

***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 08. JAVASCRIPT

DOM



OBJETIVOS DE LA CLASE

- Comprender el DOM, su alcance y su importancia para operar sobre elementos HTML.

GLOSARIO:

Clase 7

Storage o almacenamiento: nos permite almacenar datos de manera local en el navegador, sin necesidad de realizar alguna conexión con el servidor.

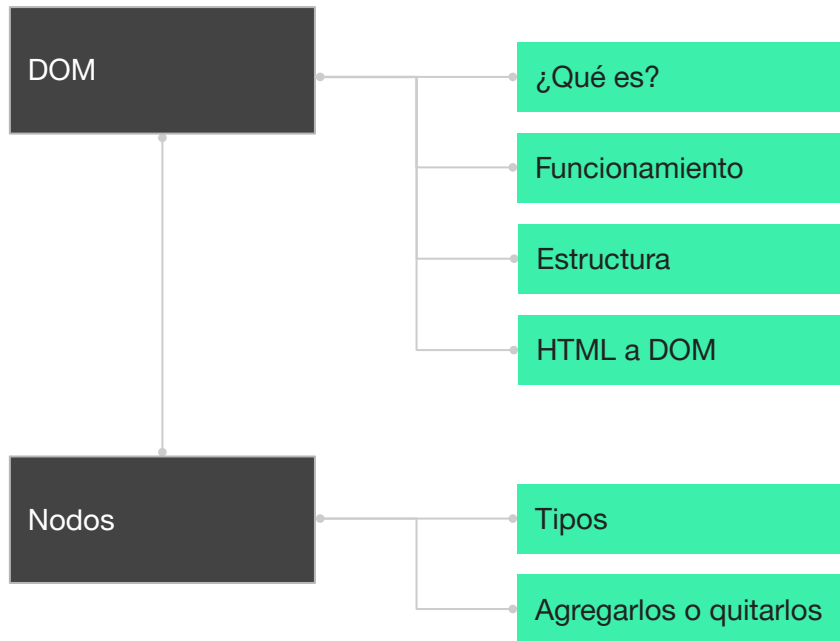
Hay dos tipos de almacenamiento: `localStorage` y `sessionStorage`. Uno es indefinido en el navegador, y otro es temporal, hasta cerrar la pestaña.

JavaScript Object Notation (JSON); es un formato basado en texto plano, para representar datos estructurados en la sintaxis de objetos de JavaScript. Es comúnmente utilizado para transmitir datos en aplicaciones web.

MAPA DE CONCEPTOS

MAPA DE CONCEPTOS CLASE 8

¡Para
recordar!



