DWC

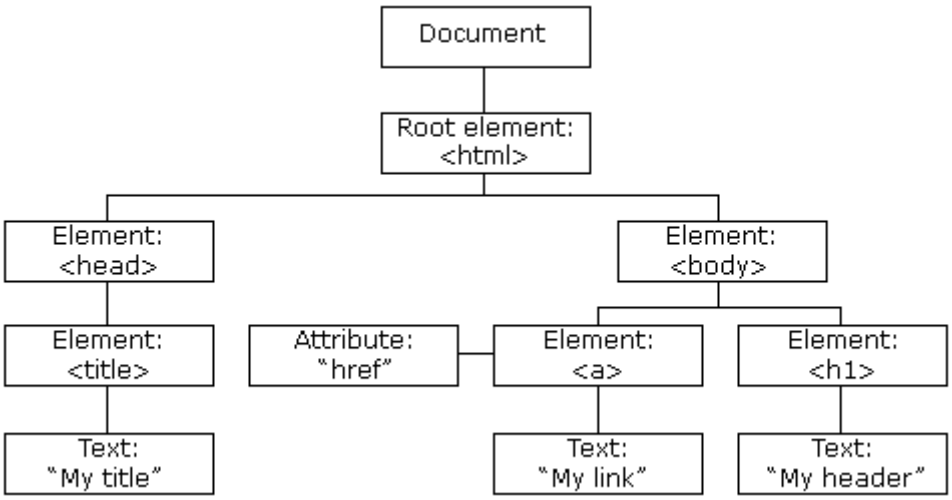
(Desarrollo Web en entorno cliente)

Índice

1.- El HTML DOM (Modelo de objeto de documento)

Cuando se carga una página web, el navegador crea una **D** DOCUMENTO **O** Object **M** Modelo de la página.

El modelo **HTML DOM** se construye como un árbol de **objetos**:



Con el modelo de objetos, JavaScript obtiene toda la potencia que necesita para **crear HTML dinámico**:

- JavaScript puede cambiar todos los elementos HTML en la página
- JavaScript puede cambiar todos los atributos HTML en la página
- JavaScript puede cambiar todos los estilos CSS en la página
- JavaScript puede eliminar elementos y atributos HTML existentes
- JavaScript puede agregar nuevos elementos y atributos HTML
- JavaScript puede reaccionar a todos los eventos HTML existentes en la página
- JavaScript puede crear nuevos eventos HTML en la página

1.1.- ¿Qué es el DOM?

El DOM es un estándar W3C (World Wide Web Consortium).

El DOM define un estándar para acceder a los documentos:

"El Modelo de objetos de documento (DOM) del W3C es una plataforma y una interfaz de lenguaje neutral que permite que los programas y los scripts accedan y actualicen dinámicamente el contenido, la estructura y el estilo de un documento".

El estándar W3C DOM está separado en 3 partes diferentes:

- **Core DOM**: modelo estándar para todos los tipos de documentos
- **XML DOM**: modelo estándar para documentos XML
- **HTML DOM** - modelo estándar para documentos HTML

1.2.- ¿Qué es el HTML DOM?

El HTML DOM es un modelo de **objetos** estándar y una **interfaz de programación** para HTML. Se define:

- Los elementos HTML como **objetos**
- Las **propiedades** de todos los elementos HTML.
- Los **métodos** para acceder a todos los elementos HTML
- Los **eventos** para todos los elementos HTML.

En otras palabras: **el HTML DOM es un estándar sobre cómo obtener, cambiar, agregar o eliminar elementos HTML.**

2.- La interfaz de programación DOM

Se puede acceder al HTML DOM con JavaScript (y con otros lenguajes de programación).
En el DOM, todos los elementos HTML se definen como **objetos**.
La interfaz de programación son las propiedades y métodos de cada objeto.

- Una **propiedad** es un valor que puede obtener o establecer (como cambiar el contenido de un elemento HTML).
- Un **método** es una acción que puede hacer (como agregar o eliminar un elemento HTML).

El siguiente ejemplo cambia el contenido (**el innerHTML**) del elemento <p> con id="demo":

```
<html>
<body>
 <p id="demo"></p>
 <script>
  document.getElementById("demo").innerHTML = "Hello World!";
 </script>
```

```
</body>
</html>
```

En el ejemplo anterior, getElementById es un **método**, mientras que innerHTML es una **propiedad**.

El método getElementById

La forma más común de acceder a un elemento HTML es usar el id del elemento. En el ejemplo anterior, el método getElementById es utilizado para encontrar el elemento con id="demo".

La propiedad innerHTML

La forma más fácil de obtener el contenido de un elemento es mediante el uso de la propiedad innerHTML. Esta propiedad es útil para obtener o reemplazar el contenido de elementos HTML. Es un atajo a la utilización estricta de métodos DOM para añadir o cambiar elementos HTML a través de la inserccion de elementos DOM como **objeto.appendChild(nodo)**

La propiedad innerHTML se puede utilizar para obtener o cambiar cualquier elemento HTML, incluidos <html>y <body>.

3.- El objeto de documento DOM HTML

El DOM representa nuestra página web.

Si deseamos acceder a cualquier elemento en una página HTML, siempre tenemos que acceder al objeto del documento.

A continuación, se muestran algunos ejemplos de cómo podemos usar el DOM para acceder y manipular HTML.

Encontrar elementos HTML

Método	Descripción
document.getElementById(id)	Encuentra un elemento por id
document.getElementsByTagName(name)	Encuentra elementos por etiqueta HTML
document.getElementsByClassName(name)	Encuentra elementos por clase CSS

Cambiar elementos HTML

Propiedad	Descripción
element.innerHTML = nuevo contenido HTML	Cambia el contenido HTML de un elemento
element.attribute = nuevo valor	Cambia el valor de un atributo de un elemento HTML
element.style.property = nuevo estilo	Cambia el estilo de un elemento HTML

Método	Descripción
element.setAttribute(attribute, value)	Cambia el valor de un atributo de un elemento HTML

Esta última forma sirve para lo mismo que la propiedad de la tabla anterior, la diferencia es que es un método que tiene como argumentos el valor y el atributo.

Añadir y eliminar elementos

Método	Descripción
document.createElement(element)	Crea un elemento HTML
document.removeChild(element)	Elimina un elemento HTML
document.appendChild(element)	Añade un elemento HTML
document.replaceChild(new, old)	Reemplaza un elemento HTML
document.write(text)	Escribe sobre el HTML de la pagina

Agregar controladores de eventos

Metodo
document.getElementById(id).onclick = function(){code}
Descripción
Añade un controlador de evento a un element HTML, en este caso añade una función para gestionar el evento onclick del elemento HTML con id=id

Encontrar objetos HTML

El primer HTML DOM Nivel 1 (1998), definió 11 objetos HTML, colecciones de objetos y propiedades. Estos todavía son válidos en HTML5.

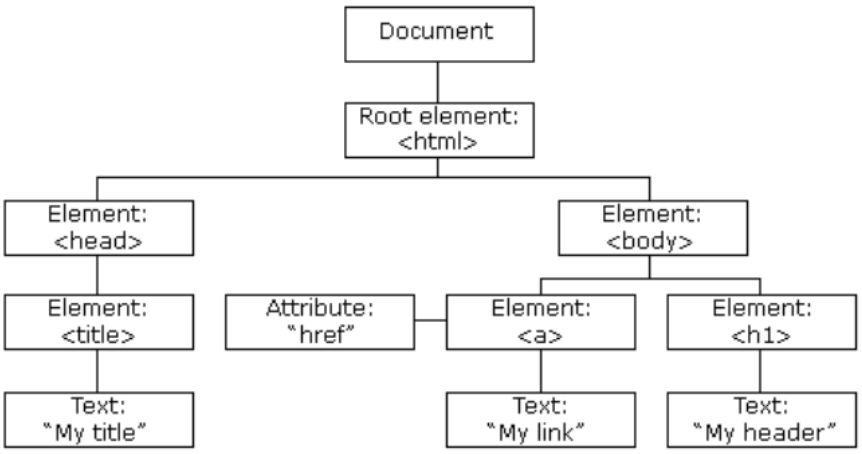
Más tarde, en HTML DOM Nivel 3, se agregaron más objetos, colecciones y propiedades como

- document.anchors
- document.body
- document.cookie
- document.doctype
- document.forms
- document.images
- document.links
- document.readyState

4.- Nodos DOM

De acuerdo con el estándar W3C HTML DOM, todo en un documento HTML es un nodo:

- Todo el documento es un nodo de documento.
- Cada elemento HTML es un nodo de elemento
- El texto dentro de los elementos HTML son nodos de texto.
- Cada atributo HTML es un nodo de atributo (en desuso)
- Todos los comentarios son nodos de comentarios



Con HTML DOM, JavaScript puede **acceder a todos los nodos** del árbol de nodos.

Se pueden **crear nuevos nodos y todos los nodos se pueden modificar o eliminar**.

4.1.- Relaciones entre nodos

Los nodos en el árbol de nodos tienen una relación jerárquica entre sí. Los términos **padre, hijo y hermano** se usan para describir las relaciones.

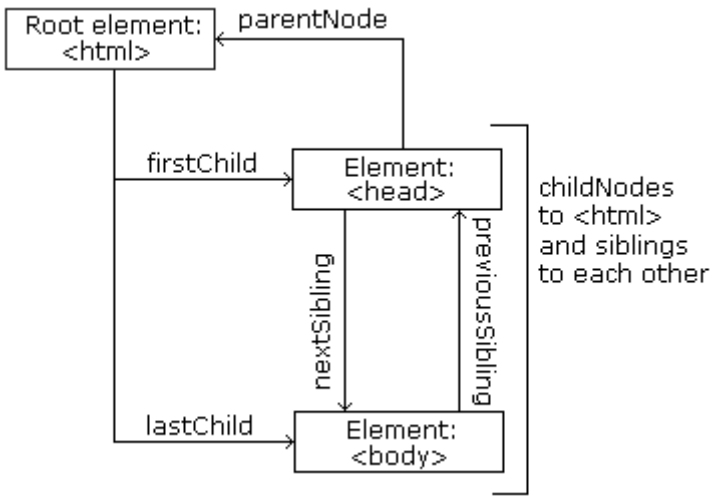
- En un árbol de nodos, el nodo superior se llama raíz (o nodo raíz)
- Cada nodo tiene exactamente un padre, excepto la raíz (que no tiene padre)
- Un nodo puede tener varios hijos
- Los hermanos (hermanos o hermanas) son nodos con el mismo padre

```
<html>

 <head>
  <title>DOM
Tutorial</title>
 </head>

 <body>
  <h1>DOM          Lesson
one</h1>
  <p>Hello world!</p>
 </body>

</html>
```



Del HTML anterior podemos ver que respecto a <html>:

- <html> es el nodo raíz
- <html> no tiene padres
- <html>es el padre de <head>y<body>
- <head> es el primer hijo de <html>
- <body> es el último hijo de <html>

Y respecto al resto de elementos:

- <head> tiene un hijo: <title>
- <title> tiene un hijo (un nodo de texto): "Tutorial DOM"
- <body>tiene dos hijos: <h1>y<p>
- <h1> tiene un hijo: "DOM Lección uno"
- <p> tiene un hijo: "¡Hola mundo!"
- <h1>y <p>son hermanos

4.2.- Propiedades de los nodos

Los nodos son objetos que forman el DOM y como tal tienen propiedades y métodos. Las propiedades son atributos que permiten identificar el nodo y saber su relación con el resto del DOM. Vamos a ver las propiedades más importantes.

4.2.1.- Propiedades básicas de los nodos

Vamos a ver las propiedades básicas que identifican a un nodo y el tipo que representa.

- La propiedad **nodeName:** especifica el nombre de un nodo.

  nodeName es de solo lectura

  nodeName de un nodo de elemento es el mismo que el nombre de la etiqueta

  nodeName de un nodo de atributo es el nombre del atributo

  nodeName de un nodo de texto siempre es #text

  nodeName del nodo del documento siempre es #documento

```
<h1 id="id01">My First Page</h1>

<script>
alert(document.getElementById("id01").nodeName);
</script>
```

  nodeName siempre contiene el nombre de etiqueta en mayúscula de un elemento HTML.

- La propiedad **nodeValue** especifica el valor de un nodo.
  nodeValue para elementos nodos es null
  nodeValue para los nodos de texto es el texto en sí
  nodeValue para los nodos de atributo es el valor del atributo

- La propiedad **nodeType** devuelve el tipo de un nodo. Es de sólo lectura. Igual que una página web está formada por distintos elementos los nodos del árbol DOM también son diferentes. En el estándar DOM se definen diferentes tipos de nodos en función del elemento que representan.

  Los más importantes son:

Node	Type	Example
ELEMENT_NODE	1	<h1 class="heading">Titulo</h1>
ATTRIBUTE_NODE	2	class = "heading" (deprecated)
TEXT_NODE	3	Texto de un elemento
COMMENT_NODE	8	<!-- This is a comment -->
DOCUMENT_NODE	9	The HTML document itself (the parent of <html>)
DOCUMENT_TYPE_NODE	10	<!Doctype html>

4.2.2.- Propiedades de relación entre los nodos

Podemos usar las siguientes propiedades de nodo para saber su relación con los demás nodos y poder desplazarnos entre nodos

- **parentNode:** nodo padre de un nodo
- **childNodes[nodenumber]:** colección de hijos de un nodo
- **firstChild:** primer hijo de un nodo
- **lastChild:** ultimo hijo de un nodo
- **nextSibling:** siguiente nodo hermano de un nodo
- **previousSibling**: anterior nodo hermano de un nodo

Nodos secundarios y valores de nodo

Un error común en el procesamiento DOM es esperar que un nodo de elemento contenga texto.

```
<title id="demo">DOM Tutorial</title>
```

El nodo del elemento <title>(en el ejemplo anterior) no contiene texto. Contiene un nodo de texto con el valor "Tutorial DOM".

Se puede acceder al valor del nodo de texto mediante la propiedad innerHTML del nodo:

```
let myTitle = document.getElementById("demo").innerHTML;
```

Acceder a la propiedad innerHTML es lo mismo que acceder nodeValue al primer hijo:

```
let myTitle = document.getElementById("demo").firstChild.nodeValue;
```

Acceder al primer hijo también se puede hacer así:

```
let myTitle = document.getElementById("demo").childNodes[0].nodeValue;
```

En el siguiente ejemplo vemos las tres formas de copiar el texto de un elemento <h1> en un elemento <p>:

```
<html>
<body>
  <h1 id="id01">My First Page</h1>
  <p id="id02"></p>

<script>
document.getElementById("id02").innerHTML = document.getElementById("id01").innerHTML;
document.getElementById("id02").innerHTML = document.getElementById("id01").firstChild.nodeValue;
document.getElementById("id02").innerHTML =
document.getElementById("id01").childNodes[0].nodeValue;
</script>

</body>
</html>
```

La propiedad innerHTML es un atajo rápido para acceder al contenido HTML de un elemento, sin embargo, aprender los otros métodos anteriores es útil para comprender la estructura de árbol y la navegación del DOM.

Propiedades Nodos raíz DOM

Hay dos propiedades especiales que permiten el acceso al documento completo:

- **document.body** - El cuerpo del documento
- **document.documentElement** - El documento completo

Si queremos insertar algo directamente en el body utilizaremos cualquiera de las dos anteriores, en caso contrario deberemos localizar el nodo en concreto sobre el que queramos realizar la operación.

5.- Encontrar y acceder a elementos HTML

A menudo, con JavaScript, deseamos manipular elementos HTML. Para hacerlo, primero debemos encontrar los elementos. Hay varias formas de hacer esto:

- Encontrar elementos HTML por ID
- Encontrar elementos HTML por nombre de etiqueta
- Encontrar elementos HTML por nombre de clase
- Encontrar elementos HTML por selectores CSS
- Encontrar elementos HTML por colecciones de objetos HTML

5.1.- Encontrar elemento HTML por id

La forma más fácil de encontrar un elemento HTML en el DOM es mediante el uso del id del elemento.Para ello usaremos **getElementById("id")**

Este ejemplo encuentra el elemento con id="intro":

```
let myElement = document.getElementById("intro");
```

Si se encuentra el elemento, el método devolverá el elemento como un objeto (en myElement). **Si no lo encuentra el elemento myElement contendrá null**.

5.2.- Encontrar elementos HTML por nombre de etiqueta

Cuando queremos seleccionar varios elementos HTML del mismo tipo utilizaremos **getElementsByTagName(HTML Tag)**. Lo que realizamos es una selección por etiqueta HTM, esta selección devuelve un **HTML Collection** "*array de elementos*" con todos los elementos encontrados

```
<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>
```

```
<script>
  let parrafos = document.getElementsByTagName("p");
  for(i=0;parrafos.length;i++)
        alert(parrafos[i].innerHTML)
</script>
```

Este ejemplo encuentra todos los elementos <p> y luego los muestra uno a uno utilizando un bucle para recorrer el HTML Collection, **ya que no es realmente un array y no se pueden utilizar los métdos de los arrays**.

Una alternativa para trabajar con los métodos de los arrays es convertir el HTML Collection en un Array con el método Array.from(HTML Collection).

```
<h2>Finding HTML Elements by Tag Name</h2>

<p>Hello World!</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>
<p id="demo"></p>

<script>
  let parrafos = document.getElementsByTagName("p");

  arrayDeParrafos=Array.from(parrafos);
  arrayDeParrafos.forEach(n=>alert(n.innerHTML));
</script>
```

Búsqueda de elementos hijos

La estructura del DOM es un árbol cuya raíz es el propio documento, con lo cual todas las búsquedas que partan desde la raíz localizarán todos los elementos que deseamos buscar. Muchas veces lo que deseamos encontrar son elementos que se encuentran en un lugar determinado y no por todo el documento, en este caso la búsqueda la deberíamos lanzar no desde la raíz, sino desde el elemento contenedor (el padre). Para ello lo primero que deberemos hacer es encontrar al padre y después desde el padre lanzar la búsqueda de los elementos que queremos encontrar dentro de él.

El siguiente ejemplo localiza todos los párrafos del div con id=c2 y los muestra en un alert.

```
<h2>Finding HTML Elements inside other element</h2>

<div id="c1">
  <p> Parrafo 1 de la capa c1 </p>
  <p> Parrafo 2 de la capa c1 </p>
</div>

<div id="c2">
  <p> Parrafo 1 de la capa c2 </p>
  <p> Parrafo 2 de la capa c2 </p>
</div>

<div id="c3">
  <p> Parrafo 1 de la capa c3 </p>
  <p> Parrafo 2 de la capa c3 </p>
</div>

<script>
  let divC2 = document.getElementById("c2");
  let parrafosC2=divC2.getElementsByTagName("p")

   arrayDeParrafos=Array.from(parrafosC2);
   arrayDeParrafos.forEach(n=>alert(n.innerHTML));
</script>
```

5.3.- Encontrar elementos HTML por nombre de clase

Muchas veces nos interesa localizar elementos que tengan la misma clase CSS para ello deberemos utilizar **getElementsByClassName("clase")**.

Esta forma de selección es muy utilizada para seleccionar elementos que o bien no son todos hijos del mismo elemento o simplemente son elementos HTML diferentes. Muchas veces se asigna a los elementos una clase ficticia que sirve para poder seleccionarlos más que para asignarles un estilo CSS determinado.

En este caso el objeto devuelto está formado por todos los elementos que tienen el atributo class indicado, es decir puede estar formado por diferentes elementos HTML (<p>, , <button>…). En este caso se devuelve un **Node List** "*array de elementos*" con todos los elementos encontrados. Realmente un Node List no es un array y no se pueden aplicar los métodos propios de los array, con lo cual, deberemos acceder a ellos con bucle y recorrerlos uno a uno, o convertirlo en array como hicimos con el HTML Collection usando Array.from(Node List)

Este ejemplo devuelve una lista de todos los elementos con class="prueba" y saca ese elemento en un alert indicanto el tipo de elemento que es div o p

```
<h2>Finding HTML Elements by className</h2>

<div id="c1" class="prueba">
  <p class="prueba"> Parrafo 1 de la capa c1 </p>
  <p> Parrafo 2 de la capa c1 </p>
</div>

<div id="c2">
  <p> Parrafo 1 de la capa c2 </p>
  <p class="prueba"> Parrafo 2 de la capa c2 </p>
</div>

<div id="c3" class="prueba">
  <p class="prueba"> Parrafo 1 de la capa c3 </p>
  <p> Parrafo 2 de la capa c3 </p>
</div>

<script>
   let clasePrueba=document.getElementsByClassName("prueba")

   arrayClasePrueba=Array.from(clasePrueba);
   arrayClasePrueba.forEach(n=>alert(n));
</script>
```

5.4.- Encontrar elementos HTML mediante selectores CSS

Si desea encontrar todos los elementos HTML que coincidan con un selector CSS especificado (id, nombres de clase, tipos, atributos, valores de atributos, etc.), deberemos usar el método **querySelectorAll(“css”)**. Este método es de reciente incorporación al estándar de JS y sólo esta soportado en versiones recientes de los navegadores.

Es un método muy potente ya que permite aprovechar toda la potencia del css para realizar búsquedas. Las pseudoclases también se pueden utilizar.

Este ejemplo devuelve todos los elementos <li> que pertenecen a listas <ul> que además ocupan la última posición.

```
<ul>
  <li>The</li>
  <li>test</li>
</ul>
<ul>
  <li>has</li>
  <li>passed</li>
</ul>
<script>
  let elements = document.querySelectorAll('ul > li:last-child');

  for (let elem of elements) {
    alert(elem.innerHTML); // "test", "passed"
  }
</script>
```

En este caso hemos utilizado el bucle for para objetos en vez de convertir el Node List en un array

La llamada a **querySelector("css")** devuelve el primer elemento para el selector CSS dado. En otras palabras, el resultado es el mismo querySelectorAll(css)[0], pero este último busca todos los elementos y elige uno, mientras que elem.querySelector solo busca uno. Es más rápido y también más corto de escribir.

5.5.- Encontrar elementos HTML por colecciones de objetos HTML

Como vimos anteriormente al lenguaje se le incorporaron objetos y colecciones para agrupar elementos concretos. Podemos realizar búsquedas sobre esos elementos también. Es decir:

Podemos acceder a todas las imágenes del documento mediante:

```
let misImagenes= document.images
```

Podemos acceder a todos los hipervínculos del documento mediante:

```
let misLinks= document.links
```

Podemos acceder a todos los formularios del documento mediante:

```
let misForms=document.forms
```

Este tipo de selección por objetos lo veremos para el caso de los formularios.

6.- Modificar los elementos del DOM

Una vez que hemos accedido a los elementos del DOM con cualquiera de las formas que hemos visto, podemos modificar las propiedades de los elementos modificando dinámicamente los elementos HTML de nuestra página.

Las modificaciones son de dos tipos de **atributos** o de **estilo CSS**

6.1.- Cambiar el valor de un atributo

Para cambiar el valor de un atributo HTML, usaremos la sintaxis:

```
element.attribute = new value
```

Todos los elementos HTML a los que accedemos tendrán los mismos atributos que vienen definidos por el estándar HTML

Este ejemplo cambia el valor del atributo src de un elemento <img>:

```
<img id="myImage" src="smiley.gif">

<script>
  document.getElementById("myImage").src = "landscape.jpg";
</script>
```

6.2.- Cambiar estilo HTML

Para cambiar el estilo de un elemento HTML, usaremos esta sintaxis:

```
elemento.style.property = new style
```

El siguiente ejemplo cambia el estilo de un elemento <p>:

```
<p id="p2">Hello World!</p>

<script>
  document.getElementById("p2").style.color = "blue";
</script>

<p>The paragraph above was changed by a script.</p>
```

7.- Operaciones sobre nodos DOM

Hasta ahora hemos accedido a elementos que ya existían en el DOM y una vez que teníamos acceso hemos podido modificarle las propiedades. En muchas ocasiones lo que vamos a necesitar es crear, modificar, eliminar o clonar nodos. Estas operaciones se utilizan para crear contenido dinámicamente que no se había diseñado en la página HTML, es decir vamos a modificar la estructura del documento HTML.

El atajo a todas estas operaciones básicas es la propiedad innerHtml ya que como hemos visto permite poder definir el HTML de un contenedor escribiendo las etiquetas HTML de su interior. En este caso no vamos a realizar esto, vamos a crear o eliminar nodos del DOM, es decir objetos que representan a las etiquetas del HTML y posteriormente las añadiremos o eliminaremos en un contenedor.

7.1.- Crear nuevos elementos HTML (nodos)

Para agregar un nuevo elemento al DOM HTML debemos hacer dos pasos:

- primero se debe crear el elemento (nodo del elemento)
- agregar el elemento nuevo a un elemento existente.

Para crear un elemento utilizaremos **document.createElement("etiqueta HTML")**

Este método lo que hace es crear un nodo del DOM que representa al objeto de la etiqueta HTML que hemos utilizado. En el momento que creamos el nodo tenemos acceso a todas las propiedades y métodos de ese objeto que representa la etiqueta HTML.

Por ejemplo:

```
let imagen=document.createElement(“img”)
imagen.id=”mi_foto”
imagen.src=”./fotos/logo.jpg”
```

En este caso hemos creado un nodo llamado imagen que es un elemento HTML de la clase <img>, en el momento que lo creamos tenemos acceso a todas las propiedades y métodos de ese tipo de etiqueta HTML. Después de crearlo le

asignamos un id (fundamental para poder localizarlo posteriormente) y la propiedad más importante es src ya que determina la imagen que se mostrará.
En este momento tenemos un nodo que en esencia es una variable nuestra como programadores y que estamos utilizando en el código de nuestro script, en la página HTML no aparece porque lo único que hemos hecho es crear el nodo.

Para que el nodo aparezca en la página web deberemos de añadirlo en el lugar que queramos, para ello utilizaremos el método **appendChild(node).**
En este a un nodo padre le insertaremos el nodo nuevo. En nuestro caso anterior, supongamos que queremos añadir la imagen a un div con id=”fotos”

```
<div id=”texto”>
  <p>Parrafo 1</p>
  <p>Parrafo 2</p>
</div>
<div id=”fotos”> //inicialmente vacia
</div>

<script>
  divFotos=document.getElementById(“fotos”);
  let imagen=document.createElement(“img”);
  imagen.id=”mi_foto”
  imagen.src=”./fotos/logo.jpg”

  divFotos.appendChild(imagen)
</script>
```

Hay etiquetas HTML que llevan dentro de ellas texto, como por ejemplo la etiqueta <p>, en esto caso hay que crear dos nodos, uno para la etiqueta <p> y otro para el texto de dentro de la etiqueta <p>. El texto es el hijo de la etiqueta <p> y deberemos proceder como en el caso anterior y añadir el texto como hijo de <p>. Finalmente deberemos añadir el nodo que representa a <p> (que ya lleva dentro añadido el texto) en el lugar de la página web que deseemos.
Para crear nodos de texto utilizaremos **document.createTextNode(“Texto”)**

Veamos un ejemplo

- Creamos elemento <p>

```
let miParafo = document.createElement("p");
```

- Para agregar texto al elemento <p>, primero debe crear un nodo de texto.

```
let miTexto= document.createTextNode("Texto del parrafo.");
```

- Agregamos el nodo de texto al elemento <p>:

```
miParrafo.appendChild(miTexto);
```

- Finalmente, agregamos el nuevo elemento a un elemento existente, en este caso al elemento con id=div1.

```
let padre = document.getElementById("div1");
padre.appendChild(miParrafo);
```

El método padre.appendChild(nodo) añade un nuevo nodo en la posición final del padre, es decir si añado un párrafo nuevo a una capa que ya tiene 5 párrafos y una imagen, el nuevo párrafo ocupará la última posición de la capa.

Hay ocasiones en las que nos interesa añadir un nodo nuevo en una posición que no sea la última. Para eso debemos de indicar antes de que nodo queremos añadir el nuestro. Utilizaremos el método **insertBefore(nodo)**

Veamos un ejemplo:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
  let miparrafo = document.createElement("p");
  let miTexto = document.createTextNode("This is new.");
  miParrafo.appendChild(miTexto);

let padre = document.getElementById("div1");
let elementoHijo = document.getElementById("p1");
padre.insertBefore(miParrafo, elementoHijo);
</script>
```

En este caso creamos un párrafo y su texto asociado y lo añadimos en la capa con id=”div1” en la posición anterior a la que ocupa el párrafo con id=”p1”

7.2.- Eliminar elementos HTML existentes

Para eliminar un elemento HTML, usaremos el método **remove()**. Este método es muy sencillo, localizamos el nodo a eliminar y lo eliminamos directamente.

```
<div>
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
let parrafo = document.getElementById("p1");
parrafo.remove();
</script>
```

El método remove() no funciona en navegadores antiguos, esto es debido a que se incorporó en los últimos estándares. Lo que se hacía antes y aún se puede hacer es eliminar un elemento HTML hijo.
Para ello lo que se debe hacer es ejecutar el método de borrado desde el objeto padre del elemento que deseemos eliminar.
Para realizar el borrado de un hijo usaremos **removeChild(nodo)**.

Veamos un ejemplo en el cuál vamos a eliminar un parrafo con id=”p1” que se encuentra en una capa con id=”div1”

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
  let padre = document.getElementById("div1");
  let nodo = document.getElementById("p1");
  padre.removeChild(nodo);
</script>
```

En este caso hemos borrado desde el padre localizando previamente al padre. Hay ocasiones en las que sabemos que elemento queremos eliminar, pero no sabemos dónde se encuentra, con lo cual no podemos utilizar la referencia de su padre porque no sabemos cuál es. Para solucionar este problema, utilizamos

una propiedad que tienen todos los nodos del DOM que hace referencia a su padre **parentNode**

El ejemplo anterior se podría haber hecho de la siguiente manera:

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
  let nodo = document.getElementById("p1");
  nodo.parentNode.removeChild(nodo);
</script>
```

7.3.- Sustitución de elementos HTML

Para reemplazar un elemento al HTML DOM, usaremos el método **replaceChild(nodoNuevo, nodoAReemplazar)**:

En este método a un nodo padre le indicamos que vamos a reemplazar un nodo nuevo por otro ya existente)

```
<div id="div1">
  <p id="p1">This is a paragraph.</p>
  <p id="p2">This is another paragraph.</p>
</div>

<script>
  let miParrafo = document.createElement("p");
  let node = document.createTextNode("This is new.");
  miParrafo.appendChild(node);

  let parent = document.getElementById("div1");
  let child = document.getElementById("p1");
  parent.replaceChild(miParrafo, child);
</script>
```

7.4.- Clonar elementos HTML

Existe un método que permite hacer una copia exacta de un nodo y de todas sus propiedades y métodos. El método es **cloneNode(deep)**

El parámetro deep es opcional y del tipo booleano.

- True indica que se clonara el nodo y todos sus descendientes
- False indica que sólo se copiara el nodo seleccionado

El método devuelve el nodo clonado, después de obtener el nodo clonado lo deberemos añadir en algún lugar del árbol DOM utilizando los métodos appendChild() o insertBefore()

En este ejemplo seleccionamos el último hijo de la lista con id=”myList2” y lo clonamos en un nodo en la variable cln. Finalmente, lo añadimos a la lista con id=”myList1”

```
<ul id="myList1"><li>Coffee</li><li>Tea</li></ul>
<ul id="myList2"><li>Water</li><li>Milk</li></ul>
<script>
  var itm = document.getElementById("myList2").lastChild;
  var cln = itm.cloneNode(true);
  document.getElementById("myList1").appendChild(cln);
</script>
```

8.- Asignar eventos utilizando el HTML DOM

El DOM HTML nos asignar eventos a elementos HTML. Los eventos los veremos más adelante, ahora solo vamos a ver una pequeña referencia.

Hasta ahora sabemos que es el DOM, como recorrerlo y modificar las propiedades de los elementos. Los elementos del DOM son elementos HTML y como tal pueden tener asociados eventos. Los eventos son cosas que pueden suceder sobre un elemento HTML (al hacer click, al cambiar, al pulsar una tecla…). Los eventos se gestionan a través de funciones, asignando a un evento determinado una función que actuara como el controlador del evento sobre ese elemento.

Si creamos un elemento nuevo y queremos que reaccione ante algún evento sobre él deberemos (al igual que asignamos valores a las propiedades) asignar código para la gestión del evento.

Vamos a asignar a un botón existente en la página web una función para gestionar el evento onclick (al hacer click sobre él).

En este caso asignaremos la función al evento onclick y lo que hará es mostrar en un párrafo la hora actual utilizando la función Date()

```
<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
document.getElementById("myBtn").onclick = displayDate;

function displayDate() {
  document.getElementById("demo").innerHTML = Date();
}
</script>
```

Muy importante la asignación de la función al evento onclick sin usar ()

Vamos ahora a crear un elemento de tipo botón y vamos a asignarle el evento onclick desde una función anónima que mostrara en un alert el texto del botón. Para ello:

- crearemos el botón como nodo
- modificaremos propiedad innerHTML para ponerle un texto
- asignaremos una función anónima para la gestión del evento
- lo añadiremos a la página dentro de body

```
<body>
</body>
<script>
  let miBoton=document.createElement("button");
  miBoton.innerHTML="Texto del boton";
  miBoton.onclick = function(){
                                    alert (this.innerHTML)
                                  };
  document.body.appendChild(miBoton);
</script>
```

En este caso para poder acceder a una propiedad del objeto en el que estamos hacemos referencia a el mismo a través de this.