Capítulo 2

HTML y JavaScript

2.1. Elementos HTML (formularios)

HTML, como ya sabemos, es un lenguaje de marcas cuyo propósito principal consiste en estructurar el contenido de las páginas web.

HTML nos va a permitir crear formularios para que los usuarios interactúen con las aplicaciones web.

```
<form action="ejemplo.php" method="get">
      <p>Nombre:  <input  type="text"  name="nombre"
size="40"></p>
      <p>Año  de  nacimiento:  <input  type="number"
name="nacido" min="1900"></p>
  <p>Sexo:
    <input type="radio" name="hm" value="h"> Hombre
    <input type="radio" name="hm" value="m"> Mujer
  </p>
  <p>
    <input type="submit" value="Enviar">
    <input type="reset" value="Borrar">
  </p>
</form>
```

Nombre:

Año de nacimiento:

Sexo: Hombre Mujer

Enviar Borrar

Figura 2.1: Formularios HTML

El DOM (Modelo de Objetos del Documento) es la **estructura de objetos que genera el navegador cuando se carga un documento**. Los objetos del DOM modelizan tanto la ventana del navegador como el historial, el documento o página web, y todos los elementos que pueda tener dentro la propia página, como párrafos, divisiones, tablas, formularios y sus campos.

https://www.youtube.com/watch?v=mg2qaADt8d0 (8:31)

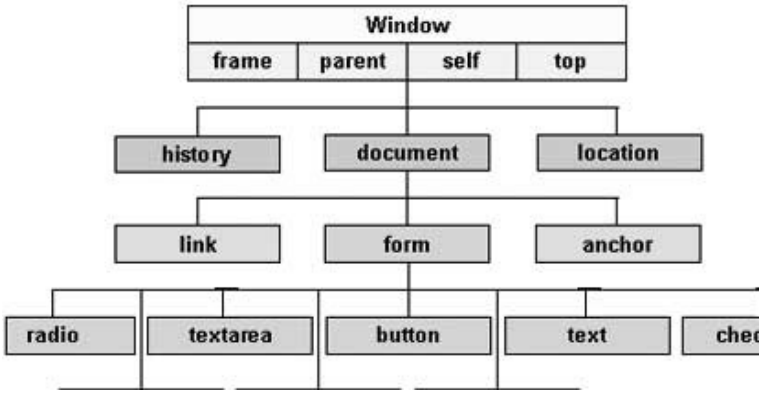


Figura 2.2: DOM

https://developer.mozilla.org/es/docs/Web/API/Document

Figura 2.3: Imagen representativa del DOM extraida de https://lenguajejs.com/javascript/dom/que-es/

Accediendo al objeto document, a través de los atributos **id** y **name** podemos acceder a los elementos de un formulario desde javascript, usando los métodos **getElementById**('valorDelId') y **getElementsByName**('name'). También existe la posibilidad de trabajar con todos los elementos de un mismo tipo a través del método **getElementsByTagName** y de todos los elementos que tienen el mismo valor en la propiedad class a través de **getElementsByClassName**.

Hay que tener en cuenta que siempre que seleccionamos más de un elemento javascript crea un objeto similar a un array.

También existe document.**querySelector** que devuelve el primer elemento del documento que coincide con el grupo especificado de selectores CSS. Si queremos todos los elementos podemos usar **querySelectorAll**.

Tipo de Selector	Selector	Descripción
Selector de ID	`#mi-id`	Selecciona el elemento con el `id` especificado.
Selector de Clase	`.mi-clase`	Selecciona el primer elemento con la clase especificada.
Selector de Etiqueta	`div`	Selecciona el primer elemento por su nombre de etiqueta (por ejemplo, `div`, `p`, `h1`, etc.).
Selector de Atributo	`[type]`	Selecciona el primer elemento que tenga el atributo especificado (por ejemplo, `type`).
Atributo con Valor Específico	`[type="text"]`	Selecciona el primer elemento con un atributo que tenga el valor especificado.
Selector de Descendiente	`div p`	Selecciona el primer elemento `p` que es descendiente de un `div`.
Selector de Hijo Directo	`div > p`	Selecciona el primer `p` que es un hijo directo de `div` (no descendiente en cualquier nivel).
Selector de Hermano Adyacente	`h1 + p`	Selecciona el primer `p` que sigue inmediatamente a `h1`.
Selector de Hermanos Generales	`h1 ~ p`	Selecciona todos los elementos `p` que son hermanos de `h1`, no necesariamente inmediatos.
Selector Universal	`*`	Selecciona el primer elemento de cualquier tipo.
Selector de Grupo	`div, p, span`	Selecciona el primer elemento que sea `div`, `p` o `span`.
Selector de Pseudo-clase	`a:hover`	Selecciona el primer enlace (`a`) cuando se pasa el ratón sobre él (`hover`).
Selector de Pseudo-elemento	`p::first-line`	Selecciona la primera línea del texto de un párrafo.
Selector de Negación	`:not(.clase)`	Selecciona el primer elemento que no tenga la clase especificada.

Figura 2.4: Selectores CSS

```
1
2 document.getElementById('idElemento');
3
4 document.getElementsByTagName('tagname')[indice];
5
6 document.querySelector("p");
7
8 document.querySelector('#mi-boton'); //id
```

2.2. Eventos

JS define numerosos eventos que permiten una interacción completa entre el usuario y las páginas/aplicaciones web. El pulsar una tecla constituye un evento, así como pinchar o mover el ratón, seleccionar un elemento de un formulario, redimensionar la ventana del navegador, etc.

JS permite asignar una función a ejecutar a cada uno de los eventos. De esta forma, cuando se produzca dicho evento, JS ejecutará la función asociada. Este tipo de funciones se denominan *event handlers* en inglés y suelen traducirse por *manejadores de eventos*.

