

# DOM

¿Qué es?



Estructura que generan todos los navegadores de internet cuando se carga un documento html

Se puede pensar al DOM como una red de nodos. Cada elemento html es un nodo y tiene un padre y puede tener varios hijos.

Documento html que recibe el navegador

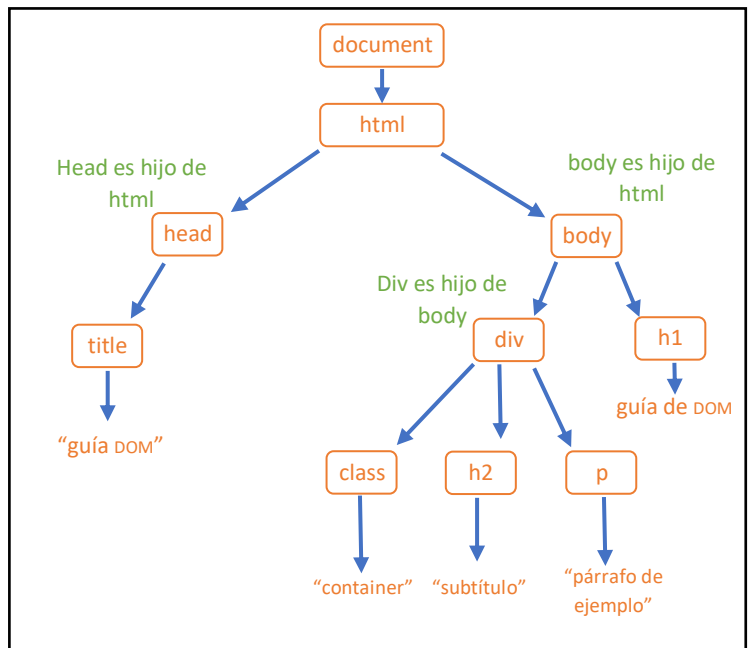
```
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>guía DOM</title>
  </head>

  <body>
    <h1>guía de DOM</h1>

    <div class="container">
      <h2>subtítulo</h2>
      <p>párrafo de ejemplo</p>
    </div>

    
  </body>
</html>
```

Estructura del documento en nodos



**DOM**

Document      Object      Model

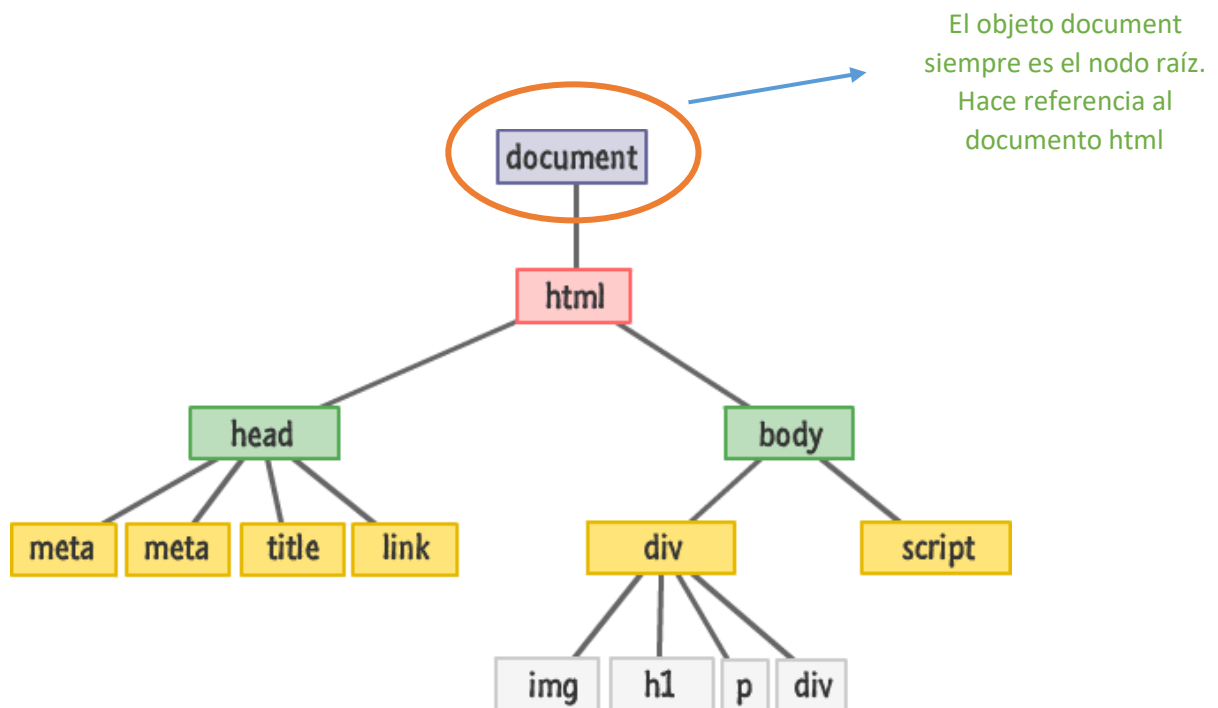
# Manipulación del DOM

Desde javascript podemos manipular el DOM. Pero ¿qué quiere decir esto?

Quiere decir que desde javascript podemos **acceder al código HTML** y modificarlo:

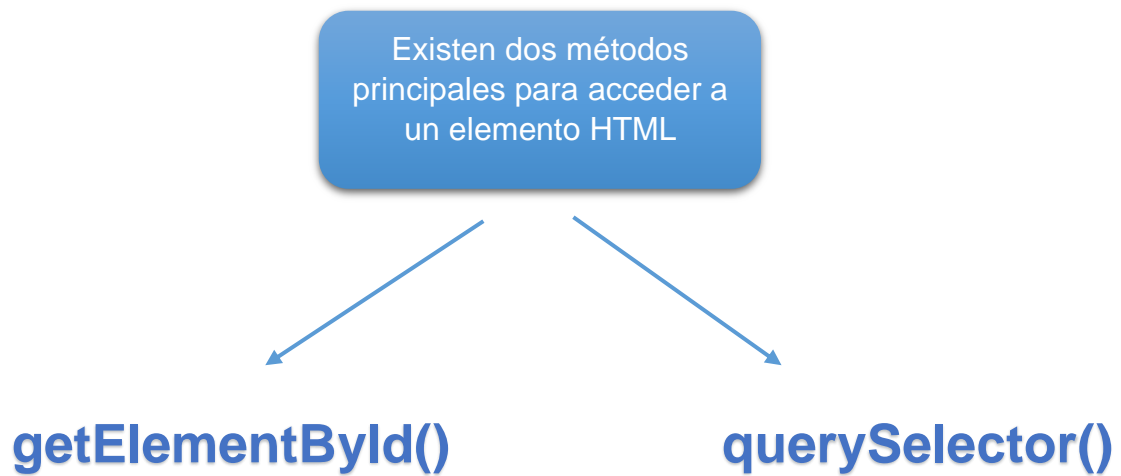
- agregar etiquetas.
- eliminar etiquetas.
- modificar atributos de etiquetas.
- modificar ID y clases.
- obtener el contenido de una etiqueta.

Para acceder al DOM, se debe de hacer desde el nodo raíz, el cual es el **objeto document**.



Entonces si queremos **acceder a un <h1>** (por ejemplo), debemos **meternos** en el objeto document y **usar sus métodos**. Cuando accedemos al h1, podemos ver su contenido (es decir, su texto) o podemos modificarlo, entre otras acciones.

## Métodos para acceder al DOM



Los dos hacen lo mismo. Seleccionan un elemento, como por ejemplo un `<h1>` o un `<p>` o un `<div>` o cualquier elemento html.

## getElementById(id)

Este método retorna el elemento que tenga el id indicado dentro de los paréntesis. Literalmente en español significa obtenerElementoConId.

```
document.getElementById("id")
```

Indicamos el id del elemento

## EJEMPLO



```
<body>
  <h1 id="title">guía de DOM</h1>

  <p id="paragraph">hola</p>
</body>
```

imaginemos que queremos acceder al <h1>



```
const title = document.getElementById("title");
// Lo retorna, por lo tanto, lo guardamos.
// (recordemos que cada elemento es un
// objeto con propiedades y métodos)
console.log(title);
```



```
io Issues
<h1 id="title">guía de DOM</h1>
|
```

Si mostramos en un console.log lo que obtenemos, veremos el elemento

### ¡CUIDADO!

Recordemos que los ID son selectores únicos, es decir, no pueden haber dos elementos con el mismo ID.

