UNIDAD 5

Modelo del Objeto del Documento (DOM)

ÍNDICE:

1.- Introducción

La creación del *Document Object Model(***DOM)** permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. Gracias al DOM, podemos obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), añadir elementos a la página(párrafos, , etc.), aplicar animaciones (aparecer/desaparecer, etc.).

Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página HTML original, que no es más que una sucesión de caracteres en una estructura más eficiente de manipular.

2.- ¿ Cómo trabaja el DOM ?

El modelo de objeto de documento (DOM) es una interfaz de programación para los documentos HTML y XML. Facilita una representación estructurada del documento y define de qué manera los programas pueden acceder, al fin de modificar, tanto su estructura, estilo y contenido

El navegador transforma todos los documentos XHTML en un árbol de nodos interconectados que representan los contenidos de las páginas web y las relaciones entre ellos.

¿ Cómo trabaja el DOM ?

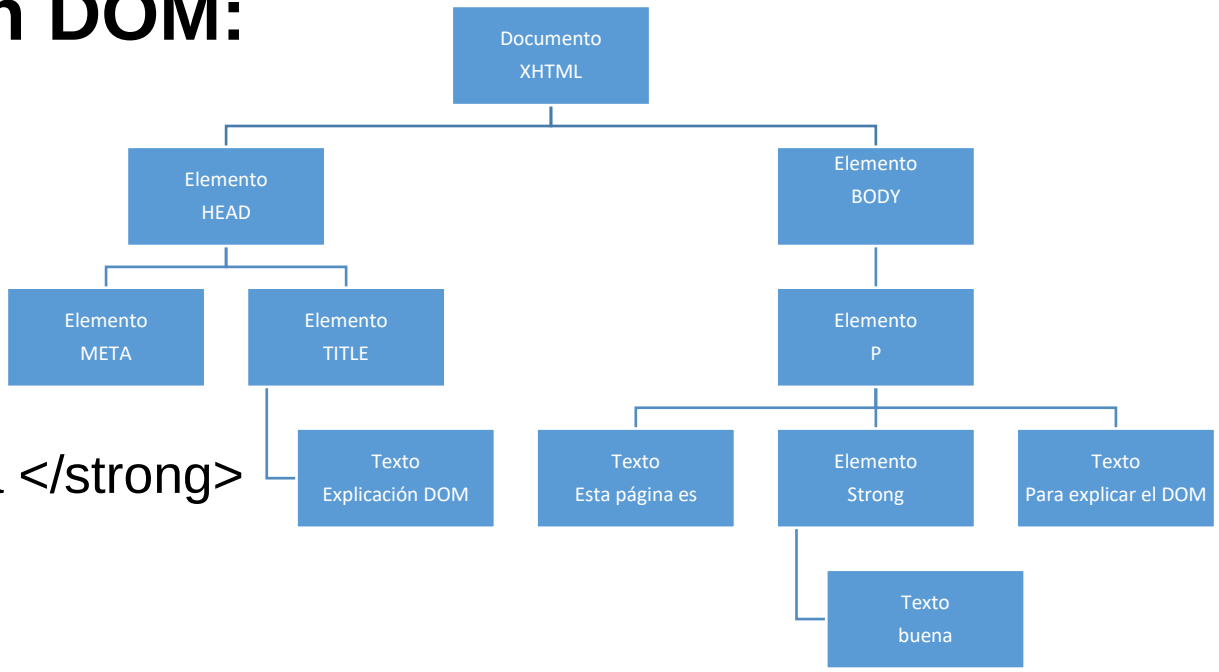
La transformación automática de la página en un árbol de nodos siempre sigue las mismas reglas:

- Las etiquetas XHTML se transforman en dos nodos: el primero es la propia etiqueta y el segundo nodo es hijo del primero y consiste en el contenido textual de la etiqueta.
- Si una etiqueta XHTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

¿ Cómo trabaja el DOM ?

