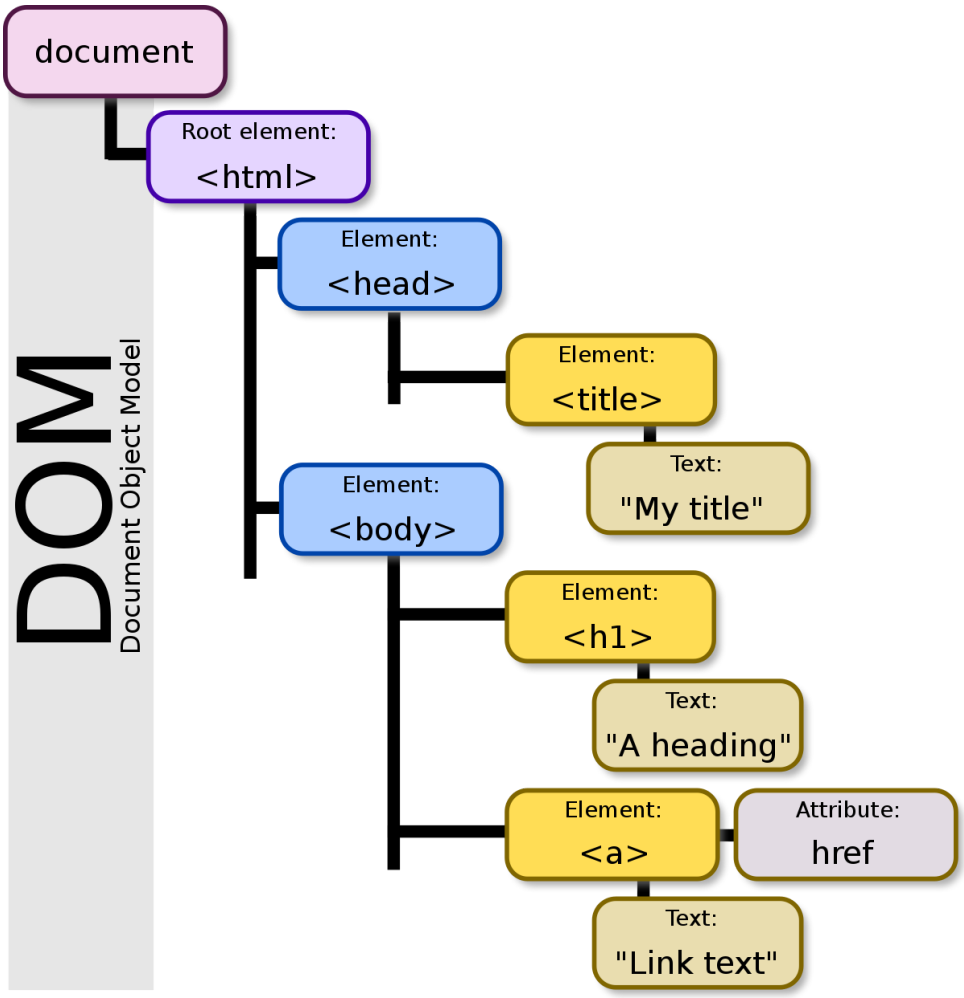
DOM - JavaScript

Introducción al Document Object Model (DOM) en JavaScript

El Document Object Model (DOM) es un concepto fundamental en el mundo del desarrollo web. Se trata de una representación jerárquica de un documento HTML o XML que permite a los desarrolladores web interactuar con el contenido de una página de manera dinámica. En el contexto de JavaScript, el DOM es una pieza clave que posibilita la creación de aplicaciones web interactivas y dinámicas.

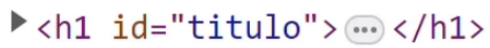
El DOM se estructura como un árbol, donde cada elemento en una página web, como etiquetas HTML, atributos, y texto, se convierte en un nodo en el árbol. Estos nodos pueden ser accedidos y manipulados a través de JavaScript, lo que brinda a los desarrolladores un control completo sobre el contenido y la estructura de una página web. Esto permite realizar acciones como cambiar el texto de un elemento, modificar estilos, agregar o eliminar elementos, y responder a eventos del usuario.

