Desarrollo Web Frontend

Edición 2021

Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología – Sede Oro Verde

¿Qué es el DOM?

Tipos de nodos

Seleccionar elementos del DOM

Creando elementos en el DOM

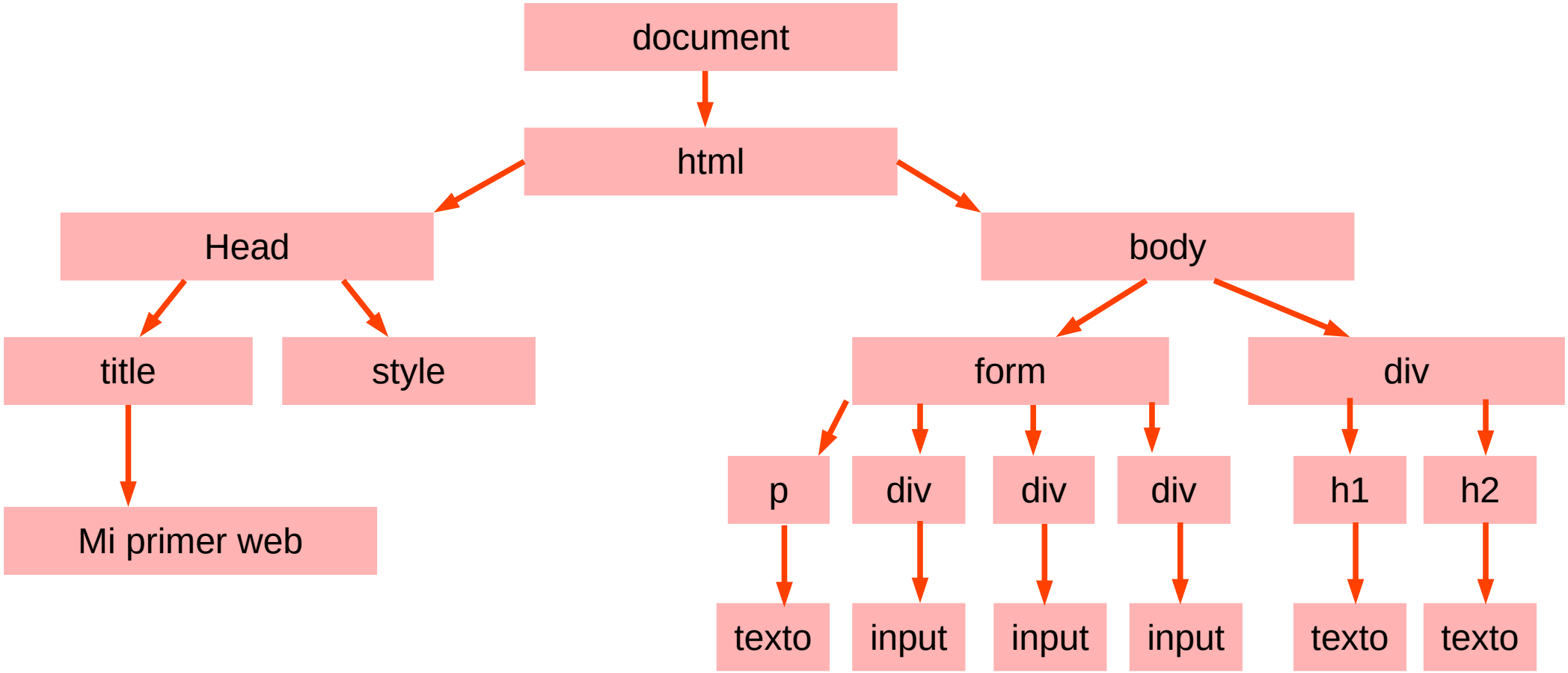
Insertar, reemplazar y eliminar elementos del DOM

¿Qué es el DOM?

- Las siglas DOM significan **Document Object Model** o estructura del documento HTML.
- Representa un modelo de cómo se organiza jerárquicamente una página web.
- Cada elemento de la página HTML se representa con un nodo, el cual puede ser superior (nodo padre o parent) y nodos que derivan de nodos padre (nodos hijos o child).
- El nodo **document** es el nodo principal de una página web. Tiene como hijo al nodo **html**, el cual representa todo el contenido dentro de las etiquetas **<html></html>**.
- Un nodo constituye un elemento complejo que suele llevar información asociada.

¿Qué es el DOM?

Representación aproximada de la estructura del DOM de una página web.