Lista de eventos

onload: es lanzado cuando se termina de cargar una página.

onunload: es disparado cuando se retira el documento de una ventana o marco.

onclick: ocurre cuando se realiza un click sobre un elemento.

ondblclick: es ejecutado cuando se hace un doble click sobre el elemento.

onmousedown: es lanzado cuando el botón del ratón es presionado sobre el elemento (independientemente de que sea soltado o no).

onmouseup: es disparado cuando el botón del ratón se suelta sobre el elemento.

onmouseover: ocurre cuando el ratón es puesto sobre el elemento.

onmousemove: es ejecutado cuando el ratón es movido mientras está sobre el elemento.

onmouseout: es lanzado cuando el ratón se quita de encima de un documento.

onfocus: es disparado cuando un elemento recibe el foco o cursor, bien sea a través del ratón o por navegación tabulada.

onblur: ocurre cuando el elemento pierde el foco bien sea a través del ratón o por navegación tabulada.

onkeypress: es ejecutado cuando una tecla es presionada y luego soltada mientras el elemento tiene el cursor.

onkeydown: es lanzado cuando una tecla es presionada (independientemente de que sea soltada o no) mientras el elemento tiene el enfoque.

onkeyup: es disparado cuando una tecla es soltada mientras el elemento tiene el enfoque.

onsubmit: ocurre cuando el formulario es enviado.

onreset: es ejecutado cuando el formulario es restablecido a sus valores por defecto.

onselect: es lanzado cuando un usuario selecciona texto en un campo de texto.

onchange: es disparado cuando un control pierde el cursor y su valor ha sido modificado desde que recibió el cursor por última vez.

onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>,<textarea>, <body>
onchange	Deseleccionar un elemento modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todas
ondblclick	Pinchar dos veces seguidas con el ratón	Todas
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>,<textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Etiquetas de formulario y <body>
onkeypress	Pulsar una tecla	Etiquetas de formulario y <body>
onkeyup	Soltar una tecla pulsada	Etiquetas de formulario y <body>
onload	La página se ha cargado completamente	<body>
onmousedown	Pulsar (sin soltar) un botón del ratón	Todas las Etiquetas
onmousemove	Mover el ratón	Todas las Etiquetas
onmouseout	El ratón "sale" del elemento (pasa por encima de otro elemento)	Todas las Etiquetas
onmouseover	El ratón "entra" en el elemento (pasa por encima del elemento)	Todas las Etiquetas
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todas las Etiquetas
onreset	Inicializar el formulario	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>

Figura 2.5: Eventos

Ejemplo uno (Evento onclick en la etiqueta que define el botón):

```
1
2  <!DOCTYPE html>
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <title>Ejemplo uno</title>
7  </head>
8  <body>
9      <form>
10         <p>Nombre: <input type="text" id="nombre" size="40"
               class="c1"></p>
11         <p>Año de nacimiento: <input type="number" id="nacido"
               min="1900"></p>
12         <p>Sexo:
13             <input type="radio" name="hm" value="h"> Hombre
14             <input type="radio" name="hm" value="m"> Mujer
15         </p>
16         <input type = "button" class = "c1" value = "Aceptar"
               onclick="visualizar()">
17
18     </form>
19     <script src="ejemplo1.js"></script>
20 </body>
21 </html>
```

```
1
2  function visualizar(){
3      let nombre = document.getElementById("nombre").value;
4      alert(" El contenido de la caja de texto nombre es "+ nombre );
5
6      let sexo = document.getElementsByName("hm");
7      for(let x = 0; x < sexo.length; x++)
8      {
9          alert(sexo[x].value + " " +sexo[x].checked);
10     }
11
12     let clases = document.getElementsByClassName("c1");
13
14     console.log("Elementos seleccionados por class");
15     Array.from(clases).forEach(elemento => console.log(elemento.value));
16     // clases.forEach(e => document.write(e)) NO FUNCIONA
17
18     console.log("Elementos seleccionados por etiqueta");
19     let inputs= document.getElementsByTagName("input");
20     Array.from(inputs).forEach(element => {
21         console.log(element.value);
22     });
23 }
```

https://stackoverflow.com/questions/3871547/iterating-over-result-of-getelementsbyc

```
1
2  Ejemplo uno, segunda forma (En la etiqueta asociada al boton no se pone
       onclick. Se pone en el código.):
3
4
5  <input type = "button" class = "c1" value = "Aceptar" id="bAceptar">
6
7  Fichero js
8
9  document.getElementById("bAceptar").onclick = visualizar;
10 //¡¡Sin paréntesis!! si se ponen paréntesis se ejecuta la función
11
12 function visualizar()
13 {
14 ......................
15 }
```

Añadir o eliminar eventos en el código

El método **addEventListener** en JavaScript se utiliza para asignar eventos a elementos del DOM.

La sintaxis es:

```
1
2  elemento.addEventListener(evento, funcion);
3
4  evento: Es el tipo de evento que deseas escuchar, como 'click',
       'mouseover', 'keydown', 'submit', etc.
5
6  funcion: Es la función que se ejecutará cuando el evento ocurra. Puede
       ser una función anónima o una función previamente definida.
7
8  boton.addEventListener('click', funcion1);
9  boton.addEventListener('click', funcion2);  // Ambos eventos se ejecutan
10
11
12 // Evento cuando el ratón pasa sobre la caja seleccionada previamente.
13         caja.addEventListener('mouseover', function() {
14             caja.style.backgroundColor = 'lightgreen';
15         });
```