CRONOGRAMA DEL CURSO

Clase 7



Storage y JSON



EJEMPLOS EN VIVO



EJERCITAR JSON Y
STORAGE

Clase 8



DOM



EJEMPLOS EN VIVO



INTERACTUAR CON HTML

Clase 9



Eventos



EJEMPLOS EN VIVO



INCORPORAR EVENTOS

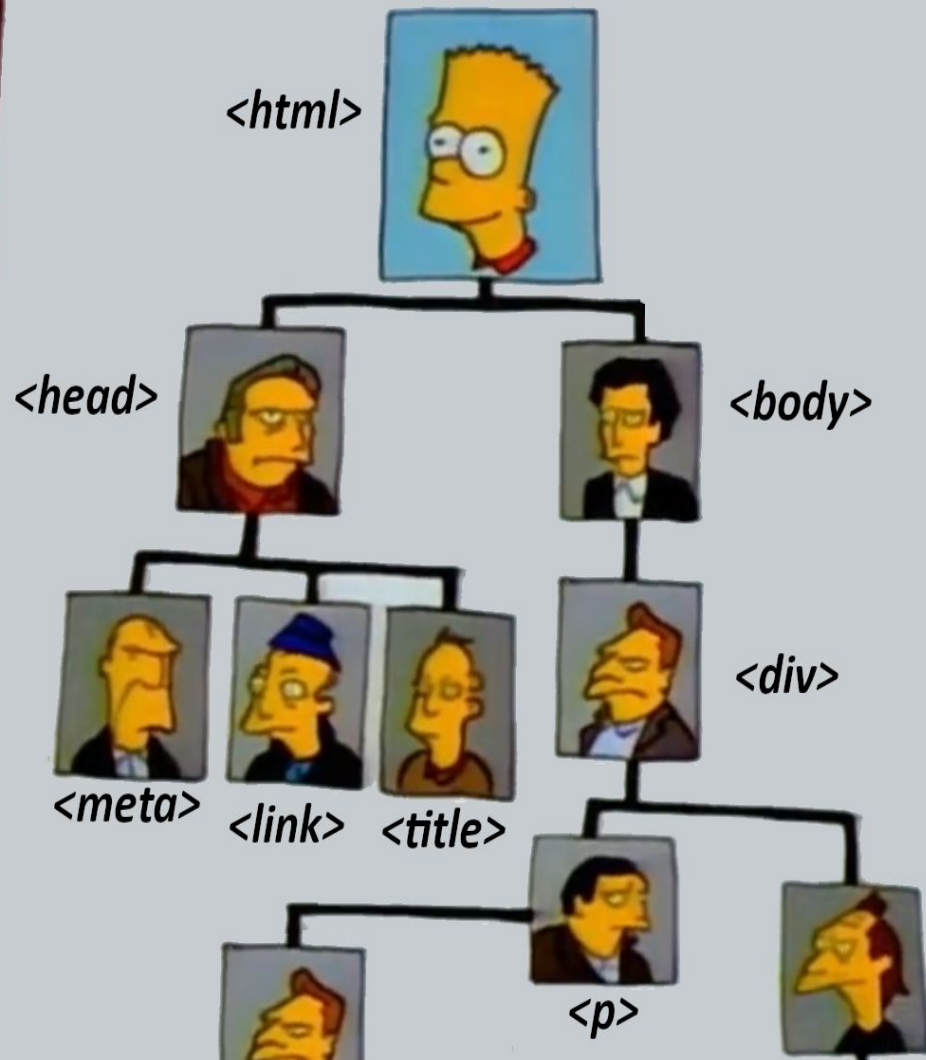


HERRAMIENTAS DE LA CLASE

Les compartimos algunos recursos para acompañar la clase

- Guión de clase N° 8 [aquí](#).
- Quizz de clase N° 8 [aquí](#)
- Booklet de Javascript [aquí](#)
- FAQs de Javascript [aquí](#)

DOM



DOM

Document Object Model

El Modelo de Objetos del Documento (DOM) es una estructura de objetos generada por el navegador, la cual representa la página HTML actual.

Con JavaScript la empleamos para acceder y modificar de forma dinámica elementos de la interfaz.

Es decir que, por ejemplo, desde JS podemos modificar el texto contenido de una etiqueta `<h1>`.

¿CÓMO FUNCIONA?

La estructura de un documento HTML son las etiquetas.

En el Modelo de Objetos del Documento (DOM), cada etiqueta HTML es un objeto, al que podemos llamar nodo. Las etiquetas anidadas son llamadas “nodos hijos” de la etiqueta “nodo padre” que las contiene.

Todos estos objetos son accesibles empleando JavaScript mediante el objeto global `document`

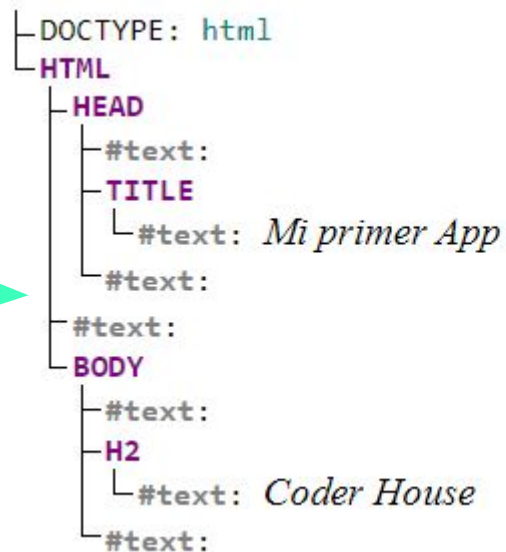
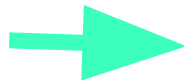
Por ejemplo, `document.body` es el nodo que representa la etiqueta `<body>`