Ejemplo transformación DOM:

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8">
<title>Explicación DOM</title>
</head>
<body>
<p>Esta página es <strong> buena </strong>
        para explicar el DOM</p>
</body>
</html>
```



3.- Tipos de Nodos del DOM

El DOM puede crear 12 tipos de nodos (Document, Element, Attr, Text, Comment, DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, Proces, singInstruction, Notation).

Realmente, vamos a destacar los siguientes:

•**Document**, nodo raíz del que derivan todos los demás nodos del árbol.

•**Element**, representa cada una de las etiquetas XHTML. Se trata del único nodo que puede contener atributos y el único del que pueden derivar otros nodos.

•**Attr**, se define para representar cada uno de los atributos de las etiquetas XHTML, es decir, uno por cada par atributo=valor.

•**Text**, nodo que contiene el texto encerrado por una etiqueta XHTML.

•**Comment**, representa los comentarios incluidos en la página XHTML.

4.- Acceso a los nodos del DOM

El DOM es una representación completamente orientada al objeto de la página web y puede ser modificado con un lenguaje de script como JavaScript.

Es importante recordar que el acceso a los nodos, su modificación y su eliminación solamente es posible cuando el árbol DOM ha sido construido completamente, es decir, después de que la página XHTML se cargue por completo.

4.1.- getElementsByTagName()

Obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función. El valor que se indica delante del nombre de la función es el nodo a partir del cual se realiza la búsqueda de los elementos. El valor que devuelve la función es un array de nodos DOM con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado.

Ejemplos:

```
//acceder al primer párrafo de la página
var parrafos = document.getElementsByTagName("p");
var párrafo=parrafos[0];
//acceder a todos todos los párrafos de la página
for(var i=0; i<parrafos.length; i++) { var parrafo = parrafos[i]; }
// acceder a todos los enlaces del primer párrafo de la página
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

4.2.- getElementsByName()

Obtener los elementos cuyo atributo name sea igual al parámetro proporcionado.

Ejemplo:
// obtener el único párrafo con el nombre especificado
var parrafo = document.getElementsByName("especial")
Esto conlleva que algún párrafo de la página sea de la forma
...
Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado, salvo en el caso de los elementos HTML *radiobutton*, donde el atributo name al ser el mismo para todos los elementos *radiobutton,* la función devuelve una colección de elementos.

4.3.- getElementById()

Devuelve el elemento XHTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.

Ejemplo:

var imagen = document.getElementById("imagen1");

Esto conlleva a que en algún momento habrá una imagen don ese id,
<img href="/" id="imagen1> </img>

Esta es la más utilizada cuando se desarrollan aplicaciones web dinámicas porque podemos acceder directamente a un nodo para poder leer o modificar sus propiedades.

4.4.- querySelector() y querySelectorAll()

querySelector devuelve solo **el primer** elemento que coincida con un selector válido css, mientras que si no coincide, devuelve null.

Sintaxis: document.querySelector(selectorcss)

querySelectorAll devuelve **todos** los elementos que coincidan con el selector css válido, en forma de array de nodos. Si no hay elementos que coincidan devuelve un array vacío.

Sintaxis: document.querySelectorAll(selectorcss)

5.- Manejo de atributos y data-atribute

Hay dos formas de acceder y manejar a los atributos de un documento o elementos html:

- Notación del punto (.)
- getAtribute() y setAtribute

```
<a href="https://www.marca.es" id="enlace" data-tema="deportes"
        style="background-color:yellow;color:green">Ir al Marca</a>
<script>
    const $imagen=document.getElementById("enlace");
            //Acceder a atributos del documento html
    //con notación del punto
    console.log(document.documentElement.lang);
    //con getAtribute
    console.log(document.documentElement.getAttribute("lang"));
    //Acceder a atributos de un elemento html
    console.log($imagen.id);
    console.log($imagen.getAttribute("id"));
    //asignar valor a un atributo del documento y de un elemento html
    document.documentElement.lang="es";
    console.log(document.documentElement.lang);
    document.documentElement.setAttribute("lang","it");
    console.log(document.documentElement.lang);
    //acceso a data-atributes
    //mediante notación punto
    console.log($imagen.dataset.tema);
    //mediante getAttribute (de la misma forma que atributos normales)
    console.log($imagen.getAttribute("data-tema"));
    </script>
```

Acceso a atributos

La transformación del nombre de las propiedades CSS compuestas consiste en eliminar todos los guiones medios (-) y escribir en mayúscula la letra siguiente a cada guión medio.