El método **removeEventListener** en JavaScript se utiliza para eliminar un evento previamente añadido a un elemento con addEventListener. Esto permite que dejes de escuchar un evento específico en un elemento del DOM, evitando que la función asignada al evento se ejecute.

```
1
2  elemento.removeEventListener(evento, funcion);
3
4  evento: El tipo de evento que deseas eliminar, como 'click', 'mouseover',
       'keydown', etc.
5
6  funcion: La misma función que se utilizó al añadir el evento con
       addEventListener. Es importante que la función sea exactamente la
```

```
    misma que la usada para añadir el evento, ya que si no es la misma
    referencia de función, el evento no se eliminará.
```

```
1
2  <!DOCTYPE html>
3  <html lang="es">
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Ejemplo de removeEventListener</title>
8  </head>
9  <body>
10     <button id="miBoton">Haz clic aquí</button>
11
12     <script>
13         const boton = document.querySelector('#miBoton');
14
15         // Definimos una función que muestra una alerta
16         function mostrarAlerta() {
17             alert('¡Has hecho clic en el botón!');
18             // Eliminar el evento de clic después de que se haga clic una
                   vez
19             boton.removeEventListener('click', mostrarAlerta);
20         }
21
22         // Añadimos el evento de clic
23         boton.addEventListener('click', mostrarAlerta);
24     </script>
25 </body>
26 </html>
```

```
Ejemplo uno, tercera forma:

document.getElementById("bAceptar").addEventListener("click",visualizar);

function visualizar()
{
...........
}
```

Objeto de tipo Evento

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Eventos. Obtener información de un evento</title>

</head>

<body>
<h1 id="eventos">Obtener información de un evento</h1>
<h2 id="parrafo1">Párrafo 1</h2>
<h2 id="parrafo2">Párrafo 2</h2>
<script>
    document.getElementById("eventos").addEventListener("mouseover",
        manejador);
    document.getElementById("eventos").addEventListener("mouseout",
        manejador);
    document.getElementById("parrafo1").addEventListener("click", saludo);
    document.getElementById("parrafo2").addEventListener("click", saludo);

    function manejador(e) {
         switch (e.type) {
            case "mouseover":
                this.style.color = "purple";
                break;
            case "mouseout":
                this.style.color = "yellow";
                break;
        }
    }

    function saludo(e) {
        if (e.target.id == "parrafo1")
            alert("Has pulsado el primer párrafo");
        else
            if (e.target.id == "parrafo2")
                alert("Has pulsado el segundo párrafo");

        alert("Has pulsado el " + e.target.id);

        // target devuelve una referencia al objetivo en la cual el
            evento fue originalmente enviado.
    }
</script>
```

Más información sobre el contenido del objecto event en https://developer.mozilla.org/es/docs/Web/API/Event

2.3. Árbol de nodos DOM

El navegador transforma todos los documentos HTML en un conjunto de elementos llamados **nodos**, que están interconectados. Por su aspecto, la unión de todos los nodos se llama **árbol de nodos**.

Ejemplo:

```
1
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4  <html xmlns="http://www.w3.org/1999/xhtml">
5       <head>
6         <meta http-equiv="Content-Type" content="text/html;
              charset=iso-8859-1" />
7         <title>Página sencilla</title>
8       </head>
9
10      <body>
11         <p>Esta página es <strong>muy sencilla</strong></p>
12      </body>
13 </html>
```

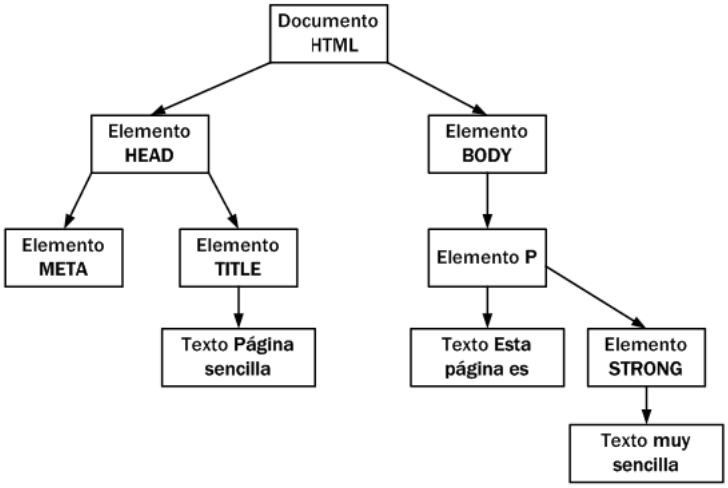


Figura 2.6: Árbol de nodos generado con el HTML anterior

Cada rectángulo de la imagen representa un nodo y las flechas indican las relaciones entre ellos. Dentro de cada nodo, se ha incluido su tipo y su contenido.

2.3.1. Nodos

https://dom.spec.whatwg.org/#interface-node

Cada nodo en el árbol DOM es un objeto (propiedades, métodos) de tipo **Node**. Los nodos mantienen relaciones con sus nodos inmediatamente vecinos, y contienen una gran cantidad de información sobre ellos mismos.

La propiedad **parentNode** nos da acceso al nodo padre y **childNodes** a los nodos hijo. También existe **children** que nos muestra sólo los hijos de tipo Element (etiquetas html). Los nodos, en general, pueden tener cualquier número de hijos por lo que las propiedades childNodes y children devuelven una **colección** (similar pero no igual que un array). childNodes devuelve una **NodeList** y children un **HTMLCollection**. Cada elemento es un hijo, en el mismo orden que aparecen en el documento.