~~¿Qué es el DOM?~~

Tipos de nodos

Seleccionar elementos del DOM

Creando elementos en el DOM

Insertar, reemplazar y eliminar elementos del DOM

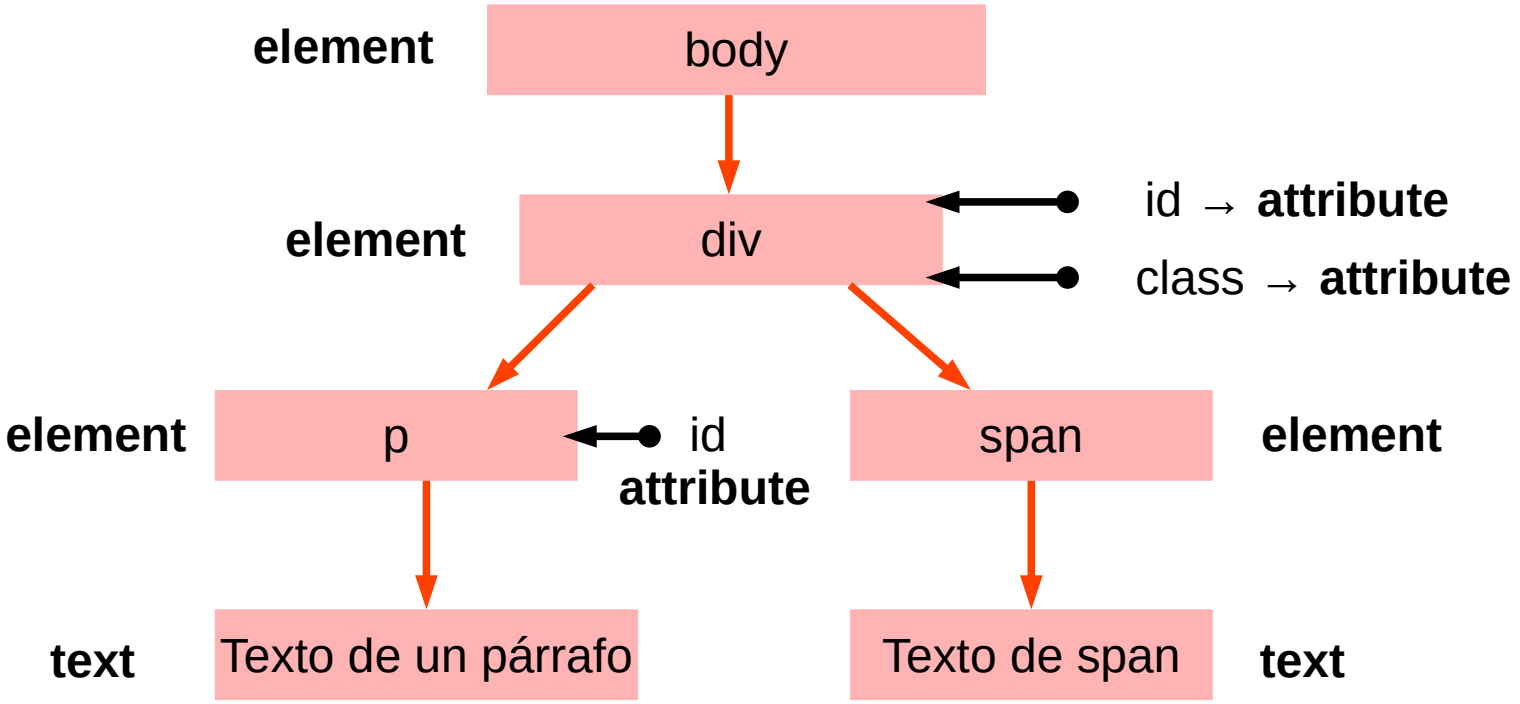
Tipos de nodos

Los nodos del DOM se pueden clasificar en diferentes tipos. Los principales son:

- **Document** → nodo raíz. Todos los nodos del documento HTML derivan de él.
- **Element** → Corresponde a aquellos nodos que han sido definidos por etiquetas HTML.
- **Text** → El texto dentro de un nodo se considera un nuevo nodo hijo de tipo **text** (texto).
- **Attribute** → los atributos de las etiquetas HTML definen nodos. En JavaScript no se los trata como nodo, sino como información asociada a un nodo tipo **element**.
- **Comentarios** → los comentarios en el documento generan nodos.

