UT6_1.- JAVASCRIPT. AVANZADO - DOM

RESULTADOS DE APRENDIZAJE ASOCIADOS
1. Reconoce las características de lenguajes de marcas analizando e interpretando fragmentos de código. 2. Utiliza lenguajes de marcas para la transmisión de información a través de la Web analizando la estructura de los documentos e identificando sus elementos
CRITERIOS DE EVALUACIÓN
- De RA1 – desde CEA hasta CEK - De RA2 – desde CEA hasta CEH

UT6_1.- JAVASCRIPT. AVANZADO - DOM

Índice de contenido

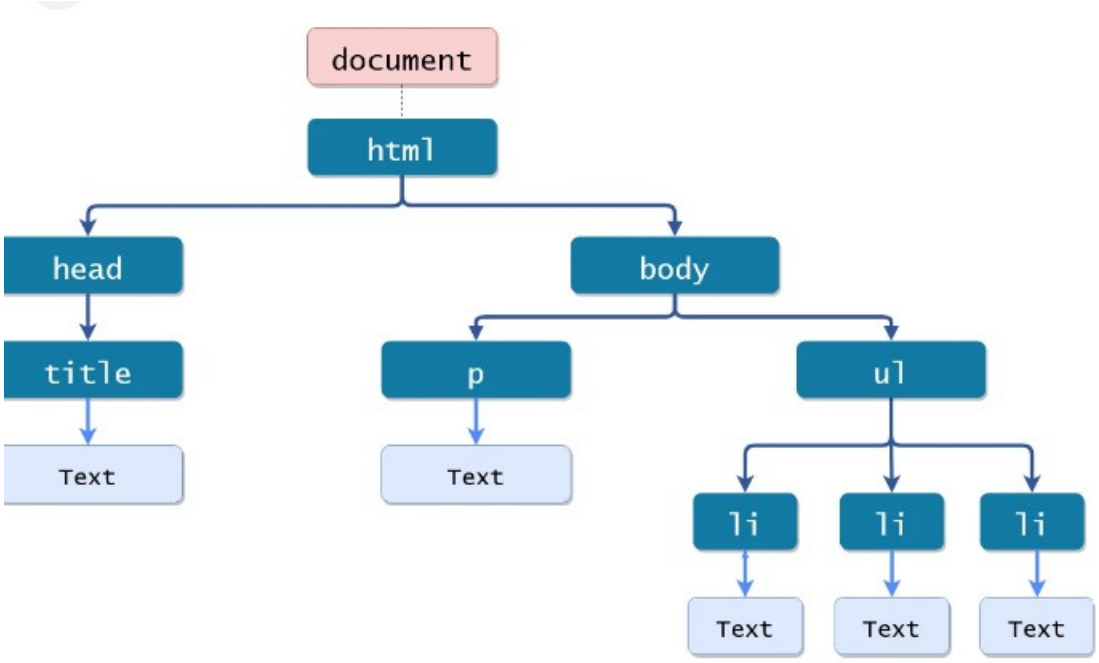
1.- Introducción....2
2.- Niveles del DOM....4
2.1.- Objetos del DOM HTML....4
2.2.- El árbol del DOM y tipos de nodos....4
2.3.- Tipos de nodos....6
2.4.- Acceso a los nodos....6
2.4.1.- GETELEMENTBYID()....7
2.4.2.- GETELEMENTSBYTAGNAME()....8
2.4.3.- GETELEMENTSBYCLASSNAME()....9
2.4.4.- QUERYSELECTOR() – QUERYSELECTORALL()....9
2.5.- Acceso a los atributos de un elemento....12
2.5.1.- Añadir y modificar atributos....13
2.5.2.- Obtener valor de un atributo....13
2.5.3.- Eliminar atributo....14

1.- Introducción

Las siglas **DOM** significan *Document Object Model*, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, **formando un**

árbol de etiquetas relacionadas entre sí, que se denomina **árbol DOM** (o simplemente DOM).

A través del DOM podemos acceder y modificar el contenido, estructura y estilo de los documentos HTML.



El responsable del DOM es el W3C

El DOM está separado en 3 partes / niveles:

- ✓ DOM Core modelo estándar para cualquier documento estructurado.
- ✓ DOM XML modelo estándar para documentos XML.
- ✓ DOM HTML modelo estándar para documentos HTML.