El DOM proporciona una interfaz de programación de aplicaciones (API) que permite a los desarrolladores acceder y modificar los elementos de una página web de manera programática. Esto se logra mediante la creación de objetos JavaScript que representan los nodos del DOM, facilitando la manipulación y la actualización del contenido en tiempo real.

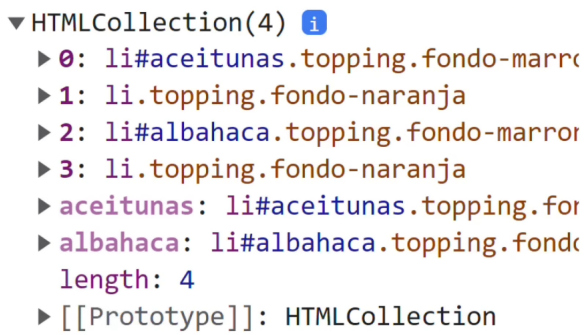


Métodos para el manejo del DOM con JavaScript

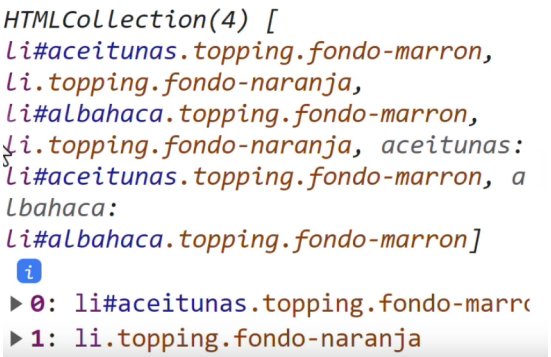
.getElementById("idElemento") → Permite seleccionar un elemento del HTML en base a su id.



.getElementByClassName("classElemento") → Permite seleccionar una lista de elementos HTML que comparten una misma clase CSS.

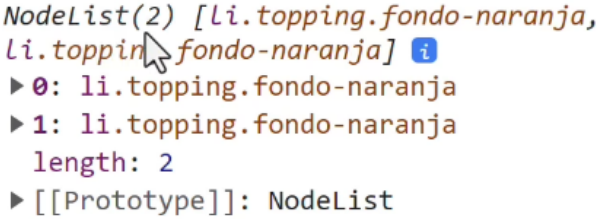


.getElementByTagName("etiquetaHTML") → Permite seleccionar una lista con los elementos HTML que comparten una misma etiqueta HTML.



.querySelector(".nombreClaseCSS/.idCss") → Permite seleccionar un elemento del HTML en base a #identificador o .clase CSS.

.querySelectorAll(".nombreClaseCSS/.idCss") → Permite seleccionar todos los elementos del HTML en base a #identificador o .clase CSS.



En HTML y JavaScript, los términos "nodo" y "elemento" se utilizan comúnmente en el contexto del Document Object Model (DOM), pero hacen referencia a conceptos ligeramente diferentes:

1. **Elemento (Element):**
   - En HTML: Un elemento se refiere a las etiquetas HTML, como `<div>`, `<p>`, `<a>`, etc. Los elementos son la estructura fundamental de un documento HTML y pueden contener contenido, atributos y otros elementos anidados.
   - En JavaScript (DOM): Un elemento del DOM es un tipo de nodo que representa un elemento HTML en la estructura del árbol del DOM. Los elementos tienen propiedades y métodos que permiten a los desarrolladores acceder y modificar las propiedades y el contenido de un elemento HTML. Por ejemplo, puedes acceder al contenido de un elemento, cambiar sus atributos o agregar eventos a él.
2. **Nodo (Node):**
   - En HTML: Un nodo en HTML se refiere a cualquier entidad dentro de la estructura de un documento, que incluye elementos, texto, atributos y más. Los nodos pueden ser elementos, atributos, texto, comentarios, entre otros.
   - En JavaScript (DOM): En el contexto del DOM, un nodo es una entidad que representa una parte del documento web. Los nodos pueden ser elementos, atributos, texto, comentarios, etc. Esto significa que los elementos son un tipo específico de nodo en el DOM. Los nodos forman una jerarquía en un árbol, y los elementos son sólo uno de los tipos de nodos en ese árbol.

Podemos decir pues que todos los elementos son nodos en el contexto del DOM, pero no todos los nodos son elementos. ***Los nodos son una categoría más amplia que incluye elementos, atributos, texto y otros elementos en la estructura de un documento web. Los elementos son nodos específicos que representan etiquetas HTML y son los más comunes en la manipulación del DOM en JavaScript***, ya que son los que normalmente queremos acceder y modificar al construir aplicaciones web dinámicas.