```
1
2  Ejemplo dos:
3
4  <!DOCTYPE html>
5  <html>
6  <head>
7      <title>Ejemplo uno DOM</title>
8  </head>
9  <body>
10     <form  id = "f1">
11             <h1>FORMULARIO xxx </h1>
12
13             <label for="apellido">Apellido: *</label>
14             <input type="text" id="apellido"><br><br>
15
16             <input type="button" value="Enviar" id="enviar">
17     </form>
18
19     <script>
20
21     </script>
22 </body>
23 </html>
```

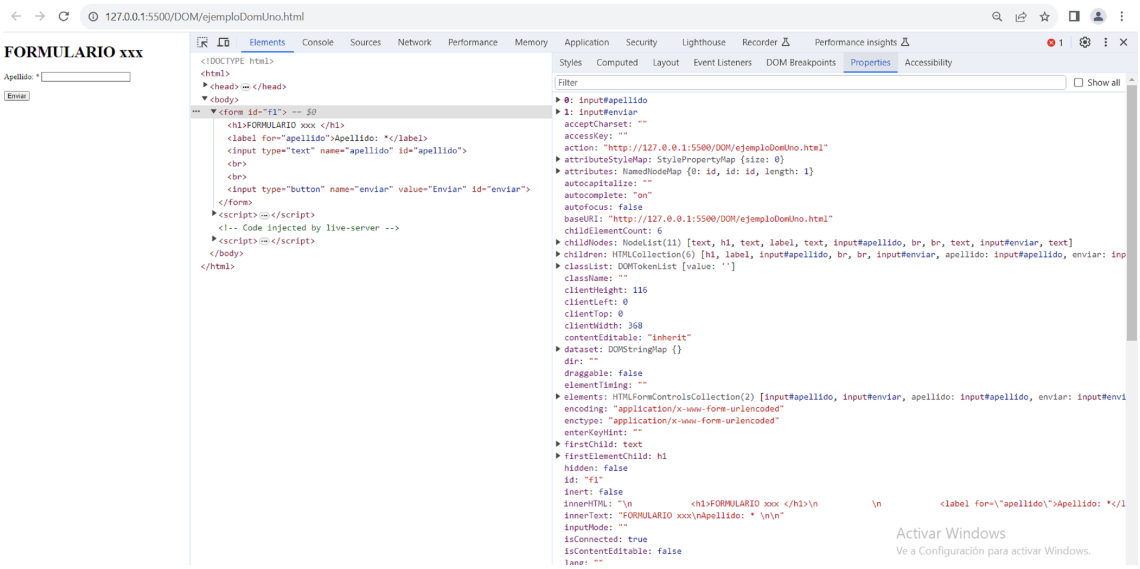


Figura 2.7: Elements - Properties

Figura 2.8: console.dir (NodeList y HTMLCollection)

2.3.2. Tipos de nodos

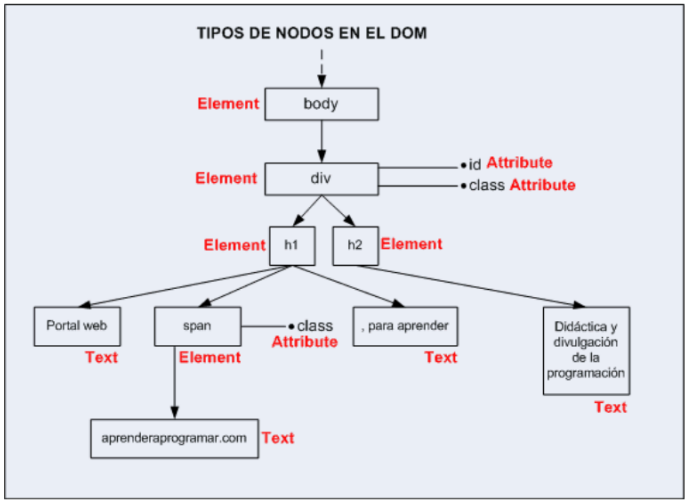


Figura 2.9: Árbol con diferentes tipos de nodos. Imagen extráida de https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=802:tipos-de-nodos-dom-document-element-text-attribute-comment-arbol-de-nodos-para-java&catid=78&Itemid=206

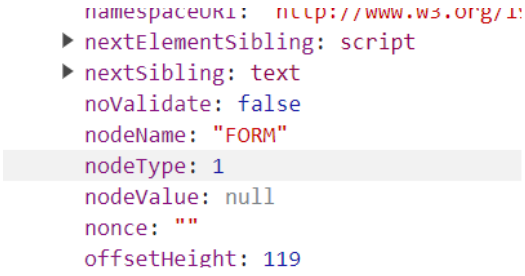


Figura 2.10: Tipo de nodo

Tipo de nodo	`nodeType`	Descripción
`Document`	9	Representa el documento completo
`Element`	1	Representa una etiqueta HTML o XML
`Text`	3	Contiene el texto dentro de un elemento
`Attribute`	2	Representa un atributo de un elemento (aunque hoy se usa más como propiedad)
`Comment`	8	Representa un comentario `<!-- -->`
`DocumentFragment`	11	Contenedor temporal para agrupar nodos antes de insertarlos en el DOM

Figura 2.11: NodeType

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas habituales no usan todos. Los tipos principales de nodos en el DOM:

- **Document (nodo documento)**

  Es la raíz del árbol DOM. Representa el documento completo. Permite acceder a todos los demás nodos.

- **Element (nodo de elemento)**

  Representa etiquetas HTML o XML. Son los más comunes (div, p, h1, span, etc.). Pueden tener atributos y nodos hijos.

```
1
2 Ejemplo: <p>Hola</p>
```

- **Text (nodo de texto)**

  Contiene el texto dentro de un nodo de elemento.

```
1
2 Ejemplo: en <p>Hola</p>, el "Hola" es un nodo de texto.
```

- **Attribute (nodo de atributo)**

  Representa un atributo de un elemento.

```
1
2 Ejemplo: en <img src="foto.png">, src="foto.png" es un nodo de
      atributo.
```

  Hoy en día se accede más como propiedades de los elementos, no tanto como nodos independientes.

- **Comment (nodo de comentario)**

  Representa un comentario en el código.