HTML

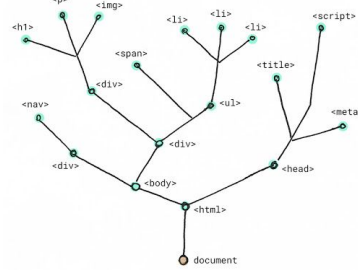
A

DOM

```
<!DOCTYPE html>
<html>
  <head>
    <title>Mi primer App</title>
  </head>
  <body>
    <h2>Coder House</h2>
  </body>
</html>
```



ESTRUCTURA DOM

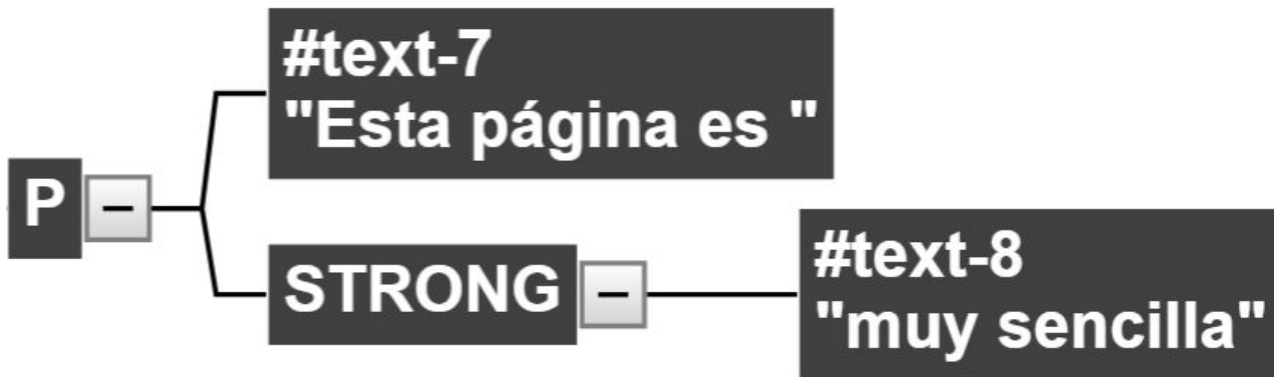


- Cada etiqueta HTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica.
- De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY.
- A partir de esta derivación inicial, cada etiqueta HTML se transforma en un nodo que deriva del correspondiente a su "etiqueta padre".
- La transformación de las etiquetas HTML habituales genera dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta XHTML.

EJEMPLO

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

La etiqueta <p> se transforma en los siguientes nodos del DOM:



EDITAR EL DOM DESDE EL NAVEGADOR

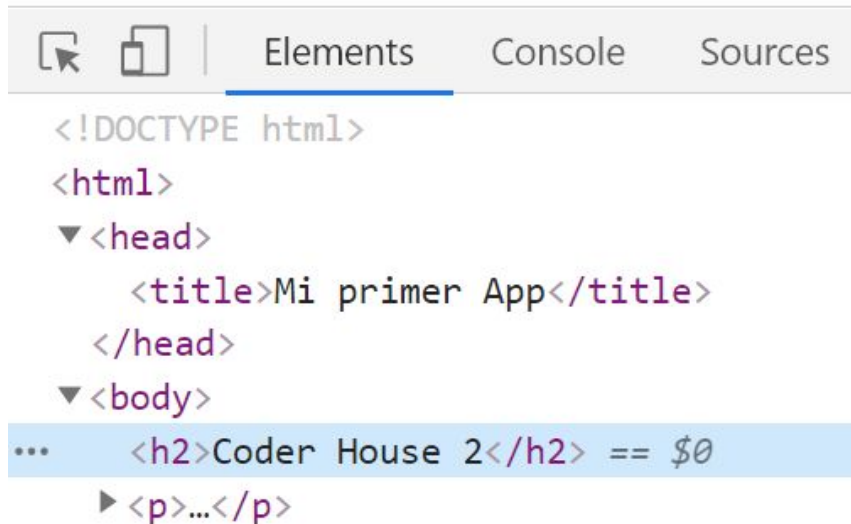
Los navegadores modernos brindan medios para editar el DOM de cualquier página en tiempo real. Por ejemplo en Chrome podemos hacerlo mediante la Herramienta para desarrolladores en la pestaña “Elements”.