Tipos de nodos

Veamos un ejemplo de la representación de nodos según su tipo.



~~¿Qué es el DOM?~~

~~Tipos de nodos~~

Seleccionar elementos del DOM

Creando elementos en el DOM

Insertar, reemplazar y eliminar elementos del DOM

Seleccionar elementos del DOM

JavaScript nos provee de varios métodos para realizar búsquedas de elementos en el documento HTML. A continuación veremos los métodos tradicionales:

- **Buscar un elemento por su id**

```
const miDiv =  document.getElementById('unDiv');  →  // <div id='unDiv'></div>
```

Si no encuentra el elemento devuelve *NULL*.

Seleccionar elementos del DOM

- **Buscar elementos por su clase**

```
const misElementos =  document.getElementsByClassName('textoAzul');

misElementos [HTMLCollection]      →  // <div class='textoAzul'></div>
                                   →  //  <p class='textoAzul'></p>
                                   →  //  <h2 class='textoAzul'></h2>
```

Si no encuentra elementos devuelve *NULL*.

Seleccionar elementos del DOM

- **Buscar elementos por su nombre**

```
const inputUser =  document.getElementsByName('nombre');

inputUser      →  // <input name='nombre'></div>
```

- **Buscar elementos por su etiqueta**

```
const parrafos = document.getElementsByTagName('p');

parrafos  [HTMLCollection]      →  //  [<p>, <p>, <p>]
```

Seleccionar elementos del DOM

Aunque se pueden utilizar los métodos de búsqueda que acabamos de ver, existen nuevos métodos que permiten obtener elementos de una forma más práctica y sencilla si se dominan los selectores css.

- **.querySelector(selector)** → devuelve el primer elemento que coincida con el selector CSS que se indique. Equivalente a **.getElementById()** devuelve un solo elemento, caso contrario *NULL***.**

```
const titulo = document.querySelector("#titulo");
titulo →  // <div id='titulo'>...</div>


const info = document.querySelector(".blog .noticias");
info → // <div class='noticias'>...</div>
```

Seleccionar elementos del DOM

Aunque se pueden utilizar los métodos de búsqueda que acabamos de ver, existen nuevos métodos que permiten obtener elementos de una forma más práctica y sencilla si se dominan los selectores css.

- **.querySelectorAll(selector)** → devuelve todos los elementos que coincidan con el selector CSS que se indique. Devuelve un array con los elementos.

```
const imagenes = document.querySelectorAll(".imagenes");
imagenes →  // [ <div class='imagenes'>, <img class='imagenes'/> ]


const inputPasswords = document.querySelectorAll('[type="password"]');
inputPasswords → // [ <input type="password"/>, <input type="password"/>]
```

~~¿Qué es el DOM?~~

~~Tipos de nodos~~

~~Seleccionar elementos del DOM~~

Creando elementos en el DOM

Insertar, reemplazar y eliminar elementos del DOM

Creando elementos en el DOM

Aunque lo normal suele ser crear contenido HTML sobre un archivo **.html,** JavaScript permite crear contenido HTML directamente desde un archivo **.js**.

Ventajas y desventajas de este proceso

- Crear contenido HTML desde un archivo **.html** siempre será más sencillo, estático y directo.

- En cambio utilizando archivos **.js** el desarrollo del contenido HTML es más complejo y menos directo, pero mucho más potente, dinámico y flexible.

Creando elementos en el DOM

Existe una serie de métodos que permiten crear diferentes elementos HTML o nodos. Podemos emplear estos métodos para crear estructuras dinámicas de componentes HTML, por ejemplo mediante el uso de bucles o estructuras definidas.

- **createElement(tag)** → Crea y devuelve el elemento HTML definido por **tag**.
- **createComment(text)** → Crea y devuelve un nodo de comentarios HTML.
- **createTextNode(text)** → Crea y devuelve un nodo HTML con el texto **text**.
- **cloneNode(deep)** → Clona el nodo HTML y devuelve una copia. El parámetro **deep** es opcional. Por defecto su valor es **false**. Si se indica **true** clonará además los nodos hijos del elemento HTML que estamos duplicando.