```
1
2 Ejemplo: <!-- Esto es un comentario -->
```

  DocumentFragment (fragmento de documento)

  Contenedor *ligero* para agrupar nodos temporalmente.

  Muy útil para manipular varios elementos antes de insertarlos en el DOM.

2.3.3. Recorrido de nodos

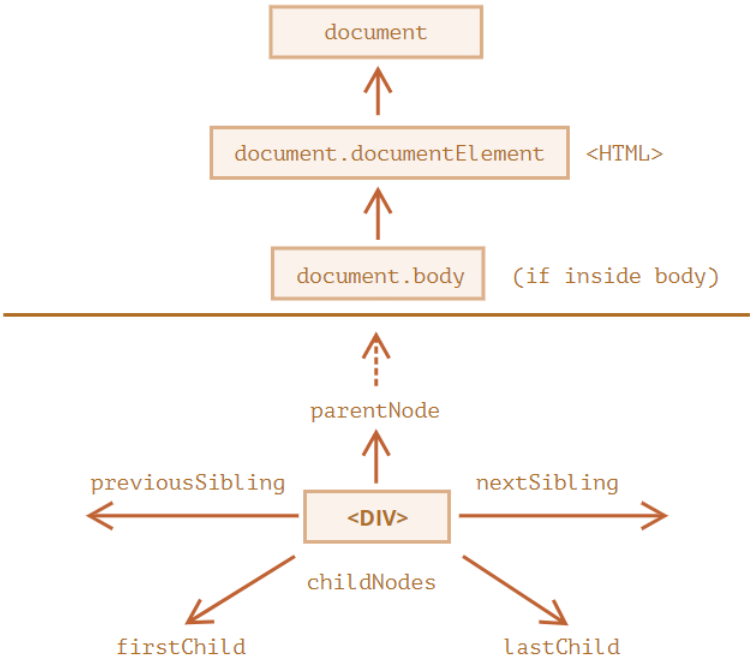


Figura 2.12: Recorrido. Imagen extraída de https://es.javascript.info/basic-dom-node-properties

Dado un nodo del DOM (métodos get o querySelector vistos anteriormente) , podemos ir a sus inmediatos vecinos utilizando las propiedades de navegación (parentElement, children, firstElementChild, lastElementChild, previousElementSibling, nextElementSibling, ...)

Algunos tipos de elementos del DOM, por ejemplo las tablas, proveen propiedades adicionales y colecciones para acceder a su contenido.

Ejemplo:

```
1
2  Ejemplo tres:
3
4  <!DOCTYPE html>
5  <html>
6  <head>
7      <title>Ejemplo tres</title>
8  </head>
9  <body>
10     <form>
11             <h1>FORMULARIO xxx </h1>
12
13             <label for="apellido">Apellido: *</label>
```

```
14              <input type="text" name="apellido" id="apellido"><br><br>
15
16              <label for="codigoPostal">Codigo postal: *</label>
17              <input type="number" name="codigoPostal"
                    id="codigoPostal"><br><br>
18
19              <label for="mensaje">Mensaje:</label><br>
20              <textarea id="mensaje"></textarea><br><br>
21
22              <input type="checkbox" name="chPolitica" id="politica">
23              <label for="politica">He leido y acepto la Politica de
                    Privacidad *</label><br><br>
24
25              <input type="button" name="enviar" value="Enviar" id="enviar">
26      </form>
27      <p> Contenido del párrafo </p>
28      <script>
29          // Accedemos al body
30          let cuerpo = document.body; // lastElementChild
31
32          // Hijos del body con chilNodes
33          let hijosBody1 =cuerpo.childNodes;  // childNodes devuelve
                nodelist
34          console.log("Número de hijos con childnodes " +
                hijosBody1.length);
35
36          console.log("Hijos con childNodes");
37          hijosBody1.forEach((hijo)=>console.log(hijo));
38
39          // Hijos del body con children
40          let hijosBody2 =cuerpo.children; // devuelve HTMLCollection
41          console.log("Número de hijos con children: " + hijosBody2.length);
42
43          console.log("Tagname de los hijos del body versión uno");
44          for(let hijo of hijosBody2)
45              console.log(hijo.tagName);
46
47          // hijosBody2.foreach(e=>console.log(hijo.tagName)); // No
                funciona. No es un array
48
49          console.log("Tagname de los hijos del body versión dos");
50          Array.from(hijosBody2).forEach(e=>console.log(e.tagName));
51
52          // restructurando
53          console.log("Tagname de los hijos del body versión tres");
54          const aHijos = [...hijosBody2];
55          aHijos.forEach(e=>console.log(e.tagName));
56
57          // Datos del primer hijo
58          console.log("Datos del primer hijo");
59          let primerHijo = hijosBody2[0]; // body.firstElementChild
60          console.log(primerHijo.tagName);
61
62          // Datos del segundo
63          console.log("Datos del segundo hijo");
64          let segundoHijo =primerHijo.nextElementSibling; // [1]
65          console.log(segundoHijo.tagName);
66          console.log(segundoHijo.textContent);
67
```

