Curso de manipulación del DOM -Platzi.

Video 1. Introducción:

En este curso para manipular el dom aprenderemos los fundamentos de JS. Así podemos dominar cualquier framework de JS.

Veremos como manipular el dom con JS, veremos tres talleres a lo largo del curso, usando JS puro sin framework.

Si usaremos NODEJS.

Profesor: @jonalvarezz
Jonatan Alvarez

Video 2. Que es el DOM.

Lecturas recomendadas:
https://caniuse.com/
https://developer.mozilla.org/es/
https://platzi.com/cursos/web-performance/

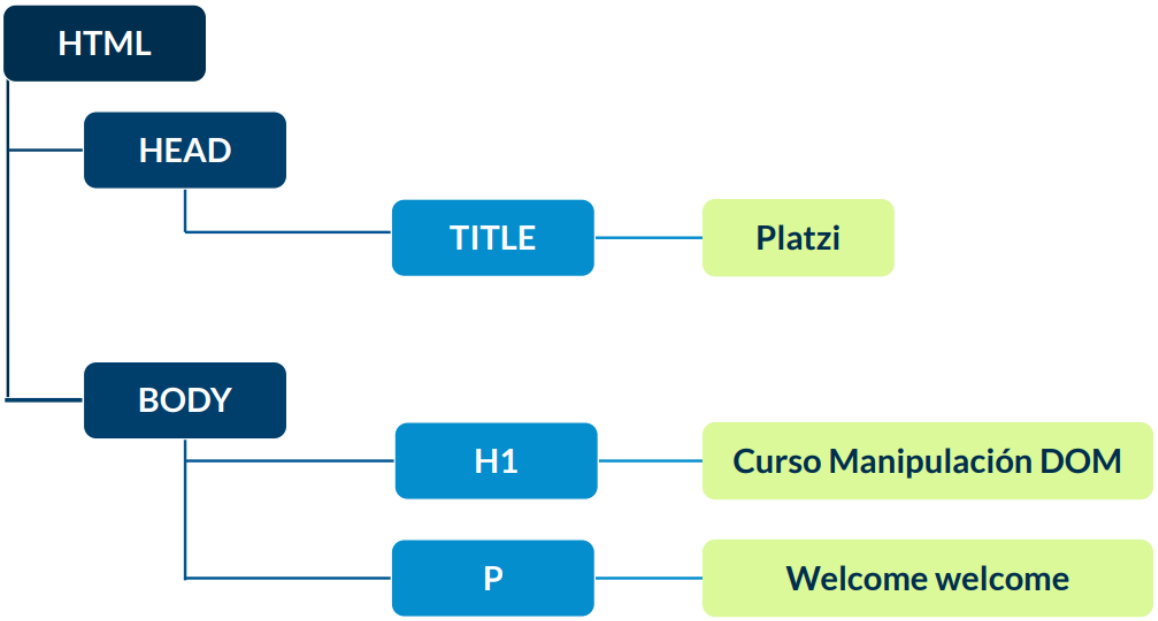
El DOM (Document Object Model en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

El DOM surgió a partir de la implementación de JavaScript en los navegadores. A esta primera versión también se la conoce como DOM 0 o "Legacy DOM". Hoy en día el grupo WHATWG es el encargado de actualizar el estándar de DOM.

Todo empieza un un proceso conocido como el Critical Rendering Path, este proceso es super importante el cual los navegadores hacen para convertir nuestro código en pixeles en las pantallas de los usuarios. Como parte de ese proceso el navegador crea dos arboles básicamente:

El DOM y el CSSOM. Ambos tienen forma de árbol, el primero es el árbol para el HTML y el segundo el árbol para el css.



Este sería la representación gráfica de los nodos para que nosotros las personas los podamos identificar.

En resumen el DOM es:

- Es una representación del HTML
- Estructura de forma de árbol de nodos.
- Es un modelo que puede ser modificado.

Video 3. Web APIs modernas.