El DOM HTML es un estándar dónde se define cómo acceder, modificar, añadir o borrar elementos HTML.

En el DOM se definen los objetos y propiedades de todos los elementos del documento, y los métodos para acceder a ellos.

2.- Niveles del DOM

Cada vez es más frecuente encontrar literatura que habla de eventos DOM Level 0 o DOM Level 2 y actualmente incluso de DOM Level 3. Esto hace referencia a las distintas versiones del estándar DOM (Document Object Model) que se han ido definiendo y cómo estas versiones definen la creación y suscripción de los eventos JavaScript en una aplicación HTML.

2.1.- Objetos del DOM HTML

Esta es la sintaxis para acceder a las propiedades o métodos de aquellos objetos que estén dentro de nuestro documento:

```
document.getElementById(objetoID).propiedad | metodo( [parametros] )
```

Listado de objetos del DOM en HTML:

Document	HTMLElement	Anchor	Area	Base
Body	Button	Event	Form	Frame/IFrame
Frameset	Image	Input Button	Input Checkbox	Input File
Input Hidden	Input Password	Input Radio	Input Reset	Input Submit
Input Text	Link	Meta	Object	Option
Select	Style	Table	TableCell	TableRow
Textarea				

2.2.- El árbol del DOM y tipos de nodos

La tarea más habitual en programación web suele ser la manipulación de páginas web, para acceder a su contenido, crear nuevos elementos, hacer animaciones, modificar valores, etc.

Todas estas tareas se pueden realizar de una forma más sencilla gracias al DOM.

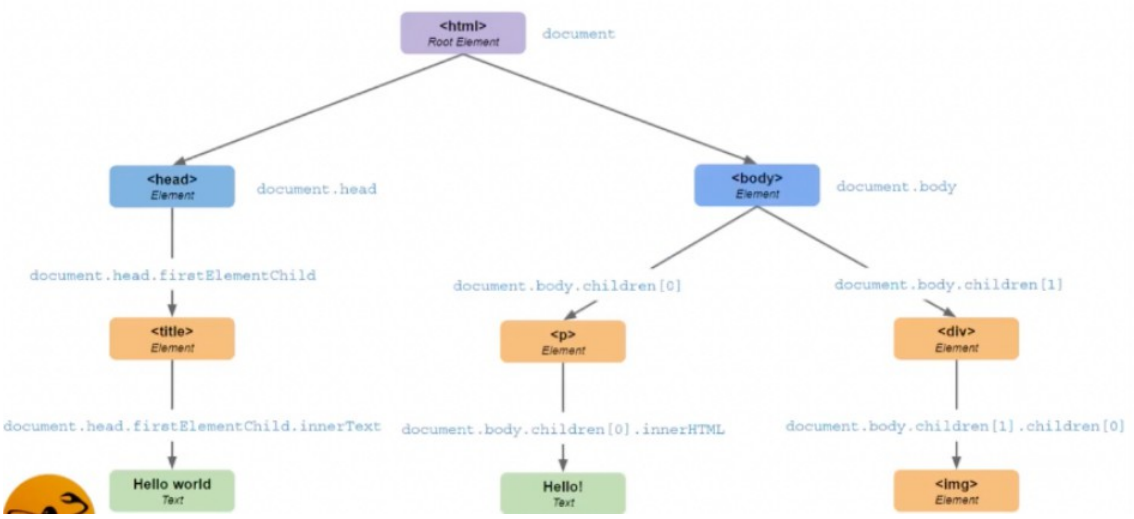
Los navegadores web son los encargados de realizar la transformación de nuestro documento, en una estructura jerárquica de objetos, para que podamos acceder con métodos más estructurados a su contenido.

El **DOM** transforma todos los documentos XHTML en un conjunto de elementos, a los que llama **nodos**. En el DOM **cada nodo es un objeto**.

Estos nodos están conectados entre sí y representan los contenidos de la página web, y la relación que hay entre ellos.

Cuando unimos todos estos nodos de forma jerárquica, obtenemos una estructura similar a **un árbol**, por lo que muchas veces se suele referenciar como árbol DOM, "**árbol de nodos**", etc.