Ejemplo:

font-weight se transforma en fontWeight
line-height se transforma en lineHeight

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class por ser una palabra reservada. En su lugar, se usa el nombre className para acceder al atributo class de XHTML:

Ejemplo:

```
var parrafo = document.getElementById("parrafo");
alert(parrafo.className);
```

Conlleva a que haya algo así:

```
<p id="parrafo" class="normal">...</p>
```

Mediante DOM, es posible acceder de forma sencilla a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página, ya que los atributos XHTML 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 XHTML detrás del nombre del nodo.

Ejemplo: obtener la dirección de un enlace

var enlace = document.getElementById("enlace");

Esto conlleva a que haya un enlace Enlace

Ejemplo: obtener el valor de la propiedad margin de la imagen (se utiliza el atributo style).

var imagen = document.getElementById("imagen");

var estilomargen=alert(imagen.style.margin);

Conlleva que exista algo como esto:

Si el nombre de una propiedad CSS es compuesto, se accede a su valor modificando ligeramente su nombre:

var parrafo = document.getElementById("parrafo"); alert(parrafo.style.fontWeight); // muestra "bold" <p id="parrafo" style="font-weight: bold;">...</p

6.- Manejo de estilos

```
<a href="https://www.marca.es" id="enlace" data-tema="deportes"
        style="background-color:yellow;color:green">Ir al Marca</a>
<script>
    const $imagen=document.getElementById("enlace");
            //Acceder al conjunto de estilos de un elemento (observar la diferencia
en la notación)
    console.log($imagen.style);
    console.log($imagen.getAttribute("style"));
            //Acceder todos los estilos aplicables a un elemento
    console.log(getComputedStyle($imagen));
            //Acceder a un estilo concreto de un elemento con la notación punto
    console.log($imagen.style.backgroundColor);
            //Acceder a un estilo concreto con getPropertieValue
    console.log(getComputedStyle($imagen).getPropertyValue("color"));

            //Asignar un estilo con la notación punto
    $imagen.style.backgroundColor="red";
    console.log($imagen.style.backgroundColor);
            //Asignar un estilo con setPropertieValue
    $imagen.style.setProperty("color","blue");
    console.log($imagen.style.color);
    </script>
```

7.- Manejo de clases

```
<style>
    .clase1{ background-color: "grey"; color:"pink"; }
    .clase2{ background-color: "brown"; color:"blue"; }
</style>
<a href="https://www.marca.es" id="enlace" data-tema="deportes"
        style="background-color:yellow;color:green" class="clase1" >Ir al Marca</a>
<script>
    const $imagen=document.getElementById("enlace");
            //Acceder a la clase de un elemento
    console.log($imagen.className);
            //Acceder a la lista de clases de un elemento
    console.log($imagen.classList);
            //Añadir una clase a un elemento
    $imagen.classList.add("clase2");
    console.log($imagen.classList);
            //Eliminar una clase de un elemento
    $imagen.classList.remove("clase1");
    console.log($imagen.classList);
            //comprobar si un elemento tiene una clase
    console.log($imagen.classList.contains("clase1"),$imagen.classList.contains("clase2"));
            //asignar una clase si no la tiene o retirarla si la tiene
    $imagen.classList.toggle("clase1");
    $imagen.classList.toggle("clase2");
    console.log($imagen.classList.contains("clase1"),$imagen.classList.contains("clase2"));
            //reemplazar una clase por otra
    console.log($imagen.classList);
    $imagen.classList.replace("clase1","clase2");
    console.log($imagen.classList);
    </script>
```