Si bien la estructura DOM está simplificada, es un medio muy útil para verificar y probar actualizaciones en la estructura.

Referencia: [Editar el DOM](#) (Chrome)

Coder House 2

Esta página es **muy sencilla**



EJEMPLO APLICADO: ACCESO POR OBJETO document

```
console.dir(document);  
console.dir(document.head)  
console.dir(document.body);
```

El acceso a body usando la referencia document.body requiere que el script se incluya luego de <head> en el HTML

```
<body>  
  <h2>Coder House</h2>  
  <script src="js/main.js"></script>  
</body>
```

TIPOS DE NODOS

La especificación completa de DOM define 12 tipos de nodos, aunque los más usados son:

- Document, nodo raíz del que derivan todos los demás nodos del árbol.
- Element, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- Attr, se define un nodo de este tipo para representar cada uno de los atributos de las etiquetas HTML, es decir, uno por cada par atributo=valor.
- Text, nodo que contiene el texto encerrado por una etiqueta HTML.
- Comment, representa los comentarios incluidos en la página HTML.

Ejemplo
en vivo



¡VAMOS AL CÓDIGO!

CODER HOUSE

ACCESO AL DOM

ACCEDER A LOS NODOS

Existen distintos métodos para acceder a los elementos del DOM empleando en la clase [Document](#). Los más utilizados son:

- getElementById()
- getElementsByClassName()
- getElementsByTagName()

GETELEMENTBYID()

El método getElementById() sirve para acceder a un elemento de la estructura HTML, utilizando su atributo ID como identificación.

```
//CODIGO HTML DE REFERENCIA
```

```
<div id = "app">  
    <p id = "parrafo1" >Hola Mundo</p>  
</div>
```

```
//CODIGO JS
```

```
let div      = document.getElementById("app");  
let parrafo = document.getElementById("parrafo1");  
console.log(div.innerHTML);  
console.log(parrafo.innerHTML);
```

GETELEMENTSBYCLASSNAME()

El método `getElementsByClassName()` sirve para acceder a un conjunto de elementos de la estructura HTML, utilizando su atributo `class` como identificación. Se retornará un Array de elementos con todas las coincidencias:

```
//CODIGO HTML DE REFERENCIA
<ul>
  <li class="países">AR</li>
  <li class="países">CL</li>
  <li class="países">UY</li>
</ul>

//CODIGO JS
let países = document.getElementsByClassName("países");
console.log(países[0].innerHTML);
console.log(países[1].innerHTML);
console.log(países[2].innerHTML);
```

GETELEMENTSBYTAGNAME()

El método `getElementsByTagName()` sirve para acceder a un conjunto de elementos de la estructura HTML, utilizando su nombre de etiqueta como identificación. Esta opción es la menos específica de todas, ya que es muy probable que las etiquetas se repitan en el código HTML.

```
//CODIGO HTML DE REFERENCIA
<div>
  <div>CONTENEDOR 2</div>
  <div>CONTENEDOR 3</div>
</div>

//CODIGO JS
let contenedores = document.getElementsByTagName("div");
console.log(contenedores[0].innerHTML);
console.log(contenedores[1].innerHTML);
console.log(contenedores[2].innerHTML);
```


EJEMPLO APLICADO: RECORRE HTMLCollection CON FOR...OF

```
let paises          = document.getElementsByClassName("paises");  
let contenedores = document.getElementsByTagName("div");  
  
for (const pais of paises) {  
    console.log(pais.innerHTML);  
}  
  
for (const div of contenedores) {  
    console.log(div.innerHTML);  
}
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

**Si adquiriste un servicio de talento,
recordá chequear tu correo de
spam, no deseado, publicidad y/o
social.**

En caso de no haberlo recibido, escribinos a talento@coderhouse.com

¡5/10 MINUTOS Y VOLVEMOS!

CODER HOUSE

AGREGAR O QUITAR NODOS

CREACIÓN DE ELEMENTOS

Para crear elementos se utiliza la función `document.createElement()`, y se debe indicar el nombre de etiqueta HTML que representará ese elemento.

Luego debe agregarse como hijo el nodo creado con `appendChild()`, al `body` u a otro nodo del documento actual.