Si hay tres <h1> con el mismo id, javascript seleccionará el primer elemento de los tres.

**NUNCA DEBE DE HABER DOS ELEMENTOS CON MISMO ID.**

## querySelector(selector)

Hace lo mismo que `getElementById()` pero la diferencia está en qué a `querySelector()` hay que indicarle el elemento a buscar mediante un selector css (por ejemplo, su class) en forma de string.

## EJEMPLO



```
<body>
  <ul class="product-list">
    <li>carnes</li>
    <li>verduras</li>
    <li>pasta de diente</li>
  </ul>
</body>
```

imaginemos que queremos acceder a esta lista.



Notese que indique la clase con un punto. Si se quiere seleccionar un ID, se debe de usar el selector #.

```
const list = document.querySelector(".product-list");
console.log(list);
```



```
▼ <ul class="product-list">
  ▶ <li>...</li>
  ▶ <li>...</li>
  ▶ <li>...</li>
  </ul>
```

Como se puede ver, accedemos al elemento mediante su clase.

# EJEMPLO CON LOS DOS

JS

```
const title = document.getElementById("title");  
const list = document.querySelector(".product-list");  
const p = document.querySelector("#paragraph");  
  
console.log(title);  
console.log(list);  
console.log(p);
```

Obtengo el <h1> mediante su id

Obtengo el <ul> mediante su clase

Obtengo el <p> mediante su id



```
<h1 id="title">guía de DOM</h1>  
▼ <ul class="product-list">  
  ▶ <li>...</li>  
  ▶ <li>...</li>  
  ▶ <li>...</li>  
</ul>  
  
<p id="paragraph">hola</p>
```

Cuando usar uno y otro: Como primera opción usaremos **getElementById**. Siempre que escribimos código HTML asignamos clases para dar estilos y asignamos ID para que tengan una identificación única al momento de obtenerlos por javascript. **querySelector** generalmente lo usaremos para clases.

¿Qué pasa si quiero seleccionar más de un elemento?

Vimos que los dos métodos obtienen un solo elemento. Obtienen el primer elemento que coincida con el ID o con la clase.

Si queremos obtener más elementos, hay otro método, que es una variante de `querySelector()`, llamada `querySelectorAll()`

## querySelectorAll(selector)

Parece que está claro lo que hace. Obtiene **todos** los elementos que cumplan con el selector indicado. Lo que retorna ya no es un objeto, sino que en realidad es un **array de objetos**. Básicamente retorna un array con todos los elementos obtenidos.

## EJEMPLO



```
<ul id="languages">
  <li class="item">python</li>
  <li class="item">javascript</li>
  <li class="item">c++</li>
  <li class="item">haskell</li>
  <li class="item">lua</li>
</ul>
```

Imaginemos que queremos acceder a todos los <li>

Usaremos querySelectorAll e indicaremos la clase que comparten.



JS

```
const languages = document.querySelectorAll(".item");
console.log(languages);
```



```
▼ NodeList(5) [li.item, li.item, li.item, li.item, li.item] ⓘ
  ▶ 0: li.item
  ▶ 1: li.item
  ▶ 2: li.item
  ▶ 3: li.item
  ▶ 4: li.item
    length: 5
  ▶ __proto__: NodeList
```

Vemos que efectivamente lo que obtenemos es un array de los items. Si se quiere ver los elementos, hay que recorrerlos con un for.

# CUIDADO

5

```
<ul id="languages">
  <li class="item">python</li>
  <li class="item">javascript</li>
  <li class="item">c++</li>
  <li class="item">haskell</li>
  <li class="item">lua</li>
</ul>

<ul id="students">
  <li class="item">Alvaro</li>
  <li class="item">Ignacio</li>
  <li class="item">Sebastian</li>
</ul>
```

Supongamos que hay dos listas cuyos items tienen la misma clase

¿query selector que seleccionará?

¿los items de la lista de lenguajes o los items de la lista de estudiantes?

querySelectorAll selecciona **absolutamente TODOS** los elementos que cumplan con la clase indicada (es decir, agarrará los items de la primer lista y de la segunda)

JS

```
const items = document.querySelectorAll(".item");
items.forEach(item => console.log(item));
```

Muestro cada elemento por un forEach



```
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
▶ <li class="item">...</li>
```

Lista de lenguajes

Lista de estudiantes

Lo ideal en estos casos sería poder diferenciar cada item.



## Diferenciar elementos con misma clase

Si queremos acceder solo a los items de la lista de lenguajes, debemos primero acceder a la lista (es decir, el elemento `<ul>`) y luego acceder a sus hijos.

Recordemos que cada elemento es un objeto con propiedades y métodos. Los métodos `getElementById()` y `querySelector()` pueden ser usados con elementos obtenidos, por ejemplo la lista.

JS

```
//primero obtenemos la lista (el padre de sus items)
const list = document.getElementById("languages");

//Luego obtenemos cada item de esa lista
const items = list.querySelectorAll(".item");

items.forEach(element => console.log(element));
```

Le aplicamos el método `querySelectorAll()` a la lista.

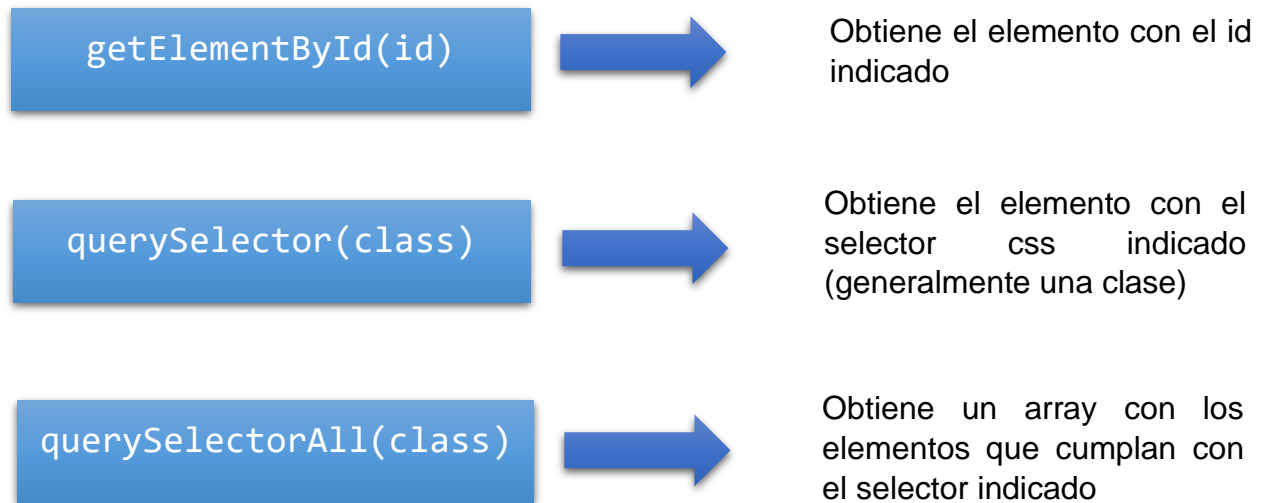
Básicamente nos metemos dentro de la lista y seleccionamos sus hijos que cumplan con la clase



```
▼ <li class="item">
  ::marker
  "python"
</li>
▼ <li class="item">
  ::marker
  "javascript"
</li>
▼ <li class="item">
  ::marker
  "c++"
</li>
▼ <li class="item">
  ::marker
  "haskell"
</li>
▼ <li class="item">
  ::marker
  "lua"
</li>
```

Ahora si obtuvimos solo los elementos que queríamos, que eran los de la lista de lenguajes

## En resumen



De preferencia vamos a usar siempre `getElementById()`, pero para seleccionar elementos por su class usaremos `querySelector()`. Y si queremos seleccionar más de un elemento, usamos `querySelectorAll()`.

## Obtener el contenido de los elementos

Aprendimos a capturar los elementos del documento, pero que pasa si quiero obtener el **texto** de un `<p>` o el de un `<h1>` o cualquier etiqueta que tenga texto.

En ese caso usaremos la propiedad **textContent**.