8.- Manejo de elementos

```
<p id="parrafo1"> Párrafo primero para aprender manejo de elementos </p>
<p id="parrafo2"> Párrafo segundo para aprender manejo de elementos </p>
<script>
let $parrafo1=document.getElementById("parrafo1");
let texto=`<p id="parrafo3">Párrafo tercero para aprender manejo de elementos</p>
        <p id="parrafo4">Párrafo cuarto para aprender manejo de elementos</p>
        <ul>
         <li>1º DAW</li>
         <li>2º DAW</li>
        </ul>`;
//mostrar elemento html y su contenido textual
console.log($parrafo1,$parrafo1.innerText);
//cambiar contenido textual del elemento html
let $textoElemento=$parrafo1.innerText;
$parrafo1.innerText=texto;
console.log($parrafo1);
$parrafo1.innerText=$textoElemento;
console.log($parrafo1);
$parrafo1.textContent=texto;
console.log($parrafo1);
$parrafo1.innerText=$textoElemento;
console.log($parrafo1);
$parrafo1.innerHTML=texto;
console.log($parrafo1);
// sustituye el párrafo 1 por el 3 y el 4 (se puede comprobar en element del navegador)
$parrafo1.outerHTML=texto;
console.log($parrafo1);
    </script>
```

9.- Recorriendo el DOM (DOM Traversing)

```
<body>
<section id="seccion" class="claseSeccion">
    <div id="parrafos" class="clasediv">
        <p id="parrafo1"> Párrafo primero para aprender DOM Traversing </p>
        <p id="parrafo2"> Párrafo segundo para aprender DOM Traversing </p>
        <p id="parrafo3"> Párrafo terdero para aprender DOM Traversing</p>
    </div>
</section>
<script>
$divparrafos=document.getElementById("parrafos");
console.log($divparrafos);
//Acceder a los nodos hijos de un elemento
$hijosDiv=document.querySelector(".clasediv").children;
console.log( $hijosDiv);
console.log($hijosDiv[0]);
//Acceder al nodo padre de un elemento
console.log(document.getElementById("seccion"));
console.log(document.getElementById("parrafos").parentElement);
//Acceder al primer y al último hijo
console.log(document.getElementById("parrafos").firstElementChild);
console.log(document.getElementById("parrafos").lastElementChild);
// acceder al hermano anterior y posterior
console.log(document.getElementById("parrafo2").previousElementSibling);
console.log(document.getElementById("parrafo2").nextElementSibling);
// acceder al padre más cercano del selector dado
console.log(document.getElementById("parrafo3").closest("#seccion"));
console.log(document.getElementById("parrafo3").closest(".claseSeccion"));
</script>
```

10.- Manejo de nodos

Hay dos formas de insertar contenido dinámicamente en un documento HTML:

- Creando nuevos nodos
- Modificando directamente el contenido de un nodo

El proceso de creación de nuevos nodos implica la utilización de tres funciones DOM:

•createElement(etiqueta): crea un nodo de tipo Element que representa al elemento XHTML cuya etiqueta se pasa como parámetro.

•createTextNode(contenido): crea un nodo de tipo Text que almacena el contenido textual de los elementos XHTML.

•nodoPadre.appendChild(nodoHijo): añade un nodo como hijo de otro

El proceso de creación de nuevos nodos implica la utilización de tres funciones DOM:

•createElement(etiqueta): crea un nodo de tipo Element que representa al elemento XHTML cuya etiqueta se pasa como parámetro.
•createTextNode(contenido): crea un nodo de tipo Text que almacena el contenido textual de los elementos XHTML.
•nodoPadre.appendChild(nodoHijo): añade un nodo como hijo de otro nodo padre. Se debe utilizar al menos dos veces con los nodos habituales: en primer lugar se añade el nodo Text como hijo del nodo Element y a continuación se añade el nodo Element como hijo de algún nodo de la página.

La forma correcta de crear y añadir a la página un nuevo elemento (nodo) XHTML consta de cuatro pasos:

1.Creación de un nodo de tipo Element que represente al elemento.

2.Creación de un nodo de tipo Text que represente el contenido del elemento.

3.Añadir el nodo Text como nodo hijo del nodo Element (o bien incluir el contenido del párrafo con la propiedad innerHTML)

4.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.