Lecturas recomendadas:
https://developer.mozilla.org/es/
https://caniuse.com/

Cuando se combina DOM + JavaScript obtenemos las Web APIs.

Tratando de darle un concepto fácil a lo que significa una API, no es más que un puente, nos permite conectar el DOM con JS para que podamos modificarlo para nuestro beneficio.

Actualmente existen más de 70 Web APIs de las cuales el DOM es solo una de ellas.

La forma como trabajamos las APIs será a través de dos preguntas:

- ¿cómo lo uso? -> developer.mozilla.org
- ¿puedo usarlo? -> caniuse.com

La MDN es la biblia en el desarrollo web, para buscar las dudas si podemos usar algo debemos buscar ahí y si podemos usarlo o no nos dirigimos a caniuse. Si un APIs existe no quiere decir que exista para todos los navegadores exista.

Video 4. Leer nodos.

Las primeras operaciones que haremos será leer nodos. Cómo traer el nodo de ese árbol y cómo leerlo. Ahora vamos a ver los reyes de internet.

```
document.getElementById();
document.getElementsByTagName();
document.getElementsByClassName();
```

Vamos a verlo en el ejemplo: 1_leer_nodos/index.html

console.log(document.getElementById('titulo'));
Devuelve el elemento que lleva esa id.

console.log(document.getElementsByTagName('p'));
Devuelve todos los elementos que tiene ese Tag y los devuelve en un HTMLCollection

console.log(document.getElementsByClassName('parrafo'));
Devuelve todos los elementos que tiene esa clase y los devuelve en un HTMLCollection

Bueno estos tres selectores solo fueron el principio, el tema de la web ha evolucionado muchísimo y han venido más métodos para leer nodos, generalmente en aplicaciones más grandes no es ya recomendado usar estos anteriores metodo.

Es por eso que existen 2 selectores más completos:

```
document.querySelector();
document.querySelectorAll();
```

```
console.log(document.querySelector('.parrafo'));

console.log(document.querySelectorAll('.item'));
```

Con estos podemos seleccionar o leer cualquier nodo que le pasemos en forma de cadena.

document.querySelector()

Si lo usamos nos traerá solo el primer elemento que encuentre con el id o class que le pasemos. Pero solo devuelve el primer elemento que encuentra.

document.querySelectorAll();

Si lo usamos nos traerá un ***nodeList*** con todos los elementos que tengan ese id o class.

Hay un detalle de esto de NodeList, eso que será… Lo veremos en la clase siguiente.

Video 5. NodeList vs Array:

Cuando usamos el document.querySelectorAll() el navegador nos devuelve un NodeList el cual es muy diferente a un array.

OJO PREGUNTA DE FRONTEND IMPORTANTE SABER!!!!!

La principal diferencia es que NodeList carece de operaciones que los arrays si tienen.

Por ejemplo:

```
const nodeList = document.querySelectorAll('.item');

Métodos que si tiene los node list
console.log(nodeList.length);
console.log(nodeList.forEach);

Métodos que no tiene el node list
console.log(nodeList.map);
console.log(nodeList.some);
console.log(nodeList.filter);
console.log(nodeList.reduce);
```

Para poder usar los métodos de los arrays en los nodelist debemos transformar ese nodelist en un array con el **spread operator** usando la nomenclatura de ES6:

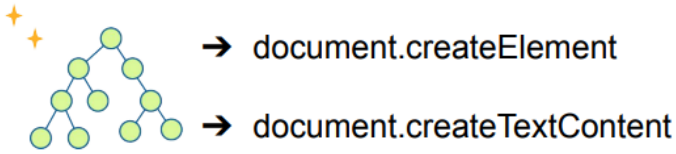
```
const nodeListAsArray  = [...nodeList];
```

Ahora ya podremos usar los métodos de arrays. Como consejo siempre que tengamos un nodeList lo pasamos a array.

Video 6. Crear y agregar Nodos:

Vamos a ver los métodos para crear y agregar nodos.