### Propiedad TextContent

El texto que guarda cada elemento está dentro de la propiedad `textContent`. Recordemos que las **propiedades** son **variables** propias de un objeto (se encuentran dentro de ellos).

```
const elemento = document.getElementById(id);  
let texto = elemento.textContent;
```

Guardamos el texto del elemento en la variable

## EJEMPLO

5

```
<h1 id="title">Hola soy un título</h1>  
<p id="paragraph">Hola soy un párrafo</p>  
<div id="verb">jugar</div>
```

Imaginemos que queremos obtener el texto de estos elementos

JS

```
//primero obtengo los elementos  
const title = document.getElementById("title");  
const paragraph = document.getElementById("paragraph");  
const verb = document.getElementById("verb");  
  
//luego quiero imprimirlos  
console.log("elementos:");  
console.log(title);  
console.log(paragraph);  
console.log(verb);  
  
//luego imprimo los textos que contienen  
console.log("contenido:");  
console.log(title.textContent);  
console.log(paragraph.textContent);  
console.log(verb.textContent);
```

```
<h1 id="title">Hola soy un título</h1>
```

Lo que hacemos es guardar el contenido en la variable

Resultado:

elementos:	
<code>&lt;h1 id="title"&gt;Hola soy un título&lt;/h1&gt;</code>	
<code>&lt;p id="paragraph"&gt;Hola soy un párrafo&lt;/p&gt;</code>	
<code>&lt;div id="verb"&gt;jugar&lt;/div&gt;</code>	
contenido:	
Hola soy un título	Contenido textual
Hola soy un párrafo	
jugar	

## OTRO EJEMPLO

5

```
<div id="container">
  <h1 id="title">Hola soy un título</h1>
  <p id="paragraph">Hola soy un párrafo</p>
  <div id="verb">jugar</div>

  <ul id="languages">
    <li class="item">python</li>
    <li class="item">javascript</li>
    <li class="item">c++</li>
    <li class="item">haskell</li>
    <li class="item">lua</li>
  </ul>

  <div id="article">
    <h1>título</h1>

    <li>uno</li>
    <li>dos</li>
    <li>tres</li>

    <p>asdasdasdasd hola</p>
  </div>
</div>
```

Si quisiera obtener el `textContent` de `container` ¿qué texto me dará?

Lo que contiene el `div container` en su interior son sus elementos hijos.

La propiedad `textContent` agarra los textos del elemento y de sus hijos.

JS

```
const padre = document.getElementById("container");
console.log(padre.textContent);
```

Agarra solo los textos, lo demás lo ignora

resultado

```
Hola soy un título
Hola soy un párrafo
jugar

python
javascript
c++
haskell
lua

título

uno
dos
tres

asdasdasdasd hola
```

Como se puede ver, agarra todos los textos ignorando las etiquetas

## Propiedad innerHTML

Al igual que `textContent`, agarra el contenido de una etiqueta, pero la diferencia está en que `textContent` agarra solo el texto, ignorando las etiquetas. `innerHTML` agarra el texto y las etiquetas de la siguiente manera.

JS

```
const padre = document.getElementById("container");
console.log(padre.innerHTML);
```



```
<h1 id="title">Hola soy un título</h1>
<p id="paragraph">Hola soy un párrafo</p>
<div id="verb">jugar</div>

<ul id="languages">
  <li class="item">python</li>
  <li class="item">javascript</li>
  <li class="item">c++</li>
  <li class="item">haskell</li>
  <li class="item">lua</li>
</ul>

<div id="article">
  <h1>título</h1>

  <li>uno</li>
  <li>dos</li>
  <li>tres</li>

  <p>asdasdasdasd hola</p>
</div>
```

# DIFERENCIAS

textContent



Contenido textual

```
Hola soy un título  
Hola soy un párrafo  
jugar
```

```
python  
javascript  
c++  
haskell  
lua
```

```
título
```

```
uno  
dos  
tres
```

```
asdasdasdasd hola
```

innerHTML



Contenido HTML

```
<h1 id="title">Hola soy un título</h1>  
<p id="paragraph">Hola soy un párrafo</p>  
<div id="verb">jugar</div>  
  
<ul id="languages">  
  <li class="item">python</li>  
  <li class="item">javascript</li>  
  <li class="item">c++</li>  
  <li class="item">haskell</li>  
  <li class="item">lua</li>  
</ul>  
  
<div id="article">  
  <h1>título</h1>  
  
  <li>uno</li>  
  <li>dos</li>  
  <li>tres</li>  
  
  <p>asdasdasdasd hola</p>  
</div>
```

## Manejar atributos de elementos

Aprendimos a obtener elementos, pero imaginemos que después de obtener esos elementos, queremos obtener o modificar atributos de un elemento

Para eso hay dos métodos

**getAttribute()**

**setAttribute()**

Los atributos html son los que están en celeste

```
<input id="input1" type="text" placeholder="escribe algo">
<input id="input2" type="text" value="soy texto">
```

escribe algo	soy texto
--------------	-----------

## getAttribute(attribute)

Retorna el valor de un atributo indicado dentro de los paréntesis en forma de string (o una variable).



```
<input id="input1" type="text" placeholder="escribe algo">
```

Le pedimos el <b>id</b>	→	<b>"input1"</b>
Le pedimos el <b>type</b>	→	<b>"text"</b>
Le pedimos el <b>placeholder</b>	→	<b>"escribe algo"</b>

## EJEMPLO

JS

```
const input1 = document.getElementById("input1");  
  
let id = input1.getAttribute("id");  
let type = input1.getAttribute("type");  
let placeholder = input1.getAttribute("placeholder");  
  
console.log(input1);  
console.log("id: " + id);  
console.log("type: " + type);  
console.log("placeholder: " + placeholder);
```

Primero obtengo el elemento

Luego le pido los atributos que quiero

Los muestro en consola



```
<input id="input1" type="text" placeholder="escribe algo">  
id: input1  
type: text  
placeholder: escribe algo
```



## setAttribute(attribute, value)

No devuelve nada, solamente agarra un elemento y le configura el atributo que queramos indicando dentro de los paréntesis junto al valor que va a tener.

## EJEMPLO



```
<input id="input2" type="text" value="soy texto">
```

Imaginemos que queremos cambiar el type y el value de este input a un botón que diga "click"

JS

```
const input2 = document.getElementById("input2");  
  
input2.setAttribute("type", "button");  
input2.setAttribute("value", "click");  
  
console.log(input2);
```

Acá modifico los atributos de la etiqueta



click

Vemos que lo que antes era un input text, ahora es un button

Lo que era antes

```
<input id="input2" type="text" value="soy texto">
```



Lo que es ahora

```
<input id="input2" type="button" value="click">
```

# EJEMPLO CON LOS DOS



```
<form id="form">
  <input id="txt-name">
  <br><br>
  <input id="txt-apellido">
  <br><br>
  <input id="rng-edad">
  <br><br>
  <input id="btn-ingresar">
</form>
```

Tenemos este html y hay que añadir por código los atributos de cada input y al form.

Por último mostrar los atributos de los elementos.

## Seteo de atributos

JS

```
//atributos del formulario
const formulario = document.getElementById("form");
formulario.setAttribute("action", "#");
formulario.setAttribute("method", "POST");

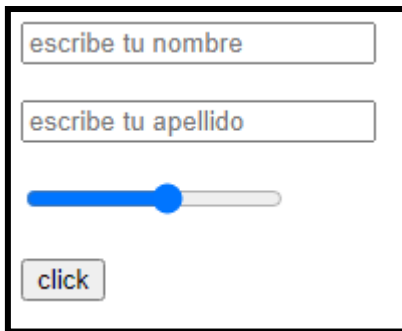
//atributos del input#nombre
const txt_nombre = document.getElementById("txt-nombre");
txt_nombre.setAttribute("type", "text");
txt_nombre.setAttribute("placeholder", "escribe tu nombre");

//atributos del input#apellido
const txt_apellido = document.getElementById("txt-apellido");
txt_apellido.setAttribute("type", "text");
txt_apellido.setAttribute("placeholder", "escribe tu apellido");

//atributos del input#rng-edad
const rng_edad = document.getElementById("rng-edad");
rng_edad.setAttribute("type", "range");
rng_edad.setAttribute("min", "18");
rng_edad.setAttribute("max", "75");

//atributos del input#btn-ingresar
const btn_ingresar = document.getElementById("btn-ingresar");
btn_ingresar.setAttribute("type", "submit");
btn_ingresar.setAttribute("value", "click");
```

## resultado



A screenshot of a web form. It contains two text input fields with placeholder text "escribe tu nombre" and "escribe tu apellido". Below these is a range slider with a blue handle. At the bottom is a button labeled "click".