Ejemplo:



Cada rectángulo del gráfico representa un nodo del DOM, y las líneas indican cómo se relacionan los nodos entre sí. La raíz del árbol de nodos es un nodo especial, denominado "**document**". A partir de ese nodo, cada etiqueta XHTML se transformará en nodos de tipo "**elemento**" o "**texto**".

Los nodos de tipo "texto", contendrán el texto encerrado para esa etiqueta XHTML.

Esta conversión se realiza de forma jerárquica.

El nodo inmediatamente superior será el nodo padre y todos los nodos que están por debajo serán nodos hijos.

2.3.- Tipos de nodos

La especificación del DOM define 12 tipos de nodos, aunque en general nosotros emplearemos solamente cuatro o cinco tipos de nodos:

- ✓ **Document**, es el nodo raíz y del que derivan todos los demás nodos del árbol.
- ✓ **Element,** representa cada una de las etiquetas XHTML. Es el único nodo que puede contener atributos y el único del que pueden derivar otros nodos.
- ✓ **Attr**, con este tipo de nodos representamos los atributos de las etiquetas XHTML, es decir, un nodo por cada atributo=valor.
- ✓ **Text,** es el nodo que contiene el texto encerrado por una etiqueta XHTML.
- ✓ **Comment,** representa los comentarios incluidos en la página XHTML.

2.4.- Acceso a los nodos

Cuando ya se ha construido automáticamente el árbol de nodos del DOM, ya podemos comenzar a utilizar sus funciones para acceder a cualquier nodo del árbol.

El acceder a un nodo del árbol, es lo equivalente a acceder a un trozo de la página de nuestro documento.

Así que, una vez que hemos accedido a esa parte del documento, ya podemos modificar valores, crear y añadir nuevos elementos, moverlos de sitio, etc.

Para acceder a un nodo específico (elemento XHTML) lo podemos hacer empleando dos métodos, bien **a través de los nodos padre**, o bien usando un método de acceso directo:

- ✓ A través de los nodos padre partiremos del nodo raíz e iremos accediendo a los nodos hijo, y así sucesivamente hasta llegar al elemento que deseemos.

- ✓ Y para el método de acceso directo, que por cierto es el método más utilizado, emplearemos funciones del DOM, que nos permiten ir directamente a un elemento sin tener que atravesar nodo a nodo

Algo muy importante que tenemos que destacar es, que, para poder acceder a todos los nodos de un árbol, el árbol tiene que estar completamente construido, es decir, cuando la página haya sido cargada por completo, en ese momento es cuándo podremos acceder a cualquier elemento de dicha página.

Consideremos el siguiente ejemplo y veamos las formas de acceso:

```
<input type="text" class=”input” id="apellidos" name="apellidos" />
```

2.4.1.- GETELEMENTBYID()

Esta función es la **más utilizada y rápida** cuando estamos accediendo a un elemento por su id. Entre paréntesis escribiremos la cadena de texto con el id.

Es necesario que el id sea único para cada elemento de una misma

página.

La función nos devolverá únicamente el nodo buscado.

Por ejemplo:

```
const apellidos= document.getElementById("apellidos");
```

Si tenemos por ejemplo una tabla con id="datos" y queremos acceder a todas las celdas de esa tabla, tendríamos que combinar *getElementById* con *getElementsByTagName*.

Por ejemplo:

```
const miTabla= document.getElementById("datos");
const filas= miTabla.getElementsByTagName("td");
```

2.4.2.- GETELEMENTSBYTAGNAME()

Esta función es muy similar a la anterior y también devuelve una colección de elementos cuya etiqueta coincida con la que se pasa como parámetro. Por ejemplo:

```
// Este array de elementos contendrá todos los elementos input del documento.
let elementos = document.getElementsByTagName("input");

// Si ya hemos hecho un getElementsByTagName

let cuarto = elementos[3];
```

```
// Haciéndolo todo a la vez en una sola línea
let cuarto = document.getElementsByTagName("input")[3];
```

2.4.3.- GETELEMENTSBYCLASSNAME()

Esta función es muy similar a la anterior y también devuelve una colección de elementos cuya clase coincida con la que se pasa como parámetro. Por ejemplo:

```
// Este array de elementos contendrá todos los elementos con la clase "nombreClase"
        del documento.
let elementos = document.getElementsByClassName("nombreClase");

let segundo = document.getElementsByClassName("nombreClase")[1];

// Si ya hemos hecho un getElementsByClassName
let segundo = elementos[1];
```

2.4.4.- QUERYSELECTOR() – QUERYSELECTORALL()

Ambos nos permiten seleccionar elementos del DOM, la diferencia es que:

- **QuerySelector()** solo selecciona el primer elemento que encuentre.