```
68          // Datos del último
69          console.log("Datos del último hijo");
70          console.log(hijosBody2[hijosBody2.length - 1].nodeType);
71          console.log(cuerpo.lastElementChild.nodeType);
72          console.log(cuerpo.lastElementChild.tagName);
73          console.log(cuerpo.lastElementChild.textContent); // Todo el
                código
74
75      </script>
76  </body>
77  </html>
```

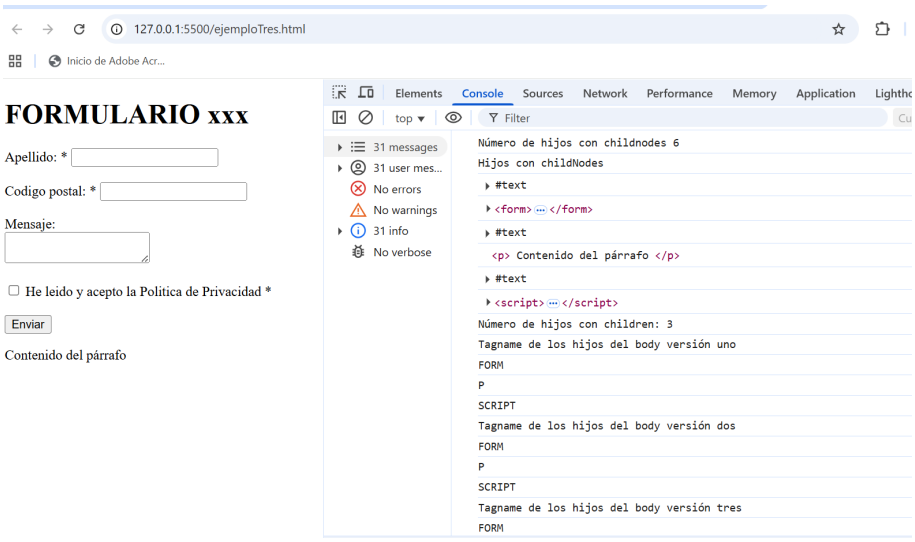


Figura 2.13: Ejecución del ejemplo tres

2.4. Modificación del DOM (creación, reemplazo o eliminación de nodos)

Crear y añadir a una página un nuevo elemento (etiqueta) consta de varios pasos:

- Creación de un nodo de tipo Element que represente al elemento.
- Creación de un nodo de tipo Text que represente el contenido del elemento o utilizar la propiedad textContent.
- Si hemos creado un elemento de tipo text, añadirlo como nodo hijo del nodo Element.
- Añadir el nodo Element a la página, en forma de nodo hijo del nodo correspondiente al lugar en el que se quiere insertar el elemento.

```
1
2  let p = document.createElement('p');
3  p.textContent=prompt("Contenido del párrafo textContent");
4  document.body.appendChild(p);
5
6
7  p = document.createElement('p');
8  let t = document.createTextNode(prompt("Contenido del párrafo
       createTextNode"));
9  p.append(t);
10 document.body.appendChild(p);
11
12
13 // Ejemplo con un documentFragment
14 <!DOCTYPE html>
15 <html lang="es">
16 <head>
17   <meta charset="UTF-8">
18   <title>Ejemplo DocumentFragment</title>
19 </head>
20 <body>
21   <ul id="lista"></ul>
22
23   <script>
24     // Lista de elementos a agregar
25     const elementos = ["Manzana", "Banana", "Cereza", "Uva"];
26
27     // Crear un DocumentFragment
28     const fragmento = document.createDocumentFragment();
29
30     // Crear los elementos <li> y agregarlos al fragmento
31     elementos.forEach(texto => {
32       const li = document.createElement("li");  // Crear elemento <li>
33       li.textContent = texto;                   // Agregar texto
34       fragmento.appendChild(li);                // Agregar al fragmento
```

```
35      });
36
37      // Agregar el fragmento completo al <ul>
38      document.getElementById("lista").appendChild(fragmento);
39    </script>
40  </body>
41  </html>
```

Hay más métodos de inserción que indican diferentes lugares donde insertar:

- node.**append**(...nodos o strings) agrega nodos o strings al final de node.
- node.**prepend**(...nodos o strings) agrega nodos o strings al principio de node.
- node.**before**(...nodos o strings) agrega nodos o strings antes de node.
- node.**after**(...nodos o strings) agrega nodos o strings después de node.
- node.**replaceWith**(...nodos o strings) reemplaza node con los nodos o strings dados.

Eliminar un nodo del árbol DOM de la página es mucho más sencillo que añadirlo. Solo es necesario utilizar la función **removeChild()**:

```
1
2  parrafo = document.getElementById("provisional");
3
4  parrafo.parentNode.removeChild(parrafo);
5
6  <p id="provisional">...</p>
```

La función removeChild() requiere como parámetro el nodo que se va a eliminar. Esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad parentNode.

Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga.

remove es la evolución de removeChild. Ya no hace falta hacer referencia al node padre.

```
1
2  <!DOCTYPE html>
3  <html>
4  <head>
```

```
5       <title>Ejemplo cinco DOM</title>
6   </head>
7   <body>
8       <button id = "bAdd" >Añadir</button>
9       <button id = "bDel">Borrar</button>
10      <p>Hola mundo inicial</p>
11      <script>
12          document.getElementById("bAdd").addEventListener("click", fAdd);
13          document.getElementById("bDel").addEventListener("click", fDel);
14          let parrafos = document.getElementsByTagName("p");
15
16          function fAdd()
17          {
18              let numero = parrafos.length;
19
20              let parrafo = document.createElement("p");
21              let contenido = document.createTextNode("Hola Mundo!" +
                    numero);
22              parrafo.appendChild(contenido);
23
24              document.body.appendChild(parrafo);
25
26              alert("El número de parrafos es: " + parrafos.length); //
                    Colección viva
27          }
28
29          function fDel()
30          {
31              // Supongo que el último
32              parrafos[parrafos.length - 1].parentNode.removeChild(
                    parrafos[parrafos.length - 1]);
33              alert("El número de parrafos es: " + parrafos.length);
34
35              let numero = parseInt(prompt("¿ Qué párrafo quieres
                    borrar?"));
36              parrafos[numero].remove();
37              alert("El número de parrafos es: " + parrafos.length);
38
39          }
40  </script>
41  </body>
42  </html>
```

2.4.1. innerHtml

La propiedad innerHTML permite obtener el HTML dentro del elemento como un string. También podemos modificarlo. Es una de las formas más fáciles de cambiar la página.