Como se puede ver, los atributos fueron seteados con éxito.

## Obtención de atributos

JS

```
//getAttributeNames() retorna un array con los atributos que tiene el elemento
//solo retorna la lista de nombres, no de valores
console.log(formulario.getAttributeNames());
console.log(txt_nombre.getAttributeNames());
console.log(txt_apellido.getAttributeNames());
console.log(rng_edad.getAttributeNames());
console.log(btn_ingresar.getAttributeNames());

//muestro los valores de los atributos del input#nombre
txt_nombre.getAttributeNames().forEach(att => {
  //recorro el array y lo uso para obtener el valor del atributo
  console.log(att + " => " + txt_nombre.getAttribute(att));
});
```

Nombre del atributo

Valor indicado por el nombre

```
▶ (3) ["id", "action", "method"]
▶ (3) ["id", "type", "placeholder"]
▶ (3) ["id", "type", "placeholder"]
▶ (4) ["id", "type", "min", "max"]
▶ (3) ["id", "type", "value"]
id => txt-nombre
type => text
placeholder => escribe tu nombre
```

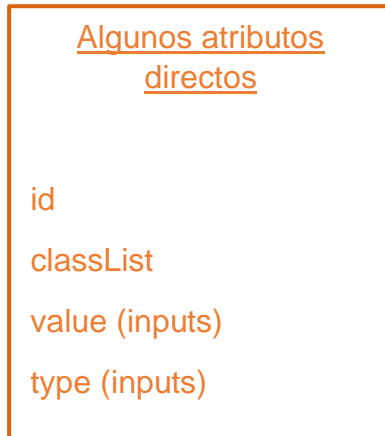
Arrays que retorna  
getAttributeNames()

For que recorre el array de  
atributos de input#nombre

## Atributos directos

Vimos que para obtener atributos y setearlos se usan los dos métodos, get y set.

Pero hay atributos que no hace falta usar esos métodos para acceder a ellos.



Estos atributos de etiquetas HTML pueden ser usados como propiedades del objeto usando la nomenclatura del punto.

### ejemplo

Imaginemos el siguiente  
input button



```
<input id="btn-enviar" type="button" value="enviar" >
```

Si quisieramos el **id** del  
elemento teneos dos  
caminos

**JS**

Usar el método `getAttribute()`

```
console.log(btn_enviar.getAttribute("id"));
```

**JS**

Usar el atributo id directo

```
console.log(btn_enviar.id);
```

### classList

Es la lista de clases que tiene la etiqueta, luego de ver eventos se explicará como trabajar con las clases.

## eventos

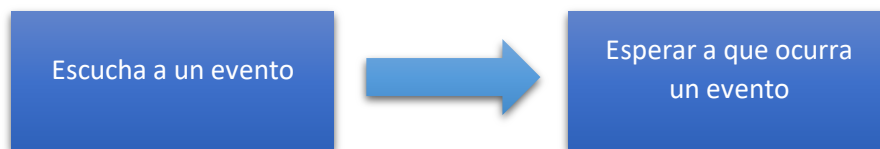
La mayor parte del tiempo en javascript vamos a estar programando eventos.

Esto se hace **capturando un elemento** (como un input de type button) y **ponerlo a la escucha de un evento** “click” y que ejecute un algoritmo **solo** cuando el usuario haga click.

Para ponerlo a la escucha de un evento, usaremos el método `addEventListener()`.

### elem.addEventListener()

El método como bien dice el nombre, agrega una escucha a un evento.



`addEventListener("evento", cb)`

Le indicamos el evento  
que queremos que  
**espere que ocurra**  
Ejemplo “click”

Callback que ejecutará  
cuando ocurra el  
evento indicado  
(cuando se dispare)

Luego de los ejemplos aparecerá una lista  
con los eventos más comunes

# EJEMPLO

5

```
<input id="btn-alert" type="button" value="saludar">
```

enviar datos

Imaginemos que queremos programar la funcionalidad de este botón.

Queremos que cuando el usuario haga un click, muestre un alert saludandolo.

JS

```
//primero se captura el elemento  
const boton = document.getElementById("btn-enviar");  
  
//Luego le agregamos la escucha al evento click y el código que  
ejecuta cuando ocurra el evento  
boton.addEventListener("click", () => alert("datos enviados"));
```

Función que ejecuta cuando el usuario haga click en el botón

**Resultado:** Cuando el usuario ha click en el botón, se dispara el evento "click" y se ejecuta el código asignado.

enviar datos

127.0.0.1:5500 dice  
datos enviados

Aceptar

# OTRO EJEMPLO



```
<input type="text" id="txt-ejemplo" placeholder="escribe algo">  
<div id="salida"></div>
```

escribe algo

Con este ejemplo vamos a ver que podemos poner la cantidad de escuchas que queramos a un elemento.

JS

```
const txt = document.getElementById("txt-ejemplo");  
  
txt.addEventListener("mouseover", () => txt.placeholder="¿vas a escribir algo?");  
txt.addEventListener("mouseleave", () => txt.placeholder="escribe algo");  
  
txt.addEventListener("click", () => {  
  console.log("diste click en el input:text");  
  txt.placeholder="a ver escribe";  
});  
  
txt.addEventListener("keypress", () => {  
  const salida = document.getElementById("salida");  
  
  //uso innerHTML porque textContent no me deja poner saltos de línea  
  salida.innerHTML += "ingresaste un caracter en el input<br>";  
});
```

## resultados

Cuando el mouse no está arriba del elemento

escribe algo

Cuando el mouse está arriba

¿vas a escribir algo?

Cuando le doy click

a ver escribe

hola a

ingresaste un caracter en el input  
ingresaste un caracter en el input  
ingresaste un caracter en el input  
ingresaste un caracter en el input  
ingresaste un caracter en el input  
ingresaste un caracter en el input

Cuando escribo en el input

# EJEMPLO PRACTICO

Imaginemos que hay dos inputs de type number y un botón que dice "sumar"

Programar una página que al apretar el botón, capture los dos números de los inputs y muestre la suma en un alert



```
<input id="num1" type="number">
<br><br>
<input id="num2" type="number">
<br><br>
<input id="btn-sumar" type="button" value="sumar">
```



```
const btn_sumar = document.getElementById("btn-sumar");

btn_sumar.addEventListener("click", () => {
  let num1 = document.getElementById("num1").value;
  let num2 = document.getElementById("num2").value;

  let suma = parseInt(num1) + parseInt(num2);

  alert("resultado: " + suma);
});
```

Los valores ingresados en los inputs se guardan en la propiedad value.

Los valores (al igual que los textContent) son Strings. Por lo tanto, hay que convertirlos a

Resultados:





## Eventos básicos

- **click**: Se dispara cuando el usuario hace click en el elemento que está a la escucha al evento (puede ser un botón, un div o cualquier cosa).
- **dblclick**: Se dispara cuando se hace doble click.
- **mousedown**: se dispara cuando se mantiene apretado el click en un elemento (Útil para Drag and Drop).
- **mouseup**: Se dispara cuando se suelta el mouse (cuando se deja de mantener, útil para Drag and Drop).
- **mousemove**: Se dispara cuando el elemento detecta que el mouse se mueve por encima de él.
- **keydown**: tecla que se mantiene presionada.
- **keyup**: tecla soltada.
- **keypress**: tecla presionada (solo presionada, no se mantiene).

## Objeto event

El callback puede recibir (si lo deseamos) un parámetro que generalmente se le pone de nombre “e” o “event”.

```
elem.addEventListener("click", e => console.log(e))
```



Es opcional darle el objeto al  
callback.

Es un objeto muy importante.

Este objeto contiene **información del evento** que fue disparado

Para ver esa información basta con mostrarla por consola

JS

```
const btn_enviar = document.getElementById("btn-enviar");  
btn_enviar.addEventListener("click", e => console.log(e));
```

Imaginemos que le agrego una escucha al evento click a un input de type button

Imprimo el objeto e para que me muestre información del evento click

```
▼ PointerEvent {isTrusted: true, pointerId: 1,  
  altKey: false  
  altitudeAngle: 1.5707963267948966  
  azimuthAngle: 0  
  bubbles: true  
  button: 0  
  buttons: 0  
  cancelBubble: false  
  cancelable: true  
  clientX: 30  
  clientY: 14  
  composed: true  
  ctrlKey: false  
  currentTarget: null  
  defaultPrevented: false  
  detail: 1  
  ...
```

Vemos que el objeto evento es un PointerEvent.