Creando elementos en el DOM

createElement()

Este método permite crear un nodo tipo **element** HTML en memoria (aún no existe en el documento HTML). Al almacenar en una variable el nodo creado podemos modificar sus características o contenido y luego insertarlo en una posición determinada del documento HTML.

```
const div = document.createElement('div');        //Se crea un elemento <div></div>
const span = document.createElement('span')       //Se crea un elemento <span></span>
const form = document.createElement('form')       // Se crea un elemento <form></form>
```

Creando elementos en el DOM

De la misma forma podemos crear comentarios o nodos de texto con **createComment()** y **createTextNode()** respectivamente.

```
const comment = document.createComment('comentario');  // <!-- comentario -->

const text = document.createTextNode('Hola mundo');  // Nodo: 'Hola mundo'
```

Tener en cuenta que los ejemplos vistos anteriormente crean los elementos en una constante. Pero aún no se han agregado al documento HTML. Por lo cual no aparecerán visualmente.

Creando elementos en el DOM

El método cloneNode()

Un error muy común al momento de querer duplicar un elemento HTML es asignar el elemento en más de una variable pensando que estamos creando una copia del mismo componente.

Veamos el siguiente ejemplo:

```
const span = document.createElement('span');
span.textContent = "Elemento 1";


const span2 = span; (No crea una copia)
span2.textContent = "Elemento 2";


console.log(span.textContent)  // Devuelve 'Elemento 2'
```

Creando elementos en el DOM

El método cloneNode()

En el ejemplo anterior vemos que no se duplica el elemento HTML, sino que se está creando una nueva referencia (span2) hacia el mismo elemento. De esta manera accedemos al mismo componente HTML con **span** o **span2**.

Para solucionar esto, hacemos uso del método **cloneNode()**

```
const span = document.createElement('span');
span.textContent = "Elemento 1";


const span2 = span.cloneNode()  //Si se crea una copia del elemento
span2.textContent = "Elemento 2";


console.log(span.textContent)  // Devuelve 'Elemento 1'
```

Creando elementos en el DOM

Modificando atributos de un componente HTML con JavaScript

La manera más sencilla de asignar atributos a los elementos HTML que hemos creado previamente con funciones de JavaScript, es asignarle valores como propiedades de objetos:

```
const div = document.createElement('div');  // <div></div>

div.id = 'principal';      // <div id='principal'> </div>

div.className = 'texto-rojo';  // <div id='principal' class='texto-rojo'></div>
```

→ utilizamos **className** ya que **class** es una palabra reservada del lenguaje.

```
div.style = 'font-size: 1em';   // <div id='principal' class='texto-rojo' style='font-size:1em'></div>
```

Creando elementos en el DOM

Aunque la forma anterior es la más rápida, también tenemos algunos métodos para utilizar sobre un elemento HTML y agregar, eliminar o editar sus atributos.

- **hasAttributes()** → Indica si el elemento tiene atributos HTML. Devuelve true o false.
- **hasAttribute(attr)** → Indica si el elemento tiene el atributo HTML **attr**. Devuelve true o false.
- **getAttributeNames()** → Devuelve un array con los atributos del elemento.
- **getAttribute(attr)** → Devuelve el valor del atributo **attr** o NULL si no existe.
- **RemoveAttribute(attr)** → Elimina el atributo indicado.
- **SetAttribute(attr, valor)** → Agrega o cambia el atributo **attr** al valor **valor**.

Creando elementos en el DOM

Veamos un ejemplo.

Dado el siguiente elemento html

Index.html

```html
<div id='principal' class='texto-rojo' data-number="5"></div>
```

Index.js

```js
const div = document.querySelector('#principal');

div.hasAttributes();          //true
div.getAttributeNames();      //['id','class','data-number']

div.hasAttribute('data-number');     //true
div.getAttribute('data-number');     //5

div.setAttribute('id', 'noticias')  //<div id='noticias'.....></div>
div.removeAttribute('class'); // <div id='noticias' data-number='5'></div>
```

~~¿Qué es el DOM?~~

~~Tipos de nodos~~

~~Seleccionar elementos del DOM~~

~~Creando elementos en el DOM~~

Insertar, reemplazar y eliminar elementos del DOM

Reemplazando contenido

Los elementos HTML (nodos) presentan una familia de propiedades que permiten de forma sencilla reemplazar el contenido de una etiqueta HTML.

Las propiedades son:

- **.nodeName** → Devuelve el nombre del nodo (una etiqueta si es un elemento HTML).
- **.textContent** → Devuelve el contenido de texto del elemento. Se puede utilizar para asignar o modificar texto al elemento.
- **.innerHTML** → Devuelve el contenido HTML del elemento. Se puede utilizar para asignar o modificar el contenido HTML.
- **.outerHTML** → Similar a .**innerHTML** pero además incluye el HTML del propio elemento.