Crear nodos:



Podemos crear un elemento o un texto. Vamos a ver como crear un elemento:

```
document.createElemento();
```

Ejemplo:

```
document.createElemento('div');
document.createElemento('H1');
```

OJO el hecho de crear un elemento no quiere decir que se ha agregado al DOM. Esta creación sucede en la memoria.

Ahora vamos a ver como crear texto:

```
document.createTextNode("Texto creado");
```

Todo lo que se pase por parámetro será interpretado **como texto** y solo eso.

Ahora para agregar nodos hay muchas formas:

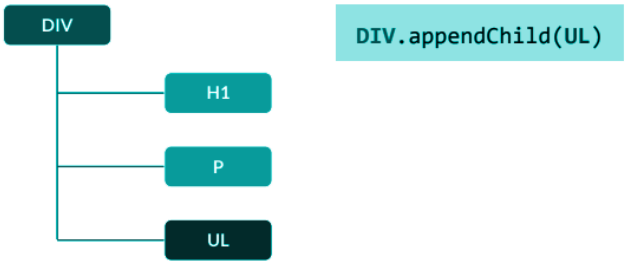
```
nodo.appendChild();
nodo.append();
nodo.insertBefore();
nodo.insertAdjacentElement();
```

Lo podemos ver en el archivo: 2_crear_agregar_nodos/app.js

Para poder agregar nodos necesitamos un nodo de referencia y el nodo que vamos agregar.

nodo.appendChild();
Este metodo siempre nos va agregar el nodo al final.

```
nodo referencia - appendChild
const body = document.querySelector('body');

Creamos los nodo
const container = document.createElement('div');
const h2 = document.createElement('h2');
const text = document.createTextNode("CONTAINER");

h2.appendChild(text);
container.appendChild(h2);

body.appendChild(container);
```

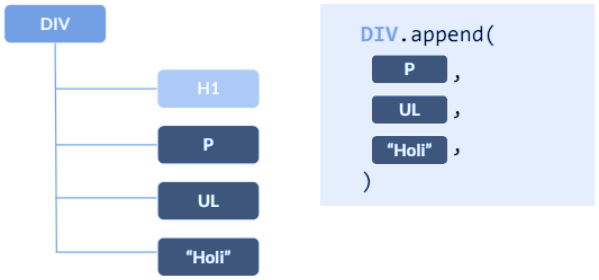


Este metodo es de los primeros que salio y esta un poco limitado, luego ha llegado el metodo:

nodo.append();

- Este metodo es la evolución de appendChild().
- Puedes agregar más de un nodo.
- Puedes agregar texto.
- IE11 no lo soporta.

```
const container2 = document.createElement('div');
const h3 = document.createElement('h3');
const text2 = document.createTextNode("CONTAINER 2");
h3.appendChild(text2);
container2.append(h3);

const mensaje = "Que maravilloso es JS";

body.append( mensaje ,container2);
```



Si tuviéramos que dar soporte a IE11 debemos crear un polyfill, eso lo buscamos en MDN.

nodo.insertBefore();

- Inserta el nodo antes que el nodo que le pasemos como referencia.

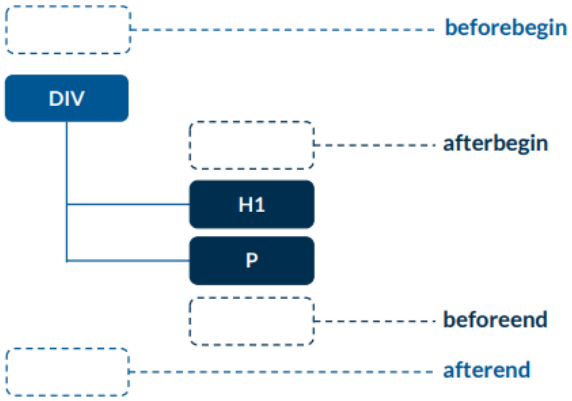

```
metodo insertBefore()
const container3 = document.createElement('div');
const p1 = document.createElement('p');
const text3 = "CONTAINER 3";

p1.append(text3);
container3.append(p1);

body.insertBefore(container3, container);
```