Nos dice si al hacer el click, mantuvimos apreado la tecla "alt"

Posición del navegador donde se hizo click

Nos dice si al hacer el click, mantuvimos apreado la tecla "ctrl"

```
screenX: 30  
screenY: 117  
shiftKey: false  
▶ sourceCapabilities: InputDeviceCapabilities { ... }  
▶ srcElement: input#btn-enviar  
  tangentialPressure: 0  
▶ target: input#btn-enviar  
  tiltX: 0  
  tiltY: 0  
  timeStamp: 4121.399999976158  
  toElement: null  
  twist: 0  
  type: "click"  
▶ view: Window {window: Window, ...}
```

Posición del monitor donde se hizo click

Nos dice si al hacer el click, mantuvimos apreado la tecla "shift"

Contiene el elemento al cual se le hizo click (contiene al **objeto** que representa a la etiqueta)

Tipo de evento

## event.target

Contiene el elemento HTML que recibió el click.

Si hice click en un input, el event.target contendrá el input.

Si hice click en un div, el event.target contendrá el div

JS

```
const btn_enviar = document.getElementById("btn-enviar");  
btn_enviar.addEventListener("click", e => {  
  console.log(e.target);  
});
```



```
<input id="btn-enviar" type="button" value="enviar">
```

Obtenemos un input, le agregamos la escucha.

En el callback recibo el objeto y muestro la propiedad target

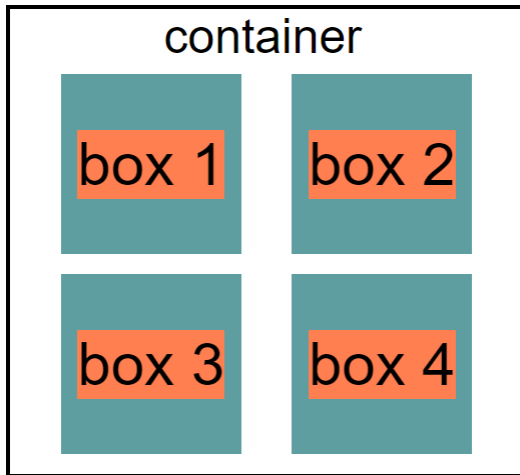
Como vemos, efectivamente contiene al elemento que recibió el click.

Es como si se hubiera hecho un `getElementById()`

### ¿Significa que puedo aplicarle métodos así como antes?

Totalmente. Todo elemento HTML es un objeto con propiedades y métodos y se les puede aplicar `querySelector()`, les puedo agregar escuchas a eventos con `addEventListener()`, puedo acceder a los atributos de la etiqueta con `getAttribute()` y `setAttribute()`, puedo acceder al id, al value, etc.

# EJEMPLO



Imaginemos que obtenemos el container para ver en cual div hijo hice click, mostrando su id por pantalla

5

```
<div id="container" class="container">
  <span class="container-title">container</span>
  <div id="box1" class="box"><span id="span1" class="box-title">box 1</span></div>
  <div id="box2" class="box"><span id="span2" class="box-title">box 2</span></div>
  <div id="box3" class="box"><span id="span2" class="box-title">box 3</span></div>
  <div id="box4" class="box"><span id="span2" class="box-title">box 4</span></div>
</div>
```

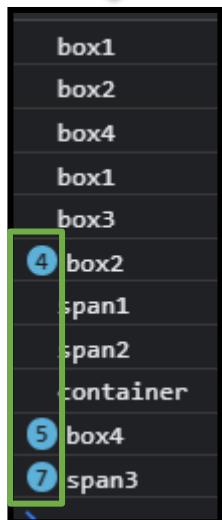
JS

```
// obtengo el container
const container = document.getElementById("container");

//Le agrego la escucha
container.addEventListener("click", e => console.log(e.target.id));
```

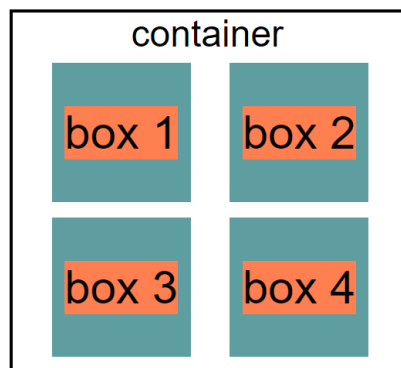
Cuando  
hago click

Que muestre el id del  
elemento que recibió el click



Hay que tener cuidado en donde hacemos click porque el target solo va a agarrar el elemento el cual hicimos click.

Como se ve, también agarra el container o los span ya que son ellos los que reciben el click.



Click en el container (fondo blanco), el target es el container

Click en el div (fondo cadet-blue), el target es el div

Click en el span (fondo coral), el target es el span

Numero de veces en los que se ejecutó el console.log

## event.preventDefault()

Este es un método que lo que hace es **cancelar el comportamiento** que tiene un evento por **defecto**.

El ejemplo que se suele dar es con el `input de type submit` de un formulario, ya que el comportamiento por defecto del evento submit es el de `enviar los datos` a un servidor y `recargar la página`.

## EJEMPLO

Tengo un formulario con un input text y un input submit.



Al hacer submit, se envía lo que escribí al servidor y se recarga la página

### ¿Cómo evitarlo?

```
const form = document.getElementById("form");  
  
form.addEventListener("submit", e => {  
  //cancelo el comportamiento por defecto  
  e.preventDefault();  
  
  alert("datos enviados");  
});
```

Le pongo la escucha al formulario ya que es el mismo formulario el que se encarga de recibir datos, enviarlos al servidor, etc

Basicamente le digo "no envíes los datos ni recargues la página"



## La propiedad classList

¿Qué es?

Es una propiedad del elemento HTML que guarda una lista de todas las clases que tiene la etiqueta.  
(por lo tanto es un array de strings)

## MOSTRAR LAS CLASES

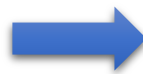


```
<h1 id="titulo" class="clase1 clase2 clase3">titulo</h1>
```

Las clases en html se separan con espacios

JS

```
const titulo = document.getElementById("titulo");  
const clases = titulo.classList;  
console.log(clases);
```



```
DOMTokenList(3) ["clase1", "clase2", "clase3"]  
0: "clase1"  
1: "clase2"  
2: "clase3"  
length: 3  
value: "clase1 clase2 clase3"
```

Accediendo a la lista, podemos:

- agregar clases
- borrar clases
- reemplazar clases

Todo esto lo hacemos mediante métodos muy intuitivos que tiene classList.

# MÉTODOS DE CLASSLIST

`classList.add("clase1", "clase2")`



Añade una o más clases al elemento.  
Si la clase ya existe, se ignora.

`classList.remove("clase1", "clase2")`



Elimina una o más clases.  
Si la clase no existe, se ignora.

`classList.contains("clase1")`



Devuelve true si la etiqueta contiene la clase indicada.