```
1
2  <html>
3      <head>
4          <title>Accediendo a la propiedad innerHTML</title>
5      </head>
6      <body>
7          <div id="parrafillo">
8              Este es un párrafo que contiene una lista
9              <ul>
10                 <li>Una</li>
11                 <li>lista</li>
12             </ul>
13         </div>
14         <script>
15             function cambia() {
16                 let el=document.getElementById( "parrafillo");
17
18                 el.innerHTML="Y lo cambio completamente"+
19                             "<ol><li>aunque</li><li>también
                                incluyo</li><li>una lista</li></ol>";
20             }
21
22             setTimeout( "cambia()", 2000 ); // Espera dos segundos  y
                   ejecuta la función
23         </script>
24     </body>
25 </html>
```

También podemos usar otro método muy versátil: **elemento.insertAdjacentHTML(posicion, texto)**.

El primer parámetro es un palabra código que especifica dónde insertar respecto a elemento. Debe ser uno de los siguientes:

beforebegin inserta html antes de elemento.

afterbegin justo dentro del elemento, antes de su primer elemento hijo..

beforeend justo dentro del elemento, después de su último elemento hijo.

afterend Después del propio elemento.

El segundo parámetro es un string HTML, que es insertado como HTML.

```
<!-- beforebegin -->
<p>
  <!-- afterbegin -->
  foo
  <!-- beforeend -->
</p>
<!-- afterend -->
```

Figura 2.14: Posiciones. Imagen extraída de https://developer.mozilla.org/es/docs/Web/API/Element/insertAdjacentHTML

```
1
2  <html>
3  <head>
4  <title>Accediendo a la propiedad innerHTML</title>
5  </head>
6  <body>
7  <div id="div">
8  </div>
9
10 <button id = "bAdd" onclick="cambia()" >Añadir</button>
11
12 <script>
13 function cambia() {
14   let el=document.getElementById( "div");
15
16   el.insertAdjacentHTML('beforebegin', '<p>Hola</p>');
17   el.insertAdjacentHTML('afterend', '<p>Adiós</p>');
18 }
19
20 </script>
21 </body>
22 </html>
```

También se pueden clonar nodos y grupos de nodos. Más información en https://es.javascript.info/modifying-document

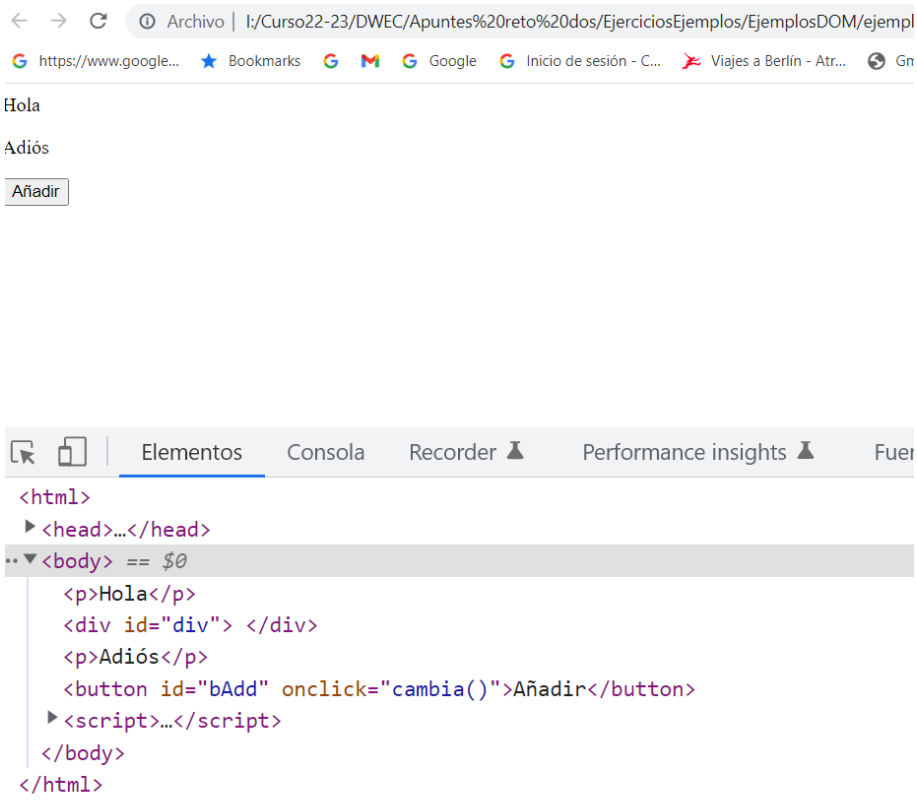


Figura 2.15: innerhtml antes y después de un div

2.5. Acceso directo a los atributos HTML

Los atributos de los elementos de la página se transforman automáticamente en propiedades de los nodos. Para acceder a su valor, simplemente se indica el nombre del atributo detrás del nombre del nodo.

El siguiente ejemplo obtiene de forma directa la dirección a la que enlaza el enlace:

```
1
2 let enlace = document.getElementById("enlace");
3 alert(enlace.href); // muestra http://www...com
4
5 <a id="enlace" href="http://www...com">Enlace</a>
```

Las propiedades CSS no son tan fáciles de obtener como los atributos HTML. Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style.

El siguiente ejemplo obtiene el valor de la propiedad margin de la imagen:

```
1
2 var imagen = document.getElementById("imagen");
3 alert(imagen.style.margin);
4
5 <img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Los nodos son objetos JavaScript. Podemos alterarlos añadiendo o eliminando propiedades y/o métodos. Si queremos acceder a un atributo no estándar hay que usar un método get.

Todos los atributos son accesibles usando los siguientes métodos:

elem.hasAttribute(nombre) comprueba si existe.

elem.getAttribute(nombre) obtiene el valor.

