al ejecutar estos comando Javascript recorre todo el html en busca de lo que se está pidiendo por ende hay que evitar si se puede utilizarlos, colocando los valores en una variable:

const titulo = document.querySelectorAll('.titulo');

titulo.innerText = ‘nuevoTexto’

document.createElement

# Patrón Módulo

Es el patron de diseño mas comun que existe en Javascript ya que es compatible con casi todas las versiones. Permite encapsular y proteger el codigo.

Consiste en colocar nuestro codigo js dentro de una funcion anonima autoinvocable. Utilizar ‘use strict’

y lo que pongamos en el return de esta funcion va a ser publico.

# [javascript-minifier.com](https://javascript-minifier.com/)

Para ofuscar y minimizar nuestro codigo. Tambien existen extenciones en VsCode. Nos permite crear un archivo cuya nomenclatura deberia ser por ejemplo: juego-min.js y esto significaria que ya está listo para ser codigo productivo

# [Underscore.js (underscorejs.org)](https://underscorejs.org/)

Libreria js con varias funcionalidades:

# Clases

Viene a reemplazar el prototype y la manera de trabajar con funciones simulando las clases

## Sets, Gets y Propiedades, sets y metodos estaticos

*class* Persona {

    static \_conteo = 0;

    static *get* conteo() {

        return Persona.\_conteo + ' instancias';

    }

    static mensaje() {

        console.log( this.nombre ); // undefined

        console.log('Hola a todos, soy un método stático');

    }

    nombre = '';

    codigo = '';

    frase  = '';

    comida = '';

*constructor*( *nombre* = 'Sin nombre', *codigo* = 'Sin código', *frase* = 'Sin frase') {

        this.nombre = nombre;

        this.codigo = codigo;

        this.frase  = frase;

        Persona.\_conteo++;

    }

*set* setComidaFavorita( *comida* ) {

        this.comida = comida.toUpperCase();

    }

*get* getComidaFavorita() {

        return `La comida favorita de ${ this.nombre } es ${ this.comida }`;

    }

    quienSoy() {

        console.log(`Soy ${ this.nombre } y mi identidad es ${ this.codigo }`);

    }

    miFrase() {

        this.quienSoy();

        console.log(`${ this.codigo} dice: ${ this.frase }`);

    }

}

*const* spiderman = new Persona( 'Peter Parker', 'Spiderman', 'Soy tu amigable vecino Spiderman' );

*const* ironman   = new Persona( 'Tony Stark', 'Ironman', 'Yo soy Ironman' );

// console.log( ironman );

spiderman.miFrase();

// ironman.miFrase();

spiderman.setComidaFavorita = 'El pie de cereza de la tía May';

// spiderman.comida = 'Duende Verde';

// spiderman.nemesis = 'Duende Verde';// es es aceptado por JS y crea la propiedad aunque no esté en su definicion

// console.log( spiderman.getComidaFavorita );

// console.log( spiderman );

// Persona.\_conteo = 2;

console.log('Conteo stático', Persona.\_conteo );

console.log( Persona.conteo );

Persona.mensaje();

Persona.propiedadExterna = 'Hola Mundo';//esto es permitido pero no recomendado

console.log( Persona.propiedadExterna );

console.log( Persona );

Mas en ES6 y en el codigo fuente de JSModerno. Herencia, subclases

# Propiedades Privadas

Al momento de realizar el curso todavia no está globalmente aceptado por eso hay que revisar este link

[JavaScript classes: Private class fields | Can I use... Support tables for HTML5, CSS3, etc](https://caniuse.com/mdn-javascript_classes_private_class_fields)

Todavia no estan aceptados los metodos privados

*class* Rectangulo {

    #area = 0;

*constructor*(*base* = 0, *altura* = 0) {

        this.base   = base;

        this.altura = altura;

        this.#area = base \* altura;

    }

    calgularArea() {

        console.log( this.#area \* 2 );

    }

}

*const* rectangulo = new Rectangulo(10, 15);

// rectangulo.#area  = 100;

rectangulo.calgularArea();

console.log(rectangulo);

# Singleton

Es una instancia unica de una clase de manera global, util para compartir informacion a lo largo de la app, con BD, etc…

*class* Singleton {

    static instancia; // undefined

    nombre = '';

*constructor*( *nombre* = '' ) {

        if ( !!Singleton.instancia ) {

            return Singleton.instancia;

        }

        Singleton.instancia = this;

        this.nombre = nombre;

    }

}

*const* instancia1 = new Singleton('Ironman');

*const* instancia2 = new Singleton('Spiderman');

*const* instancia3 = new Singleton('BlackPanther');

console.log(`Nombre en la instancia1 es: ${ instancia1.nombre }`);

console.log(`Nombre en la instancia2 es: ${ instancia2.nombre }`);

console.log(`Nombre en la instancia3 es: ${ instancia3.nombre }`);

# Multiples Constructores

A diferencia de otros lenguajes JS no lo soporta pero se puede simular creando unmetodo estatico, es util siq queremos crear instancias de diferentes formas:

*class* Persona {

    static porObjeto({ *nombre*, *apellido*, *pais* }) {

        return new Persona( nombre, apellido, pais );

    }

*constructor*( *nombre*, *apellido*, *pais* ) {

        this.nombre   = nombre;

        this.apellido = apellido;

        this.pais     = pais;

    }

    getInfo() {

        console.log(`info: ${ this.nombre }, ${ this.apellido }, ${ this.pais }`);

    }

}

*const* nombre1   = 'Melissa',

      apellido1 = 'Flores',

      pais1     = 'Honduras';

*const* fher = {

    nombre:   'Fernando',

    apellido: 'Herrera',

    pais:     'Costa Rica'

}

*const* persona1 = new Persona( nombre1, apellido1, pais1 );

*const* persona2 = Persona.porObjeto( fher );

persona1.getInfo();

persona2.getInfo();