`classList.replace("clase1", "clase2")`



Reemplaza una clase por la otra.  
En este caso, reemplaza la clase 1 por la 2

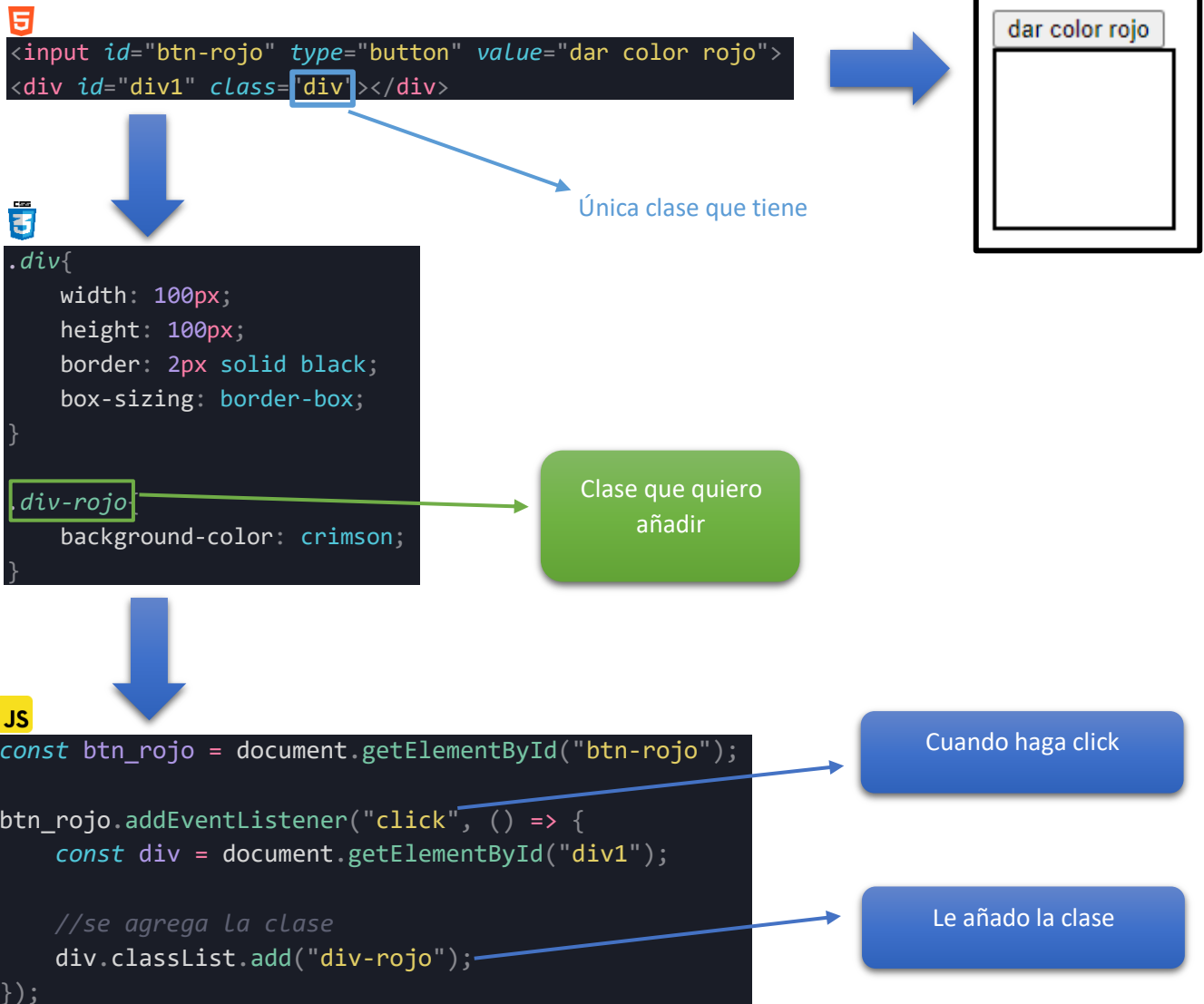
`classList.toggle("clase")`



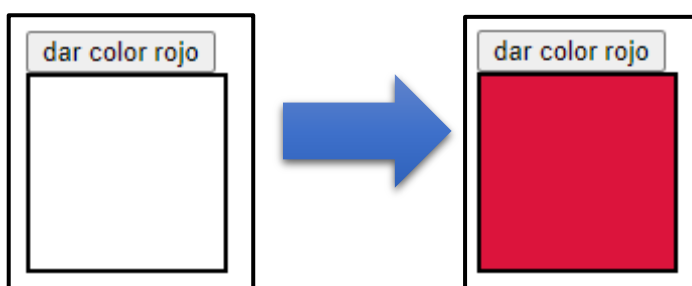
Funciona como interruptor.  
Si tiene la clase, la quita.  
Si no la tiene, la agrega.

## Ejemplo de add()

Imaginemos un botón y un div. El botón cuando se le da click, le agrega una clase css al div (un color rojo de fondo).



Resultado:



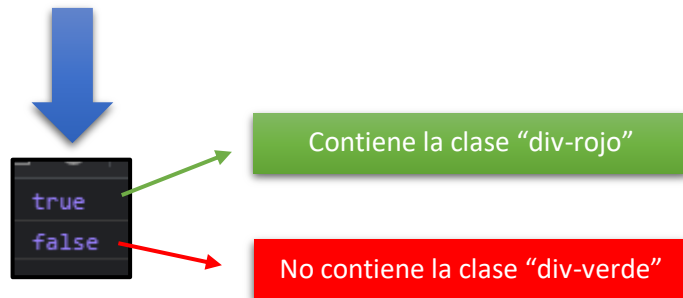


## Ejemplo de contains()

Siguiendo el ejemplo anterior puedo preguntar si el mismo div contiene la clase que ya le agregué (div-rojo) y si tiene otro más (div-verde).

JS

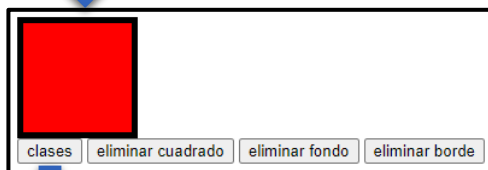
```
console.log(div.classList.contains("div-rojo"));  
console.log(div.classList.contains("div-verde"));
```



## Ejemplo de remove()

JS

```
<div id="box" class="cuadrado fondo-rojo borde-negro"></div>  
  
<input id="btn-mostrarClases" type="button" value="clases">  
<input id="btn-eliminarCuadrado" type="button" value="eliminar cuadrado">  
<input id="btn-eliminarFondo" type="button" value="eliminar fondo">  
<input id="btn-eliminarBorde" type="button" value="eliminar borde">
```



JS

```
btn_mostrarClases.addEventListener("click", () => console.log(box.classList));
```

```
▼ DOMTokenList(3) ["cuadrado", "fondo-rojo", "borde-negro"]  
  0: "cuadrado"  
  1: "fondo-rojo"  
  2: "borde-negro"  
  length: 3  
  value: "cuadrado fondo-rojo borde-negro"  
  ► [[Prototype]]: DOMTokenList
```

Botón que lo único que hace es mostrarnos como queda la classList a medida que se van eliminando clases

JS

```
btn_eliminarCuadrado.addEventListener("click", () => box.classList.remove("cuadrado"));
```

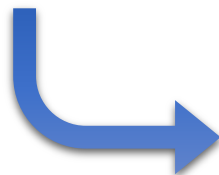


```
▼ DOMTokenList(2) ["fondo-rojo", "borde-negro"]
  0: "fondo-rojo"
  1: "borde-negro"
  length: 2
  value: "fondo-rojo borde-negro"
  ▶ [[Prototype]]: DOMTokenList
```

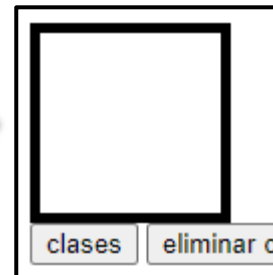
Se eliminó la clase que le da el tamaño de cuadrado

JS

```
btn_eliminarFondo.addEventListener("click", () => box.classList.remove("fondo-rojo"));
```



```
▼ DOMTokenList(2) ["cuadrado", "borde-negro"]
  0: "cuadrado"
  1: "borde-negro"
  length: 2
  value: "cuadrado borde-negro"
  ▶ [[Prototype]]: DOMTokenList
```



JS

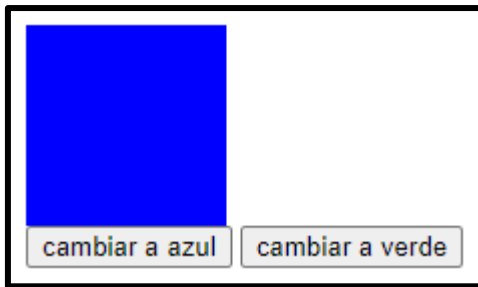
```
btn_eliminarBorde.addEventListener("click", () => box.classList.remove("borde-negro"));
```



```
▼ DOMTokenList(2) ["cuadrado", "fondo-rojo"]
  0: "cuadrado"
  1: "fondo-rojo"
  length: 2
  value: "cuadrado fondo-rojo"
  ▶ [[Prototype]]: DOMTokenList
```



## Ejemplo de replace()



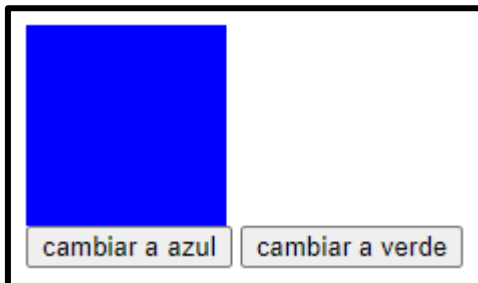
Un div con un color de fondo y dos botones con los que voy a poder cambiar el color de fondo **reemplazando la clase** que da el color de fondo