Asignar estilos CSS con JS:

```
const primerTopping = document.querySelector('.toppi

primerTopping.style.backgroundColor = 'blue';
primerTopping.style.color = '#6dff00';
primerTopping.style.textTransform = 'uppercase';
```

Modificar el Texto

```
const titulo = document.getElementById('titulo');

titulo.innerText = 'Mis Toppings Favoritos';
```

Atributos en JavaScript

```
const enlaces = document.getElementsByTagName('a');

console.log(enlaces[0].getAttribute('href'));
```

```
const enlaces = document.getElementsByTagName('a');

console.log(enlaces[0].setAttribute('href', 'https://
```

Clases en JavaScript

```
const primerTopping = document.querySelector('.toppi

primerTopping.classList.add('texto-verde');

console.log(primerTopping.classList);
```

```
const primerTopping = document.querySelector('.topping');

primerTopping.classList.remove('topping');
```

Crear un elemento y añadirlo al DOM en JavaScript

```
const listaDeToppings = document.getElementById('lista-top

const toppingNuevo = document.createElement('li');
toppingNuevo.classList.add('topping', 'fondo-marron');
toppingNuevo.innerText = 'Queso Extra';

listaDeToppings.append(toppingNuevo);
```

Elemento padre en JavaScript (también se puede emplear partentElement y encadenar varios parentElement.parentElement y llegar al abuelo)

```
const listaDeToppings = document.getElementById('lista

console.log(listaDeToppings.parentNode);
```

```
<div id="contenedor"> flex
  <h1 id="titulo">...</h1>
  <ul id="lista-toppings">...</ul>
</div>
```

Elemento hijo en JavaScript

```
const listaDeToppings = document.getElementById('lista

console.log(listaDeToppings.children);
```

```
HTMLCollection(4) [li#aceitunas.topping.fondo-marror
li.topping.fondo-naranja,
li#albahaca.topping.fondo-marron,
li.topping.fondo-naranja, aceitunas:
li#aceitunas.topping.fondo-marron, albahaca:
li#albahaca.topping.fondo-marron] i
▶ 0: li#aceitunas.topping.fondo-marron
▶ 1: li.topping.fondo-naranja
▶ 2: li#albahaca.topping.fondo-marron
▶ 3: li.topping.fondo-naranja
▶ aceitunas: li#aceitunas.topping.fondo-marron
▶ albahaca: li#albahaca.topping.fondo-marron
  length: 4
▶ [[Prototype]]: HTMLCollection
```

También existen el caso de .firstChild() y .lastChild() que trabajan con nodos y .firstElementChild() y .lastElementChild() que trabajan con elementos.

Evento

> “Algo” que ocurre en el sitio web como resultado de interacción con el usuario o por otra causa como cambios en el estado del dispositivo o de la ventana.

```
const albahaca = document.getElementById('albahaca');

function mostrarClic(e) {
  console.log(e);
}

albahaca.addEventListener('click', mostrarClic);
```

```
PointerEvent {isTrusted: true, pointerId: 1, w
idth: 1, height: 1, pressure: 0, …}
  isTrusted: true
  altKey: false
  altitudeAngle: 1.5707963267948966
  azimuthAngle: 0
  bubbles: true
  button: 0
  buttons: 0
```

Por consola vemos la información del objeto PointerEvent que almacena datos sobre el evento en cuestión, como las coordenadas X e Y relativas al lugar de la pantalla donde se produjo el evento, si estaba la tecla ctrl pulsada o el elemento HTML sobre el que se produjo la acción, que está contenido en la variable target como veremos a continuación.

```
const albahaca = document.getElementById('albahaca');

function mostrarClic(e) {
  console.log(e.target);
}

albahaca.addEventListener('click', mostrarClic);
```

```
<li class="topping fondo-marron" id="albahac
a">Albahaca</li> flex
```

También se puede asignar el evento como tal como una función flecha, como se ve en la imagen siguiente:

```
const toppings = document.getElementsByClassName('topping');

for (const topping of toppings) {
  topping.addEventListener('click', (e) => {
    console.log(e.target.innerText);
  });
}
```