- **QuerySelectorAll()** selecciona todos los elementos que encuentre.

```
<div id="prueba">
    <p id="id1" class="posicion" title="Azul">
      Primero
    </p>
    <p id="id2" class="posicion" title="Verde">
      Segundo
```

```
            </p>
            <p id="id3" class="posicion" title="Naranja">
               Tercero
            </p>
            <p id="id4" class="posicion" title="Lila">
               Cuarto
            </p>
            <p id="id5" class="posicion" title="Rojo">
               Quinto
            </p>
         </div>
```

QuerySelector() lo podemos utilizar en sustitución de **getElementById(),** aunque cuando vamos hacer una selección por id, normalmente siempre se utiliza **getElementById**, ya que es más rápido:

```
let primeroid = document.getElementById("id5");
```

o bien

```
let primeroquerySelector = document.querySelector("#id5");
```

NOTA: Obtenemos un objeto con el elemento que tiene el id id5.

La forma que tiene **document.querySelector("#id5")** *de* ***indicar que es un id es poniendo una almohadilla antes del nombre del identificador.***

(“#id5”)

querySelectorAll() lo vamos a utilizar en sustitución de **getElementsByTagName() getElementsByClassName().** Nos va a devolver una colección (NodeList) con todos elementos que tengan la clase o la etiqueta seleccionados. Para seleccionar clases y etiquetas se utiliza querySelectorAll

```
let primeroClass =document.getElementsByClassName(“posicion”);
```

o bien

```
let primeroqueryClass=document.querySelectorAll(“.posicion”);
```

Obtenemos una colección con todos los elementos que tengan la clase posicion.

La forma que tiene **document.querySelectorAll(“.posicion”) de *indicar que es una clase es poniendo un punto antes del identificador de la clase.* (“.posicion”)**

```
let primeroTag = document.getElementsByTagName(“p”);
```

o bien

```
let primeroquerySelector = document.querySelectorAll("p");
```

Obtenemos una colección con todos los elementos que tengan con la etiqueta p.

La forma que tiene* document.querySelectorAll("p")*de indicar que es una clase es indicando el nombre de la etiqueta. ("p")

2.5.- Acceso a los atributos de un elemento

Hasta ahora hemos visto que podemos acceder a un atributo a través del elemento al que pertenece, y a partir de ahí, podemos leer y modificar el contenido de un atributo.

```
let elemento = document.getElementById ("id5");
elemento.title = "Nuevo título";
console.log(elemento.title);
```

Además de esta forma de trabajar **JavaScript nos permite leer, escribir, crear, modificar y borrar los atributos de un elemento, mediante una serie de métodos.**

2.5.1.- Añadir y modificar atributos

- **setAttribute()** nos permite tanto la modificación como la creación de atributos.

```
elemento.setAttribute("title", Nuevo título);
```

En este caso si el atributo title no existe se crea y si existe se modifica.

Por ejemplo, para el siguiente HTML:

```
<form action="#" method="GET">
    <label>Nombre(*):</label>
        <input id="nombre" type="email" name="nombre" required>
    <label>Apellidos(*):</label>
      <input id="apellidos" type="email" name="apellidos" required>
 </form>
```

Le queremos poner al campo "apellidos" **type='text'** y un **value = "Fernández y Martínez"**, haríamos*:*

```
let apellidos = document.getElementById("apellidos");
apellidos.setAttribute("type","text");
apellidos.setAttribute("value","Fernández y Martínez");
```

2.5.2.- Obtener valor de un atributo

- **getAttribute()** nos permite leer el contenido de un atributo.

```
let apellidos = document.getElementById("apellidos”);
let valor = apellidos.getAttribute('type');
console.log(valor);
```

2.5.3.- Eliminar atributo

- **removeAttribute()** nos permite eliminar un atributo.

```
let apellidos = document.getElementById("apellidos”);
apellidos.removeAttribute(“placeholder”);
```