JS

```
const box = document.getElementById("box");
const btn_azul = document.getElementById("btn-azul");
const btn_verde = document.getElementById("btn-verde");

btn_azul.addEventListener("click", () => box.classList.replace("verde", "azul"));
btn_verde.addEventListener("click", () => box.classList.replace("azul", "verde"));
```

Cambio el azul      Por el verde



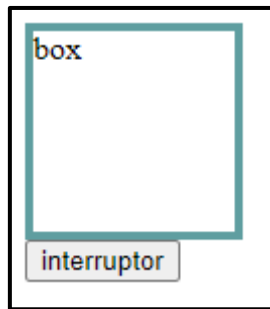
```
body>
<div id="box" class="tem azul"></div>
<input id="btn-azul" type="button" value="cambiar a azul">
<input id="btn-verde" type="button" value="cambiar a verde">
```

Como se puede ver, la clase se reemplaza por otra



```
body>
<div id="box" class="tem verde"></div>
<input id="btn-azul" type="button" value="cambiar a azul">
<input id="btn-verde" type="button" value="cambiar a verde">
```

## Ejemplo de toggle()



Imaginemos que hay un div al que puedo agregarle y quitarle una clase que le cambia el color de fondo a negro y las letras a blanco.

En principio solo tiene una clase que le da tamaño y borde



```
<div id="box" class="tam">box</div>  
<input id="btn-interruptor" type="button" value="interruptor">
```



```
const interruptor = document.getElementById("btn-interruptor");  
  
interruptor.addEventListener("click", () => {  
  const box = document.getElementById("box");  
  
  box.classList.toggle("apagado");  
});
```

Si no tiene la clase, se la agrega.

Si la tiene, se la saca

## Crear elementos HTML

Una forma de agregar elementos en el html era usar la propiedad `innerHTML` para inyectar código dentro del documento.

Otra opción es usar el método `createElement()`

```
const elem = document.createElement("etiqueta");
```

Objeto que representa el elemento html.  
Tiene todas las propiedades y métodos  
que se enseñaron en clases anteriores

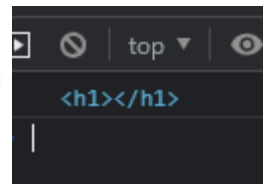
Es como si se hiciera  
`querySelector()`, solo que acá se está  
creando un elemento, no se está  
buscando.

Hay que indicarle la  
etiqueta que queremos  
crear

## Ejemplo

JS

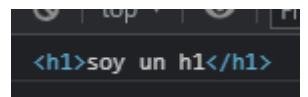
```
const elem = document.createElement("h1");  
console.log(elem);
```



En este ejemplo se ve que se crea  
un h1 vacío. Si se desea agregarle  
contenido se debe de usar la  
propiedad `textContent` como siempre

JS

```
const elem = document.createElement("h1");  
elem.textContent = "soy un h1";  
console.log(elem);
```



## Agregar elemento en el html

Ahora que está creado el elemento que se desea agregar, falta indicarle a javascript donde agregarlo con el método `append()`.

Para esto hay que tener un contenedor preparado para agregarlo dentro y aplicarle el método (puede ser un *div* o un *ul* si se desea agregar un *li*)

## Ejemplo 1

JS

```
// imaginemos que estoy agarrando un div del html
const container = document.querySelector("#div-container");

// le añado la etiqueta h1
const elem = document.createElement("h1");
elem.textContent = "soy un h1";
container.append(elem);
```



Como se ve se  
agrega  
perfectamente al  
html

## Ejemplo 2

Imaginemos un botón que agrega una tarea a una lista de tareas.



```
<input id="btn-agregar" type="button" value="agregar">
<ul id="lista"></ul>
```

JS

```
const lista = document.querySelector("#lista");
let i = 1;

document.querySelector("#btn-agregar").addEventListener("click", () => {
  const tarea = document.createElement("li");
  tarea.textContent = "tarea " + i++;
  lista.append(tarea);
})
```



agregar

- tarea 1
- tarea 2
- tarea 3

Como se puede ver el `append()` actúa igual que el `push()` en los arrays. Agrega el elemento siempre al final

## Añadir más de un elemento de una vez

Para eso se usa un objeto llamado `documentFragment`, que va guardando elementos de forma **iterativa**.

Es un objeto que **simula** ser un elemento html, cuando en realidad no lo es, solo es una **simulación**.

## Ejemplo

Tengo un array con los días de la semana y los quiero agregar a una lista en el html apretando un botón.



JS

```
const btn_agregarDias = document.querySelector("#btn-agregarDias");
const lista = document.querySelector("#lista");

const dias = ["lunes", "martes", "miercoles", "jueves", "viernes"];
```

## Primer código sin fragment

JS

```
btn_agregarDias.addEventListener("click", () => {
  lista.innerHTML = "";

  dias.forEach(dia => {
    const item = document.createElement("li");
    item.textContent = dia;
    lista.append(item);
  })
});
```

Vacía completamente la lista

Recorro el array de días

En cada iteración de día, creo el elemento, le agrego texto y lo añado a la lista del html



El código es correcto, pero no tiene mucho sentido comparado a usar fragment.

## Código con fragment

```
JS const fragment = document.createDocumentFragment();

btn_agregarDias.addEventListener("click", () => {
  lista.innerHTML = "";

  dias.forEach(dia => {
    const item = document.createElement("li");
    item.textContent = dia;
    fragment.append(item);
  });

  lista.append(fragment);
});
```

Se crea el  
documentFragment

En cada iteración agrego el li  
al fragment con append()

Se agrega el fragment a la  
lista

## ¿Qué está pasando?

Sin fragment



Estamos entrando al DOM, agregando un Li, luego saliendo, luego volviendo a entrar para agregar otro, luego saliendo y así sucesivamente.

Es como abrir un guardaropa, guardar una campera, luego cerrarlo, después abrirlo otra vez, guardar una camisa, luego cerrarlo y volverlo a abrir para guardar otra cosa.

**¿Por qué no hacerlo una sola vez?**

Con fragment



Con el fragment agregamos los li a un objeto que simula ser una etiqueta y al finalizar entramos **una única vez** al DOM para agregar lo que está en el fragment

# DOM Traversing

Con DOM traversing se hace referencia a acceder a los hijos de un elemento, a un hijo en específico, a un hermano o a un padre.

## Acceder al padre

`elem.parentNode`



obtenemos al padre del elemento.  
(es como si usamos `querySelector()` para obtenerlo)

```
<ul id="lista">
  <li id="item1">item 1</li>
  <li id="item2">item 2</li>
  <li id="item3">item 3</li>
  <li id="item4">item 4</li>
  <li id="item5">item 5</li>
</ul>
```

JS

Si imprimimos el `parentNode` de un elemento, nos muestra que el `item3` está dentro del `<ul>`

```
const item = document.querySelector("#item3");
console.log(item.parentElement);
```



```
<ul id="lista">...</ul>
```

JS

Es un objeto que representa el elemento html. Por lo tanto, tiene las propiedades y métodos que se aprendió en este pdf.

```
const padre = item.parentElement;
console.log(item, "está dentro de", padre);
```



```
<li id="item3">...</li> 'está dentro de' <ul id="lista">...</ul>
```

## Acceder a los hijos

Para obtener todos los hijos de un elemento se usa la propiedad children, que es una lista de objetos [{},{},{}] que guarda los hijos de un elemento.