Ejemplos

```
<section id="seccion" class="claseSeccion">
        <div id="parrafos" class="clasediv">
            <p id="parrafo1"> Párrafo primero para aprender DOM Traversing</p>
            <p id="parrafo2"> Párrafo segundo para aprender DOM Traversing</p>
            <p id="parrafo3"> Párrafo terdero para aprender DOM Traversing</p>
        </div>
    </section>

    <script>
       //insertar nodos de dos formas distintas: primera en forma de nodo real
var nodoParrafo = document.createElement("p");
var textoParrafo = document.createTextNode("Este es otro párrafo insertado");
nodoParrafo.appendChild(textoParrafo);
document.getElementById("parrafos").appendChild(nodoParrafo);
console.log(document.getElementById("parrafos"));
       // insertar contenido modificando contenido de un nodo
var nodoParrafo1 = document.createElement("p");
nodoParrafo1.innerHTML = "Este es otro párrafo ´más insertado";
document.getElementById("parrafos").appendChild(nodoParrafo);
console.log(document.getElementById("parrafos"));
</script>
```

Insertar un nodo con insertAdjacentElement

Sintaxis: elementoObjetivo.insertAdjacentElement(posición, elemento);

- posicion

- 'beforebegin': Antes del elementoObjetivo.
- 'afterbegin': Dentro del elementoObjetivo, antes de su primer hijo.
- 'beforeend': Dentro del elementoObjetivo, después de su último hijo.
- 'afterend': Después del elementoObjetivo.

- **Elemento**
  - El elemento a insertar

```
<section id="seccion" class="claseSeccion">
        <div id="parrafos" class="clasediv">
            <p id="parrafo1"> Párrafo primero para aprender DOM Traversing </p>
            <p id="parrafo2"> Párrafo segundo para aprender DOM Traversing </p>
            <p id="parrafo3"> Párrafo terdero para aprender DOM Traversing</p>
         </div>
    </section>
        <script>
            var $parrafo2=document.getElementById("parrafo2");
            var $seccion=document.getElementById("seccion");
            var $parrafoNuevoAntes = document.createElement("p");
            $parrafoNuevoAntes.innerHTML="Este es el nuevo párrafo insertado antes de párrafo2";
            var $parrafoNuevoDespues = document.createElement("p");
            $parrafoNuevoDespues.innerHTML="Este es el nuevo párrafo insertado después de párrafo2";
            var $parrafoNuevoAntesPrimerHijo = document.createElement("p");
            $parrafoNuevoAntesPrimerHijo.innerHTML="Nuevo párrafo insertado antes del primer hijo del párrafo2";
            var $parrafoNuevoDespuesUltimoHijo = document.createElement("p");
            $parrafoNuevoDespuesUltimoHijo.innerHTML="Nuevo párrafo insertado antes del primer hijo del párrafo2";
            $parrafo2.insertAdjacentElement("beforebegin", $parrafoNuevoAntes);
            $parrafo2.insertAdjacentElement("afterend", $parrafoNuevoDespues);
            $seccion.insertAdjacentElement("afterbegin",$parrafoNuevoAntesPrimerHijo);
            $seccion.insertAdjacentElement("beforeend",$parrafoNuevoDespuesUltimoHijo);
            console.log(document.getElementById("parrafos"));
        </script>
```

Insertar código HTML o texto plano

Sintaxis: elementoObjetivo.insertAdjacentHTML(posición, elemento);

- posicion

- 'beforebegin': Antes del elementoObjetivo.
- 'afterbegin': Dentro del elementoObjetivo, antes de su primer hijo.
- 'beforeend': Dentro del elementoObjetivo, después de su último hijo.
- 'afterend': Después del elementoObjetivo.

- **Elemento**
  - El código HTML a insertar

```
<section id="seccion" class="claseSeccion">
       <div id="parrafos" class="clasediv">
           <p id="parrafo1"> Párrafo primero para aprender DOM Traversing </p>
           <p id="parrafo2"> Párrafo segundo para aprender DOM Traversing </p>
           <p id="parrafo3"> Párrafo terdero para aprender DOM Traversing</p>
        </div>
   </section>
       <script>
         var $parrafo2=document.getElementById("parrafo2");
         $parrafo2.insertAdjacentHTML("afterend","<p>Párrafo Nuevo después del párrafo2</p>")
       </script>
```

NOTA: Igualmente existe insertAdjacentText para insertar texto plano

Eliminar un nodo

Solo es necesario utilizar la función removeChild()

Ejemplo:

```
var parrafoAEliminar = document.getElementById("parrafo1");
parrafoAEliminar.parentNode.removeChild(parrafoAEliminar);
console.log(document.getElementById("seccion"));
```

Esto conlleva a que haya un párrafo ...
La función removeChild() requiere como parámetro el nodo que se va a eliminar. Además, 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 nodoHijo.parentNode.
Así, para eliminar un nodo de una página XHTML se invoca a la función removeChild() desde el valor parentNode del nodo que se quiere eliminar. Cuando se elimina un nodo, también se eliminan automáticamente todos los nodos hijos que tenga, por lo que no es necesario borrar manualmente cada nodo hijo.