```
// Crear nodo de tipo Elemento, etiqueta p
let parrafo = document.createElement("p");
// Insertar HTML interno
parrafo.innerHTML = "<h2>¡Hola Coder!</h2>";
// Añadir el nodo Element como hijo de body
document.body.appendChild(parrafo);
```

ELIMINAR ELEMENTOS

Se pueden eliminar nodos existentes y nuevos. El método `removeChild()` permite eliminar nodos hijos a cualquier nodo con tan sólo pasarle las referencias del nodo hijo [a] eliminar y su correspondiente padre:

```
let parrafo      = document.getElementById("parrafo1");  
//Eliminando el propio elemento, referenciando al padre  
parrafo.parentNode.removeChild(parrafo);  
  
let paises       = document.getElementsByClassName("paises");  
//Eliminando el primer elemento de clase paises  
paises[0].parentNode.removeChild(paises[0])
```

OBTENER DATOS DE INPUTS

Para obtener o modificar datos de un formulario HTML desde JS, podemos hacerlo mediante el DOM. Accediendo a la propiedad value de cada input identificado con un ID.

```
//CODIGO HTML DE REFERENCIA
```

```
<input id = "nombre" type="text">
```

```
<input id = "edad" type="number">
```

```
//CODIGO JS
```

```
document.getElementById("nombre").value = "HOMERO";
```

```
document.getElementById("edad").value = 39;
```


EJEMPLO APLICADO: CREANDO OPCIONES DESDE UN ARRAY

```
//Obtenemos el nodo donde vamos a agregar los nuevos elementos
let padre      = document.getElementById("personas");
//Array con la información a agregar
let personas   = ["HOMERO", "MARGE", "BART", "LISA", "MAGGIE"];
//Iteramos el array con for...of
for (const persona of personas) {
    //Creamos un nodo <li> y agregamos al padre en cada ciclo
    let li = document.createElement("li");
    li.innerHTML = persona
    padre.appendChild(li);
}
```

PLANTILLAS DE TEXTO

PLANTILLAS LITERALES

Al momento de incluir valores de las variables en una cadena de caracteres (string) empleábamos la concatenación.

Esta forma puede ser poco legible ante un gran número de referencias.

Un elemento incluido en JS ES6 que solventa esta situación son los templates de literales.

```
let producto = { id: 1, nombre: "Arroz", precio: 125 };  
let concatenado = "ID : " + producto.id + " - Producto: " + producto.nombre + "$ "+producto.precio;  
let plantilla   = `ID: ${producto.id} - Producto ${producto.nombre} $ ${producto.precio}`;  
//El valor es idéntico pero la construcción de la plantilla es más sencilla  
console.log(concatenado);  
console.log(plantilla);
```

PLANTILLAS LITERALES E innerHTML

Las plantillas son un medio para incluir variables en la estructura HTML de nodos nuevos o existentes , modificando el innerHTML

```
let producto    = { id: 1,  nombre: "Arroz", precio: 125 };
let contenedor = document.createElement("div");
//Definimos el innerHTML del elemento con una plantilla de texto
contenedor.innerHTML = `

### ID: ${producto.id}</h3> <p> Producto: ${producto.nombre}</p> <b> $ ${producto.precio}</b>`; //Agregamos el contenedor creado al body document.body.appendChild(contenedor);


```

EJEMPLO APLICADO: CREANDO ELEMENTOS DESDE OBJETOS

```
const productos = [{ id: 1, nombre: "Arroz", precio: 125 },
                    { id: 2, nombre: "Fideo", precio: 70 },
                    { id: 3, nombre: "Pan" , precio: 50},
                    { id: 4, nombre: "Flan" , precio: 100}];

for (const producto of productos) {
  let contenedor = document.createElement("div");
  //Definimos el innerHTML del elemento con una plantilla de texto
  contenedor.innerHTML = `<h3> ID: ${producto.id}</h3>
                          <p> Producto: ${producto.nombre}</p>
                          <b> $ ${producto.precio}</b>`;
  document.body.appendChild(contenedor);
}
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



INTERACTUAR CON HTML

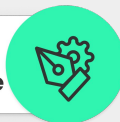
Con lo que vimos sobre DOM, ahora puedes sumarlo a tu proyecto, para interactuar entre los elementos HTML y JS.