Reemplando contenido

Veamos a continuación un ejemplo

Dado el siguiente elemento HTML

Index.html

```html
<div id='principal' class='texto-rojo' data-number="5">
  <p>Texto del div</p>
</div>
```

Index.js

```js
const div = document.querySelector('#principal');

div.nodeName;        // Devuelve DIV
div.textContent;     // Devuelve "Texto del div"
div.innerHTML;       // Devuelve "<p>Texto del div</p>"
div.outerHTML        /* Devuelve "<div id='principal' class='texto-rojo' data-number='5'>
                                  <p>Texto del div</p>
                                </div>"
                     */

div.textContent = 'nuevo Texto';
div.textContent;     //Devuelve "nuevo Texto"
div.innerHTML;       //Devuelve "nuevo Texto"
```

Insertando contenido HTML en el DOM

Hasta ahora solo hemos creado elementos HTML en memoria y editado sus atributos. Los siguientes métodos permiten insertar elementos HTML en el DOM:

- **.appendChild(nodo)** → Agrega como hijo el nodo **nodo**. Devuelve el nodo insertado.
- **.insertAdjacentElement(pos, elem)** → Inserta el elemento **elem** en la posición **pos**.
- **.insertAdjacentHTML(pos, str)** → Inserta el código HTML **str** en la posición **pos**.
- **.insertAdjacentText(pos, text)** → Inserta el texto **text** en la posición **pos**.

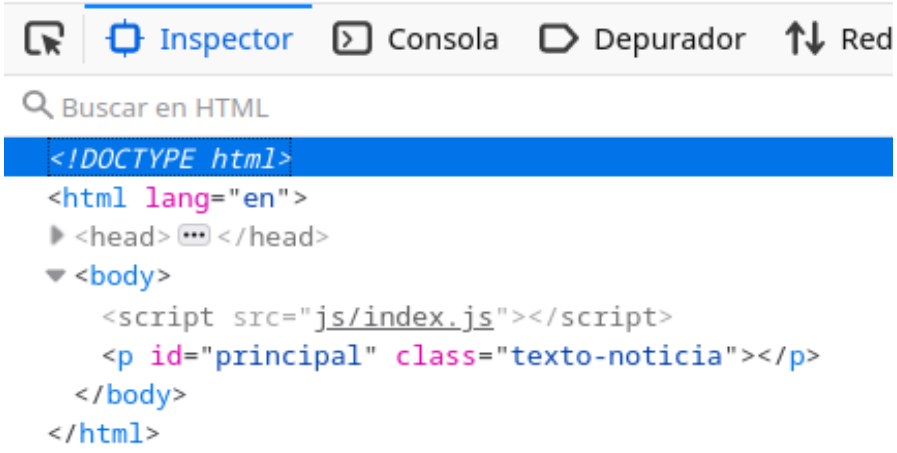
Insertando contenido HTML en el DOM

appendChild()

Este método es uno de los más comunes para agregar elementos HTML creados con JavaScript. Inserta el elemento como un hijo al final de todos los elementos hijos que existan.

```
const p = document.createElement('p');
p.id = 'principal';
p.className = 'texto-noticia';

document.body.appendChild(p);
```



Insertando contenido HTML en el DOM

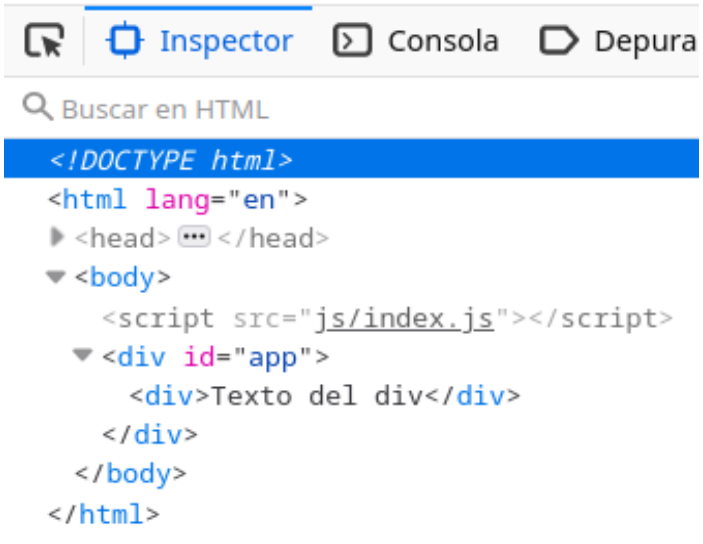
appendChild()

Veamos otro ejemplo

```
const div = document.createElement('div');
div.textContent = 'Texto del div';

const app = document.createElement('div');
app.id = 'app';
app.appendChild(div);

document.body.appendChild(app);
```



Insertando contenido HTML en el DOM

La familia de métodos insertAdjacentHTML(), insertAdjacentElement() e insertAdjacentText(), son más versátiles que **.appendChild()** ya que brindan mayores posibilidades de insertar elementos en diversas posiciones.