Ejemplo:

```
var parrafoConHijosAEliminar = document.getElementById("parrafos");
parrafoConHijosAEliminar.parentNode.removeChild(parrafoConHijosAEliminar);
console.log(document.getElementById("seccion"));
```

11.- Uso de fragmento

Un fragmento se usa para no tener que estar insertando continuamente en el DOM, sino que lo que haremos será insertar en un espacio virtual llamado fragmento, y, después será el fragmento el que insertaremos ya en el DOM, con lo que ganaremos rapidez y eficacia.

```
    <script>
 //insertar en fragmento
const alumnos=["Ana","Antonio","Belén","Conchi","Daniel","Eugenio","Felipe",
               "Gabriela","Higinio","Isabel","Juan","Laura","Miguel","Pepe","Raul","Susana"];
        var lista=document.createElement("ul");
        var fragmento=document.createDocumentFragment();
        //añadiremos nodos al fragmento
        alumnos.forEach(mes =>{
            const li=document.createElement("li");
            li.innerHTML=mes;
            fragmento.appendChild(li);
        })
        //agregamos el fragmento a la lista y ya la lista al body del documento
        document.write("<h1>Alumnos del ciclo</h1>");
        lista.appendChild(fragmento);
        document.body.appendChild(lista);
        console.log(document.querySelector("ul"));
    </script>
```

12.- Selectores

Selectores CSS:

- Universal
- De tipo
- Clases
- Identificador
- Por atributo
- Descendiente
- Pseudo-clases

```
/* Selector Universal */
*    {
     background-color:grey;
     }
/* Selector de tipo */
p
     {
     background-color:red;
     }
/* Selector de clase */
.clases
     {
     background-color:green;
     }
/* Selector por identificador */
#cuadrotexto
     {
     background-color: blueviolet;
     }
/* Selector por atributo */
[atributo="estiloatributo"]
     {
     background-color: chartreuse;
     }
```

```
/* Selector por dependiente */
h4 b
     {
     background-color: darkgoldenrod;
     }
h4 p b
     {
     background-color: rgb(72, 174, 187);
     }
.clases strong
     {
     background-color: ghostwhite;
     }
/* Selector por pseudo-clases */
a:hover
     {
     background-color: gold;
     }
```

Ejemplo con los selectores anteriores

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="./css/estilos.css">
</head>
<body>
<h1 class="clases">Ejemplo para practicar estilos</h1>
<h2 class="clases">Distintos selectores se aplican al <strong>mismo o distintos
elementos</strong></h2>
<input type="text" id="cuadrotexto" value="cuadro de texto">
<button atributo="estiloatributo">botón</button>
<div>
<p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Vero quod amet ea rerum, </p>
<p>Debitis voluptatem non, saepe sequi incidunt q</p>
<p>Eaque odio assumenda veritatis error illum vero sentium quas porro esse!</p>
<p>Fuga, voluptatum ratione laboriosam non assumendandae qui.</p>
<p>Assumenda recusandae laboriosam, excepturi amet, pariatur, corporiscupiditate!</p>
</div>
<h4>Esto está en <b>H4</b></h4>
<h4><p>Esto está en <b>H5</b> dentro de un párrafo</p></h4>
<h5>Esto es para explicar las pseudoclases <a href="http://www.marca.es">Enlace al Marca</a></h5>
</body>
</html>
```

12.1.- Especificidad

Nosotros vamos a aplicar 3 formas de aplicar estilos:

- Estilos en línea
- Con la etiqueta <style></style>
- Con archivo css

Por tanto, a la hora de aplicar estilos hay que tener en cuenta cierta jerarquía en caso de conflicto entre varios estilos con distintos selectores al mismo componente del documento

La jerarquía expuesta es la que hay que tener en cuenta a la hora de aplicar estilos css en documentos html y js.