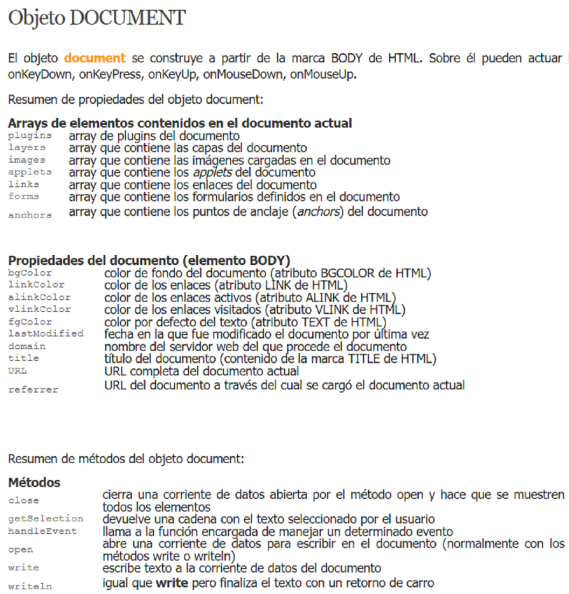
elem.setAttribute(nombre, valor) establece el valor.

elem.removeAttribute(nombre) elimina el atributo.

2.6. El objeto document

https://developer.mozilla.org/es/docs/Web/API/Document

El objeto document por si mismo, también tiene propiedades y métodos, muchas de las cuales ya hemos utilizado.



Objeto DOCUMENT

El objeto **document** se construye a partir de la marca BODY de HTML. Sobre él pueden actuar los eventos: onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseUp.

Resumen de propiedades del objeto document:

Arrays de elementos contenidos en el documento actual

plugins	array de plugins del documento
layers	array que contiene las capas del documento
images	array que contiene las imágenes cargadas en el documento
applets	array que contiene los *applets* del documento
links	array que contiene los enlaces del documento
forms	array que contiene los formularios definidos en el documento
anchors	array que contiene los puntos de anclaje (*anchors*) del documento

Propiedades del documento (elemento BODY)

bgColor	color de fondo del documento (atributo BGCOLOR de HTML)
linkColor	color de los enlaces (atributo LINK de HTML)
alinkColor	color de los enlaces activos (atributo ALINK de HTML)
vlinkColor	color de los enlaces visitados (atributo VLINK de HTML)
fgColor	color por defecto del texto (atributo TEXT de HTML)
lastModified	fecha en la que fue modificado el documento por última vez
domain	nombre del servidor web del que procede el documento
title	título del documento (contenido de la marca TITLE de HTML)
URL	URL completa del documento actual
referrer	URL del documento a través del cual se cargó el documento actual

Resumen de métodos del objeto document:

Métodos

close	cierra una corriente de datos abierta por el método open y hace que se muestren todos los elementos
getSelection	devuelve una cadena con el texto seleccionado por el usuario
handleEvent	llama a la función encargada de manejar un determinado evento
open	abre una corriente de datos para escribir en el documento (normalmente con los métodos write o writeln)
write	escribe texto a la corriente de datos del documento
writeln	igual que **write** pero finaliza el texto con un retorno de carro

Figura 2.16: Propiedades y métodos del objeto document

2.7. BOM (Modelo de objetos del navegador)

El Browser Object Model o BOM, nos permite acceder y modificar las propiedades de las ventanas del propio navegador.

Mediante BOM, es posible redimensionar y mover la ventana del navegador, modificar el texto que se muestra en la barra de estado y realizar muchas otras manipulaciones no relacionadas con el contenido de la página HTML.

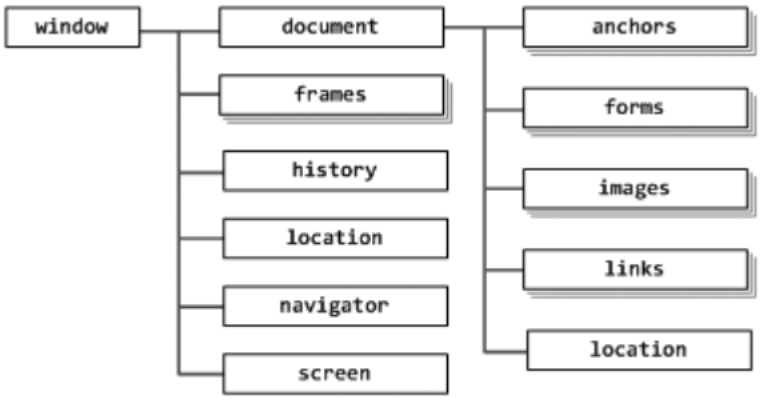


Figura 2.17: Modelo de objetos del navegador. Imagen extraída de https://es.javascript.info/browser-environment

2.7.1. El objeto Window

Representa una ventana abierta en un navegador.

https://www.w3schools.com/jsref/obj_window.asp

https://developer.mozilla.org/es/docs/Web/API/Window

2.7.2. El objeto Navigator

El objeto del navegador contiene información sobre el navegador.

https://www.w3schools.com/jsref/obj_navigator.asp

https://developer.mozilla.org/es/docs/Web/API/Navigator

2.7.3. El objeto Screen

El objeto de pantalla contiene información sobre la pantalla del visitante.

https://www.w3schools.com/jsref/obj_screen.asp

https://developer.mozilla.org/es/docs/Web/API/Screen

2.7.4. El objeto History

El objeto de historial contiene las URL visitadas por el usuario (en la ventana del navegador).

https://www.w3schools.com/jsref/obj_history.asp

https://developer.mozilla.org/es/docs/Web/API/Window/history

2.7.5. El objeto Location

El objeto de ubicación contiene información sobre la URL actual.

https://www.w3schools.com/jsref/obj_location.asp

https://developer.mozilla.org/es/docs/Web/API/Location

2.8. Fuentes

Además de las urls indicadas en los apuntes, se han utilizado como referencia las siguientes páginas

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=802:tipos-de-nodos-dom-document-element-text-attribute-comment-arbo catid=78&Itemid=206

https://es.javascript.info/document

http://www.w3schools.com/js/js_htmldom_document.asp

http://www.epsilon-eridani.com/cubic/ap/cubic.php/pag/JavaScript---Jerarquia-de-obj html