INTERACTUAR CON HTML

Formato: Página HTML y código fuente en JavaScript. Debe identificar el apellido del estudiante en el nombre de archivo comprimido por “claseApellido”.

Sugerencia: Generalmente, identificamos a un único elemento del DOM con el atributo id y a un conjunto asociado por class.

Desafío
entregable



>> Consigna: Traslada al proyecto integrador el concepto de objetos, visto en la clase de hoy. En función del tipo de simulador que hayas elegido, deberás:

- Crear elementos manipulando el DOM a partir de la información de tus objetos.
- Modificar etiquetas existentes en función del resultado de operaciones.

>>Aspectos a incluir en el entregable:

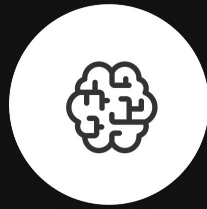
Archivo HTML y Archivo JS, referenciado en el HTML por etiqueta `<script src="js/miarchivo.js"></script>`, que incluya la definición de un algoritmo en JavaScript que opere sobre el DOM, modificando, agregando o eliminando elementos.

>>Ejemplo:

Podemos crear elementos HTML en función del listado de nuestros objetos identificados en la clase 6.

Establecer un mensaje de bienvenida aleatorio usando un array de mensajes.

Capturar una o más entradas por `prompt()` y mostrarlas en el HTML, modificando el DOM



¡PARA PENSAR!

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de zoom
el enlace a un breve quiz de tarea.

Para el profesor:

- *Acceder a la carpeta "Quizzes" de la camada*
 - *Ingresar al formulario de la clase*
 - *Pulsar el botón "Invitar"*
 - *Copiar el enlace*
- *Compartir el enlace a los alumnos a través del chat*

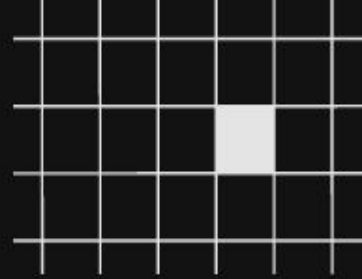
¿PREGUNTAS?



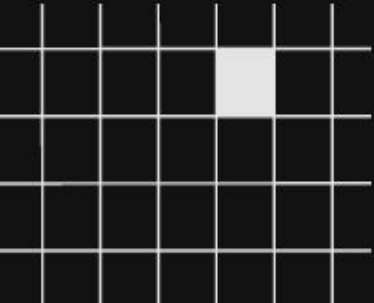
¡DESCUENTO EXCLUSIVO!



[Quiero saber más](#)



¡Completa tu carrera y potencia tu desarrollo profesional!
*Ingresando el cupón **CONTINUATUCARRERA** tendrás un*
descuento para inscribirte en el próximo nivel. Puedes
acceder directamente desde la plataforma, entrando en la
sección ["Cursos y Carreras"](#).





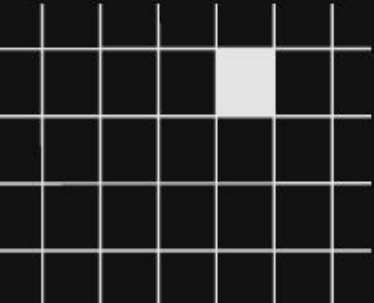
RECURSOS:

- Ejemplos interactivos: DOM |
Árbol del Modelo de Objetos del Documento (DOM)
Recorriendo el DOM
Propiedades de los nodos
- Documentación |
Documentación DOM



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- DOM: definición y uso.
 - Árbol de nodos.
 - Acceder a los elementos, crear y eliminar nodos.
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE