Introducción

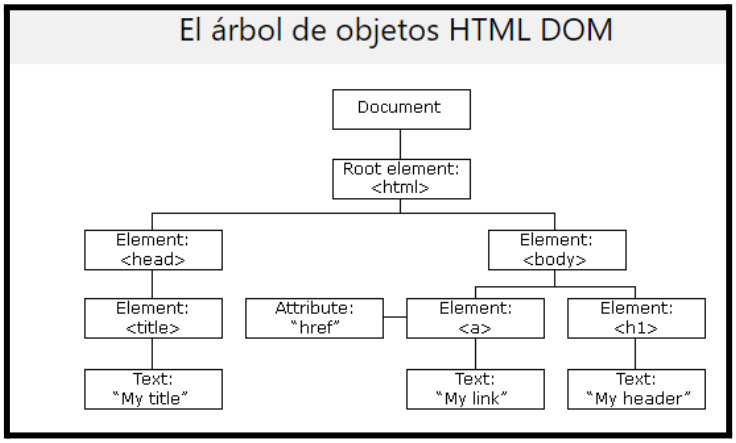
La W3C *(consorcio internacional que genera recomendaciones y estándares)* define el DOM (*Document Object Model o Modelo de Objetos del Documento)* como "una interfaz de programación de aplicaciones (API) para documentos HTML y XML." Es decir, el DOM define la estructura lógica de los documentos y el modo en que se accede y manipulan los mismos independientemente del lenguaje, lo que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, la estructura y el estilo de un documento.

En realidad el estándar DOM del W3C diferencia tres modelos diferentes:

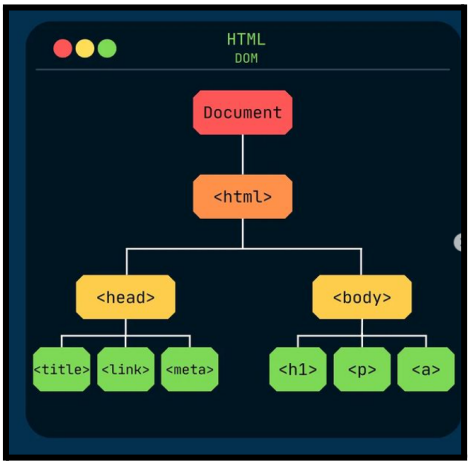
- **Core** DOM: modelo estándar para todos los tipos de documentos
- **XML** DOM: modelo estándar para documentos XML
- **HTML** DOM: modelo estándar para **documentos HTML**

El que a nosotros nos compete en esta UT es **HTML DOM**: estándar que define cómo obtener, cambiar, agregar o eliminar elementos HTML.

Cuando se carga una página web, el navegador crea un ***objeto document*** cuyo modelo DOM se ajusta al siguiente ***árbol de objetos:***

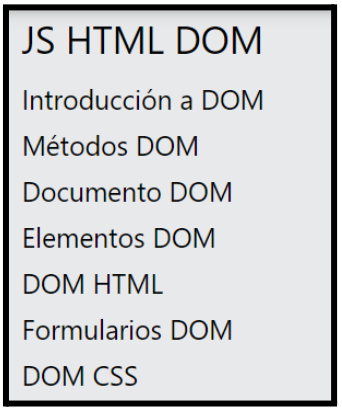


De forma más simplificada lo puedes ver así:



- En el DOM, todos los elementos HTML se definen como objetos. Así el ***objeto document*** representa nuestra página web y es el propietario de todos los demás objetos de la página. Conoces ya un método y una propiedad de este objeto:
  - ***getElementById*** es un **método** que nos permite localizar **un elemento** en el objeto document
  - ***innerHTML*** es una **propiedad.** La forma más sencilla de obtener el contenido de un elemento es mediante la propiedad innerHTML de dicho elemento.

Revisa en w3schools los capítulos de DOM que se muestran en la siguiente imagen:



- Con la lectura de los tres primeros apartados *(Introducción, Métodos y Document)* obtendrás una visión general de lo que JavaScript puede hacer con este modelo; puede acceder y cambiar todos los elementos de un documento HTML.

- Las siguientes tablas muestran un resumen de cómo se puede utilizar el ***objeto document*** para acceder y manipular HTML. Recuerda: *cada vez que quieras acceder a cualquier elemento en una página HTML, deberás empezar por acceder al* ***objeto document*** *(objeto del documento)*

Encontrar elementos HTML

Method	Description
document.getElementById(*id*)	Find an element by element id
document.getElementsByTagName(*name*)	Find elements by tag name
document.getElementsByClassName(*name*)	Find elements by class name

Cambiar elementos HTML

Property	Description
element.innerHTML = *new html content*	Change the inner HTML of an element
element.attribute = *new value*	Change the attribute value of an HTML element
element.style.*property* = *new style*	Change the style of an HTML element
Method	**Description**
element.setAttribute(*attribute, value*)	Change the attribute value of an HTML element

Agregar y eliminar elementos

Method	Description
document.createElement(*element*)	Create an HTML element
document.removeChild(*element*)	Remove an HTML element
document.appendChild(*element*)	Add an HTML element
document.replaceChild(*new, old*)	Replace an HTML element
document.write(*text*)	Write into the HTML output stream

Agregar controladores de eventos

Method	Description
document.getElementById(*id*).onclick = function(){*code*}	Adding event handler code to an onclick event

➢ En el apartado **"Elementos DOM"** vas a aprender más acerca de cómo localizar elementos HTML (recuerda que para poder manipularlos, primero has de localizarlos

en el document/página). El primero de los métodos ya lo conoces ***`getElementById(id)`*** pero verás que hay más formas de hacerlo:

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

Los tres primeros métodos corresponden a los **métodos tradicionale**s de Javascript para manipular el DOM. Se denominan tradicionales porque son los que existen en Javascript desde versiones más antiguas. Dichos métodos te permiten buscar elementos en la página dependiendo de los atributos id, name o de la propia etiqueta respectivamente:

.getElementById(id) busca un elemento HTML con el id especificado en id por parámetro. *En principio, un documento HTML bien construído no debería tener más de un elemento con el mismo id, por lo tanto, este método devolverá siempre un solo elemento. Ejemplo:*

```
const page = document.getElementById("page");   // <div id="page"></div>
```

En el caso de no encontrar el elemento indicado devolverá null.

.getElementsByClassName(class) busca los elementos con la clase especificada en class. Es importante darse cuenta del matiz de que el método contiene “getElements” en plural, y esto es porque al devolver clases (al contrario que los id) se pueden repetir, y por lo tanto devolver varios elementos, no sólo uno. Ejemplo:

```
const items = document.getElementsByClassName("item"); // [div, div, div]
console.log(items[0]);     // Primer ítem encontrado: <div class="item"></div>
console.log(items.length); // 3
```

.getElementsByTagName(tag) funciona exactamente igual, salvo que se encarga de buscar elementos HTML por su propio nombre de etiqueta HTML. Ejemplo:

```
const divs = document.getElementsByTagName("div");
// Obtiene todos los elementos <div> de la página
```

Los dos últimos métodos devuelven una lista con todos los elementos encontrados que encajen con el criterio (un tipo de dato HTMLCollection o NodeList). Ojo!, porque aunque actúan de forma muy similar a un array no lo son y por lo tanto pueden carecer de algunos métodos, como por ejemplo forEach()

Además de los métodos tradicionales que acabamos de ver, disponemos de métodos que nos permiten localizar elementos HTML mediante **selectores CSS** y mediante **colecciones de objetos:**

.querySelectorAll()
El siguiente ejemplo devuelve una lista de todos los elementos <p> con class="intro":

```
const x = document.querySelectorAll("p.intro");
```

Por último, el siguiente ejemplo encuentra el elemento de formulario con id="frm1" en la **colección** de formularios y muestra todos los valores del elemento:

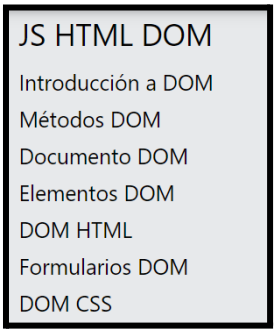
```
const x = document.forms["frm1"];
let text = "";
for (let i = 0; i < x.length; i++) {
  text += x.elements[i].value + "<br>";
}
document.getElementById("demo").innerHTML = text;
```

De igual manera se puede acceder a otros objetos y **colecciones** de objetos *(**document.links, document.images**...)*

Al realizar una búsqueda de elementos y guardarlos en una variable, podemos realizar la búsqueda posteriormente sobre esa variable en lugar de hacerla sobre document. Esto permite realizar búsquedas más acotadas por zona en lugar de realizarlas sobre document que buscará en todo el documento HTML. Ejemplo:

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
'El primer párrafo (índice 0) dentro de "main" es: ' + y[0].innerHTML;
```

Si has llegado hasta aquí, has debido revisar y entender en el tutorial de w3schools los cuatro primeros apartados de esta imagen



- Cuando ya sepas localizar un elemento, en el **apartado "DOM HTML"** aprenderás a cambiar el contenido de un elemento HTML:
  - podrás cambiar el **contenido html** de un elemento mediante el uso de la propiedad ***innerHTML .*** Sintaxis:
    `document.getElementById(id).innerHTML = nuevo HTML`
  - podrás cambiar el valor de un atributo HTML mediante: ***`document.getElementById(id).attribute = nuevo valor`***
  - recuerda que puedes escribir directamente sobre el documento con ***document.write ()*** *pero ojo!! con su uso: nunca uses document.write() después de cargar el documento ya que se sobrescribirá el documento.*

- Continúa con el **apartado "DOM CSS"** del tutorial, donde vas a aprender a cambiar desde Javascript el estilo aplicado a cualquier elemento.
  `document.getElementById(id).style.property = new style`

- Por último en el **apartado "Validación de formularios"** vas a ver una breve introducción a los posibles métodos de validación de un formulario.
  - validación del lado del servidor *(lo haremos en el segundo trimestre con ASP.net)*
  - validación del lado del cliente, donde has de distinguir:
    - validación mediante JavaScript
    - validación automática de formularios
    - validación de restricciones (HTML5)

Tarea:

1. Revisa y asegúrate de que entiendes y sabes hacer los **ejercicios y ejemplos propuestos** en cada uno de los apartados.

2. Cuando lo hayas hecho, prepara una página con una funcionalidad que exija poner en práctica lo aprendido. *No se trata de hacer un "copia-pega" de los ejemplos del tutorial, sino de "pensar" en una posible aplicación de los conocimientos adquiridos con ellos para lograr la funcionalidad que desees en tu página*. Sube a Moodle el trabajo realizado.