OJO: Para este metodo el nodo referencia tiene que ser hijo directo del padre.

nodo.insertAdjacentElement();

- Es uno de los más complicados para agregar nodos.
- También es uno de los mas poderosos y más flexibilidad da a la hora de agregar nodos.

```
referencia.insertBefore(
  POSICIÓN:
     'beforebegin' | 'afterbegin' | 'beforeend'
     | 'afterend',
  NUEVO_NODO,
)
```




```
metodo insertAdjacentElement()
const container4 = document.createElement('div');
const text4 = "CONTAINER4";

container4.append(text4);

container.insertAdjacentElement('beforebegin', container4);
container.insertAdjacentElement('beforeend', container4);
container.insertAdjacentElement('afterbegin', container4);
container.insertAdjacentElement('afterend', container4);

Este es uno de los más difícil de entender pero si lo analizas
bien verás su potencia.
```

Video 7. Otras formas de agregar:

Existen otras formas de leer y crear cadenas de texto, tenemos los atributos .outerHTML(leer) y .innerHTML(escribir).

Estos dos métodos están muy bien pero si los podemos evitar estaría bien, sobre todo por el tema de seguridad.

Para ver cómo lo hacemos ver la carpeta:
3_outer_inner

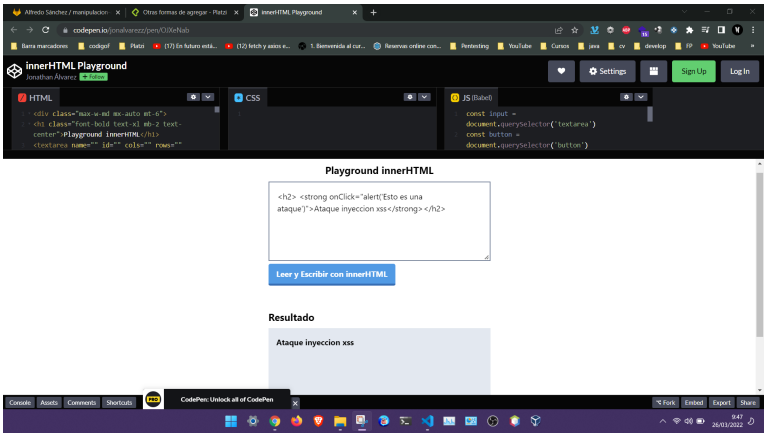
```
// Vamos a ver como leer y modificar elementos con innerHTML y outerHTML

let titulo = document.querySelector('#titulo');

// Leo la variable que viene del dom
console.log(titulo.outerHTML);

// Modifico el valor del elemento
titulo.innerHTML = 'Hola';
console.log(titulo.outerHTML);
```

Esto puede generar problemas de seguridad, lo vemos en el playground que tiene el profesor:

https://codepen.io/jonalvarezz/pen/OJXeNab

Donde le pasa desde el primer cuadro código que será pasado al segundo cuadro, es peligroso porque permite inyección de código xss, código maligno.

Vemos que le pasamos un código y el no tiene reparo de ejecutarlo.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Ejemplo de outer y inner</title>
</head>
<body>
    <h1 id="titulo">outerHTML y innerHTML</h1>
    <h2> <strong onClick="alert('Esto es una ataque')">Ataque
inyeccion xss</strong></h2>
    <script src="./app.js"></script>
</body>
</html>
```

Ver el playground del profesor para entender mejor. Por lo general debemos evitar usarlo cuando la cadena de texto se va a renderizar proviene del usuario. NUNCA pero NUNCA debemos usarlo, lo ideal es pasar por un proceso llamado **sanitizer** para limpiar ese código y revisar que no tenga código maligno y poder usarlo.

Video 8. Atributos y propiedades:

Los atributos y propiedades son los que realmente le dan vida a los elementos que están en el DOM.

El 80% del tiempo que estemos manipulando el dom lo que estamos haciendo es estar cambiando de forma dinámica con JS las propiedades de un elemento para adecuarlos a nuestras necesidades.

Ahora vamos a ver como acceder a todas las propiedades de ese elemento.

Lo podemos ver por la consola, en nuestro html tenemos:

```
    <h1 id="titulo" class="titulo">Atributos y
propiedades</h1>
```

titulo.id
titulo.className

Con la nomenclatura del punto podemos acceder a todos sus atributos y propiedades que tendría ese elemento.

Un o interesante es el .value el cual nos dice el valor de ese elemento:

titulo.value

¿Cuál es la diferencia entre un atributo y una propiedad?

Atributo:
Son usado únicamente al inicio del HTML, son los valores que se le asignan al elemento en el HTML:
<input type='text' class='form-control' id='firstName' placeholder='Escribe tu nombre'>
Los atributos serian type, class, id, placeholder

Las propiedades pueden cambiar en cualquier momento según se modifiquen:

input.value = 'Valor';

La clave está en que atributos está al inicio y propiedades las cosas que cambian en el navegador.

Video 9. Eliminar nodos:

Básicamente hay 3 formas de eliminar nodos:

- removeChild()
- remove()
- replaceChild()

- removeChild()

```
    <div id="padre_1">
        <h2 id="hijo_1">Hola soy un H2</h2>
        <p>Lorem Ipsum is simply dummy text of the printing
and.</p>
    </div>
```

Para utilizar este metodo necesitamos saber que elemento vamos a eliminar y su padre directo.

Para saber cual es el padre de un elemento podemos usar el metodo parentElement:

```
    p.parentElement;

    const padre1 = document.querySelector('#padre_1');
    const hijo1 = document.querySelector('#hijo_1');
    console.log(hijo1.parentElement);
    padre1.removeChild(hijo1);
```

Para verlo de forma gráfica:

```
    <div>
        <p></p>
    </div>

    div.removeChild(p);
```

- remove()

Esta es la evolución de removeChild, y tiene también unas limitantes:
- Es la evolución de removeChild,
- No está soportado por IE.

Este no nos pide ningún padre ni ninguna referencia, remove por dentro usa removeChild.

```
    <h1>Eliminar y reemplazar nodos</h1>
    const referencia = document.querySelector('h1');
```

```
    referencia.remove();
```

- replaceChild()

Para ejecutar este metodo necesitamos el padre, el elemento a reemplazar y el de referencia.

```
    <div id="padre">
        <h3 id="msj">Soy un H3</h3>
    </div>

    const padre = document.querySelector('#padre');
    const h3 = document.querySelector('#msj');
    const h2 = document.createElement('h2');
    h2.textContent = 'Hola yo soy un H2 creado desdes JS';
    padre.replaceChild(h2,h3);
```

Video 10. Operaciones en lote.

El DOM es un API maravilloso que nos permite muchas cosas y manipularlo a nuestro antojo, pero hacer operaciones con el DOM no es gratuito. Sobre todo cuando usamos framework con reactJS, vueJS AngularJS, etc.

La idea es hacer las operaciones mínimas necesarias.

Vamos a ver un tips para eso. Operaciones en lote.

Vamos a agregar 100 input a nuestro dom.

Asi no seria no óptimo:

```
const body = document.querySelector('body');
for(let i = 1; i<=100; i++){
    const nodo = document.createElement('input');
    body.appendChild(nodo);
}
```

Esto no es óptimo ya que por cada ciclo estamos modificando el dom al ejecutar appendChild y requiere un consumo de recursos.

Forma óptima:

```
const nodos = [];
for(let i = 1; i<=100; i++){
    const nodo = document.createElement('input');
    nodos.push(nodo);
}
body.append(...nodos);
```

OJO: los ... Es el spread operator que vino en la versión ES6, de esta forma agregamos de una sola vez en el dom.

https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Spread_syntax

FIN DE LA TEORÍA…