En cualquiera de las versiones de **insertAdjacent** se debe indicar por parámetro la posición en donde se va a insertar el contenido. Las opciones son:

- **beforebegin** → el elemento se inserta **antes** de la etiqueta HTML de apertura.
- **afterbegin** → el elemento se inserta **dentro** de la etiqueta HTML, **antes del primer hijo**.
- **beforeend** → el elemento se inserta **dentro** de la etiqueta HTML, **después de su último hijo**. Es equivalente a **.appendChild()**.
- **afterend** → el elemento se inserta **despúes** de la etiqueta HTML de cierre.

Insertando contenido HTML en el DOM

Veamos un ejemplo:

```
const div = document.createElement('div');
div.textContent = 'Texto del div';

const app = document.createElement('div');
app.id = 'app';
app.textContent = 'Texto del app';

app.insertAdjacentElement('beforebegin', div);
//Resultado: <div>Texto del div</div> <div id='app'>Texto del app</div>

app.insertAdjacentElement('afterbegin', div);
//Resultado: <div id='app'> <div>Texto del div</div> Texto del app</div>

app.insertAdjacentElement('beforeend', div);
//Resultado: <div id='app'>Texto del app <div>Texto del div</div> </div>

app.insertAdjacentElement('afterend', div);
//Resultado: <div id='app'>Texto del app</div> <div>Texto del div</div>
```

Eliminar contenido HTML del DOM

Al igual que con insertar o reemplazar elementos también podemos eliminarlos. Eliminar un elemento no implica ser borrado, sino **desconectarlo del DOM o documento HTML.**

El método mas sencillo que nos brinda JavaScript es utilizar **.remove()** sobre el elemento o etiqueta que se desea eliminar.

```
const div = document.querySelector('.eliminar');

div.isConnected;        //true
div.remove();
div.isConnected;        //false
```

*El atributo **.isConnected** devuelve true o false dependiendo si el elemento HTML se encuentra conectado o no al DOM.*

Eliminar contenido HTML del DOM

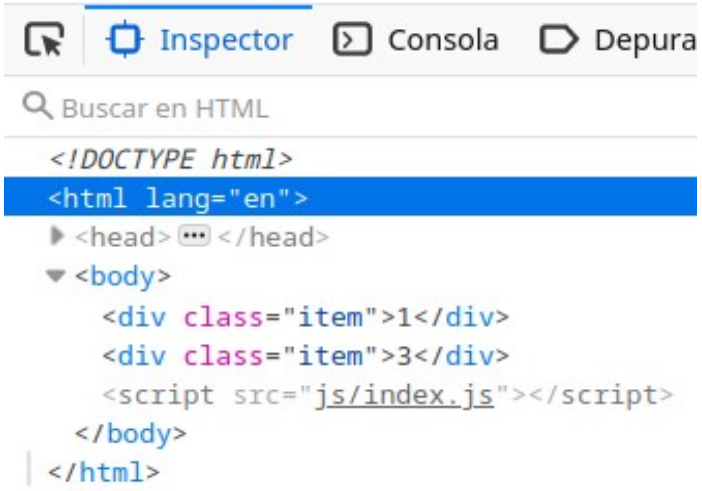
En algunos casos nos interesará eliminar sólo un nodo hijo de un elemento. En este caso podemos hacer uso del método **.removeChild(node)** donde **node** es el nodo hijo que vamos a eliminar.

```
<body>

  <div class='item'>1</div>
  <div class='item'>2</div>
  <div class='item'>3</div>

<script src="js/index.js"></script>
</body>
```

```
const div = document.querySelector(".item:nth-child(2)");
document.body.removeChild(div);
```



*:nth-child(2) obtiene el segundo elemento (en este caso el segundo elemento que contenga la clase **.item**)*

¿Consultas?