DOM y formularios

Laura Isabel López Naves

Creación de nodos: createElement() y createTextNode()

- Para crear nodos se puede utilizar:

  1.- createElement(): para nodos de tipo element (o tag). Ej: document.createElement('DIV');

  2.- createTextNode(): para nodos de tipo texto. Ej: document.createTextNode('El texto');

- Para añadir el nodo creado a otro nodo se pueden utilizar:
  - **node.append(...nodes or strings)** – añade nodes o strings al final de node.(*)
  - **node.prepend(...nodes or strings)** – inserta nodes o strings al principio de node.
  - **node.before(...nodes or strings)** – inserta nodes o strings antes de node (fuera de el).
  - **node.after(...nodes or strings)** – inserta nodes o strings después de node (fuera de el).

```
<div id="id0" style="border:solid 1px black">Esto es un contenedor</div>
<script>
    const donde = document.querySelector('#id0');
    const e1 = document.createElement('DIV');
    e1.innerHTML="<strong>Prueba1</strong>";
    donde.append(e1);
    const e2 = document.createElement('DIV');
    e2.innerHTML="<strong>Prueba2</strong>";
    donde.prepend(e2);
```

Prueba2
Esto es un contenedor
Prueba1

(*) También se puede utilizar appendChild() pero solo permite añadir nodos de tipo element (no permite strings).

Eliminar y clonar nodos: remove() y cloneNode()

- Eliminar un nodo consiste en localizar el nodo y llamar a **remove()**.
- Para clonar un nodo hay que llamar a **cloneNode()** que nos devuelve el nodo clonado:
  - **node.cloneNode(true)**: hace una copia profunda (copia atributos y subnodos).
  - **node.cloneNode(false):** hace una copia superficial (solo copia el nodo actual).

```
<div id="id0" style="border:solid 1px black">Esto es un contenedor</div>
<script>
    const d = document.querySelector('#id0');
    const copiaProfunda=d.cloneNode(true);
    d.after(copiaProfunda);
    const copiaSuperficial=d.cloneNode(false);
    d.after(copiaSuperficial);
    d.remove();
    console.log(copiaProfunda.outerHTML);
    console.log(copiaSuperficial.outerHTML);
</script>
```

```
<div id="id0" style="border:solid 1px black">Esto es un contenedor</div>
<div id="id0" style="border:solid 1px black"></div>
```

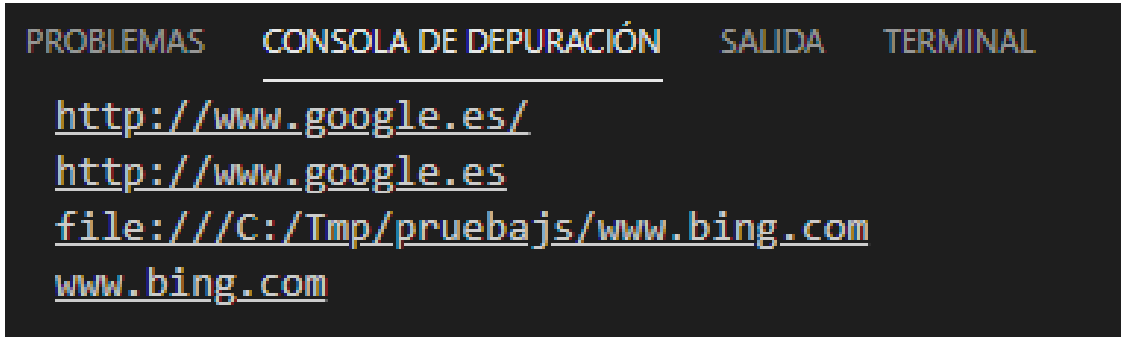
Salida por consola

Atributos y propiedades

- Cuando el navegador carga y parsea una página, automáticamente convierte los atributos HTML en propiedades de los objetos del DOM.
- Por ejemplo, el atributo “href” de una etiqueta “<A>” se convierte en una propiedad “href” para dicho elemento.
- Se pueden acceder o modificar las propiedades directamente o los atributos mediante **getAttribute()** y **setAtribute():**

```
<a></a>
<script>
    const r = document.querySelector("a");
    r.href="http://www.google.es";
    console.log(r.href);
    console.log(r.getAttribute('href'));
    r.setAttribute("href","www.bing.com");
    console.log(r.href);
    console.log(r.getAttribute('href'));
```

El valor almacenado en el atributo puede ser distinto al valor de la propiedad. Ejemplo:



Nota: de aquí se deduce que siempre hay que utilizar el protocolo de comunicación (HTTP, FILE, etc.) en los atributos o propiedades href.

Resumen DOM

- Los tipos de nodos más importantes son:
  - **Document**: representa el nodo raíz (todo el documento HTML).
  - **Element**: representa una etiqueta (puede contener hijos de tipo Element).
  - **Attr**: representa un attributo de un elemento.
  - **Text**: representa texto (aparece como subnodo de un Element).
- Api para la creación y manipulación del DOM:
  - Ver las páginas 79,80 y 81 de "Aprendiendo JavasScript" de Carlos Azaustre.
  - Referencia: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- Ej:

```
const colores = ['Verde', 'Azul', 'Rojo'];
let i = 1;
const fragment = new DocumentFragment()
colores.forEach(function (color) {
    let l = document.createElement('li');
    l.value = i++ * 10;
    l.innerHTML = color;
    fragment.appendChild(l);
})
const lista = document.querySelector('#listaColores');
lista.appendChild(fragment);
```

10. Verde
20. Azul
30. Rojo

En el ejemplo se utiliza un objeto "DocumentFragment" para construir los elementos de una lista ordenada independiente de "document" . "DocumentFragment" evita que el navegador tenga que procesar cada elemento individualmente permitiendo mas eficiencia.

Ejercicio 2 (ej2.html)

- Crear una página web que, después de cargarse, realice las siguientes tareas utilizando los objetos del DOM:
  - Crear una matriz de strings con URLs de diferentes sitios de Internet.
  - Crear una lista desordenada (“ul”).
  - Crear un nuevo elemento (“li”) por cada enlace y dentro de cada elemento crear un enlace (“a”) que contenga una URL.
  - Añadir los elementos a la lista desordenada.
  - Añadir la lista al final del documento.
- Ejemplo:

Ejercicio 3 (ej3.html)

- En una página web tenemos la siguiente capa:

```
<div id="principal" style="border:solid 1px black;
        height:300px; width:700px; margin: 0 auto; background:orange;">
</div>
```

- Realizar las siguientes tareas utilizando la API DOM:
  - Crear una matriz con seis mensajes publicitarios.
  - Borrar todos los elementos (si existen) en la etiqueta “principal” .
  - Crear seis capas dentro de “principal” que contengan los mensajes publicitarios, en una posición aleatoria y de un color aleatorio.
  - Repetir todo el proceso cada segundo.
- El formato debe ser similar a :