La lista es un HTMLCollection.



```
<ul id="lista">
  <li id="item1">item 1</li>
  <li id="item2">item 2</li>
  <li id="item3">item 3</li>
  <li id="item4">item 4</li>
  <li id="item5">item 5</li>
  <li>item 6 sin id</li>
  <li>item 7 sin id</li>
</ul>
```

JS

```
const item = document.querySelector("#lista");
console.log(item.children);
```

```
HTMLCollection(7) [li#item1, li#item2, li#item3, li#item4, li#item5, li, li]
  0: li#item1
  1: li#item2
  2: li#item3
  3: li#item4
  4: li#item5
  5: li
  6: li
  length: 7
  item1: li#item1
  item2: li#item2
  item3: li#item3
  item4: li#item4
  item5: li#item5
  [[Prototype]]: HTMLCollection
```

Todos los hijos

Se puede acceder a un hijo por su posición en el array

Cantidad de hijos

Todos los hijos que tienen id

También se puede acceder por medio de su id

JS

```
console.log("el hijo en posición 2 tiene el id: " + item.children[2].id);
console.log("el hijo en posición 4 tiene el id: " + item.children.item5.id);
```



```
el hijo en posición 2 tiene el id: item3
el hijo en posición 4 tiene el id: item5
>
```

Son objetos por lo que puedo llamar a sus propiedades y métodos

## Acceder al primer hijo

`elem.firstChild`



Contiene al primer hijo del elemento

## ¿Tiene hijos?

`elem.hasChildNodes()`



Retorna true si tiene nodos hijos y false si no los tiene

## Identificar hermanos

`elem.previousElementSibling`



El hermano anterior

`elem.nextElementSibling`



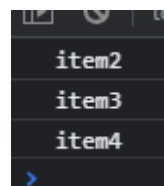
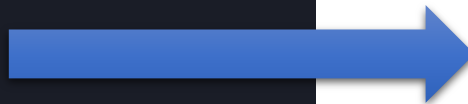
El hermano siguiente

JS

```
// agarro el item3
const item = document.querySelector("#item3");

// agarro a los dos hermanos
const anterior = item.previousElementSibling;
const siguiente = item.nextElementSibling;

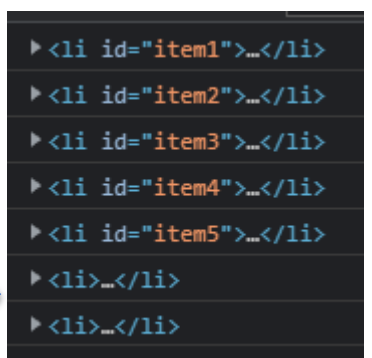
console.log(anterior.id);
console.log(item.id);
console.log(siguiente.id);
```



JS

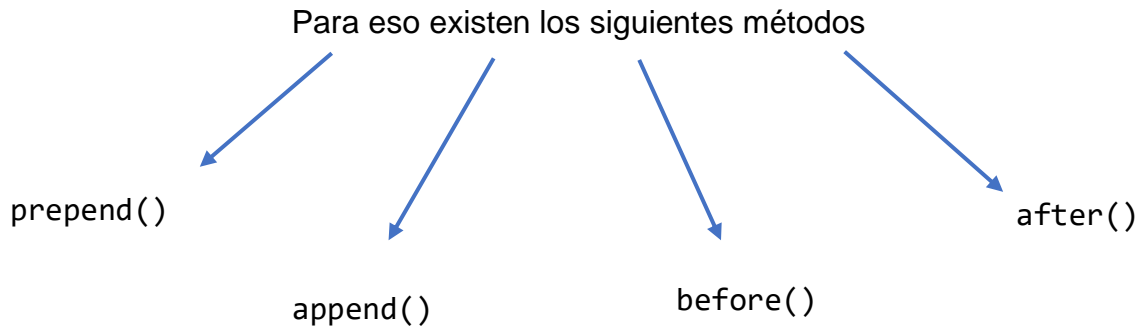
```
// recorro los hermanos hasta el final
let item = document.querySelector("#item1");

do{
  console.log(item)
  item = item.nextElementSibling;
}while(item);
```

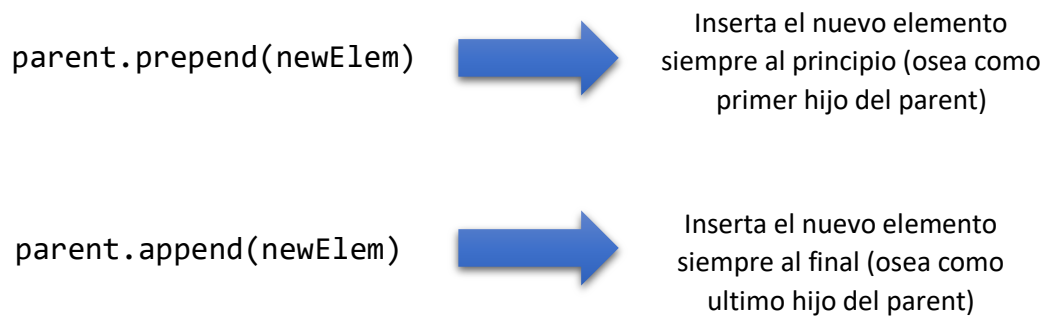


## Insertar hijos

Se había dicho que el `append()` agrega siempre al final. ¿Qué ocurre si en lugar de añadir un elemento al final, se desea insertar **luego** de determinado elemento o **antes**?



### Al principio y al final



## Ejemplo

```
<div id="salida">
  <h1 id="elem1">elemento 1</h1>
  <h1 id="elem2">elemento 2</h1>
  <h1 id="elem3">elemento 3</h1>
</div>
```

**elemento 1**

**elemento 2**

**elemento 3**

Se usaran botónes para ejecutar los distintos métodos

JS

```
var i = 0;

btn-prepend.addEventListener("click", () => {
  const p = document.createElement("p");
  p.textContent = "hola me agrego al inicio " + (++i);

  salida.prepend(p);
});
```



hola me agrego al inicio 3  
hola me agrego al inicio 2  
hola me agrego al inicio 1

**elemento 1**

**elemento 2**

**elemento 3**

JS

```
btn-append.addEventListener("click", () => {
  const p = document.createElement("p");
  p.textContent = "hola me agrego al final";

  salida.append(p);
});
```



hola me agrego al final  
hola me agrego al final  
hola me agrego al final  
hola me agrego al final  
hola me agrego al final

## Antes o despues de un elemento

¿Qué ocurre si quiero agregar al nuevo elemento como hermano de otro?

elem.before(newElem)



Inserta el nuevo elemento antes  
del elemento referencia.

Es decir, como hermano anterior.

elem.after(newElem)



Inserta el nuevo elemento después  
del elemento referencia.

Es decir, como hermano siguiente.

# Ejemplo

JS

```
var anteriorId = 0;

btn_before.addEventListener("click", () => {
  const p = document.createElement("p");
  p.textContent = "soy el hermano anterior " + (++anteriorId);

  const elemento3 = document.querySelector("#elem3");
  elemento3.before(p);
});
```

Le agrego hermanos al elemento3

JS

```
var siguienteId = 0;

btn_after.addEventListener("click", () => {
  const p = document.createElement("p");
  p.textContent = "soy el hermano siguiente " + (++siguienteId);

  const elemento3 = document.querySelector("#elem3");
  elemento3.after(p);
});
```

**elemento 1**

**elemento 2**

soy el hermano anterior 1

soy el hermano anterior 2

**elemento 3**

soy el hermano siguiente 5

soy el hermano siguiente 4

soy el hermano siguiente 3

soy el hermano siguiente 2

soy el hermano siguiente 1

## Reemplazar elemento por otro

elem.replaceWith(newElem)



Elimina al elemento y lo reemplaza por otro nuevo

# Ejemplo

JS

```
btn_replace.addEventListener("click", () => {
  const p = document.createElement("p");
  p.textContent = "hola soy el nuevo";

  // document.querySelector("#elem3").replaceWith(p);
  salida.children[2].replaceWith(p);
});
```

Puedo seleccionar la referencia de distintas maneras, con un querySelector entrando al DOM o con la lista de children

~~elemento 2~~

**elemento 3**



**elemento 2**

hola soy el nuevo

## Eliminar elemento

`elem.remove()`



Selecciono al elemento y le digo que se suicide.

`parent.removeChild(elem)`



Le digo al padre que mate al hijo que le mando por paréntesis

## Ejemplo

JS

```
btn_suicidio.addEventListener("click", () => {  
  // decirle al elemento que se suicide  
  const elem2 = document.querySelector("#elem2");  
  elem2.remove();  
});
```

**elemento 1**  
**elemento 2**  
**elemento 3**



**elemento 1**  
**elemento 3**

JS

```
btn_matar.addEventListener("click", () => {  
  // decirle al padre que mate a su hijo  
  elem2 = document.querySelector("#elem2");  
  salida.removeChild(elem2);  
});
```

Los dos códigos tienen el mismo efecto porque eliminan el mismo elemento 2