# DOM
# (Modelo de Objetos del Documento)

# Contenido

# 1. DOM

La creación del *Document Object Model* o **DOM** es una de las innovaciones que más ha influido en el desarrollo de las páginas web dinámicas y de las aplicaciones web más complejas.

El Modelo de Objetos del Documento (DOM) es un API para documentos HTML y XML. Proporciona una representación estructural del documento, permitiendo la modificación de su contenido y visualización. Esencialmente, comunica las páginas web con los scripts o los lenguajes de programación.

DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML. De hecho, DOM se diseñó originalmente para manipular de forma sencilla los documentos XML.

A pesar de sus orígenes, DOM se ha convertido en una utilidad disponible para la mayoría de lenguajes de programación (Java, PHP, JavaScript) y cuyas únicas diferencias se encuentran en la forma de implementarlo.

## 1.1. Árbol de nodos

Una de las tareas habituales en la programación de aplicaciones web con JavaScript consiste en la manipulación de las páginas web. De esta forma, es habitual obtener el valor almacenado por algunos elementos (por ejemplo los elementos de un formulario), crear un elemento (párrafos, `<div>`, etc.) de forma dinámica y añadirlo a la página, aplicar una animación a un elemento (que aparezca/desaparezca, que se desplace, etc.).

Todas estas tareas habituales son muy sencillas de realizar gracias a DOM. Sin embargo, para poder utilizar las utilidades de DOM, es necesario *"transformar"* la página original. Una página HTML normal no es más que una sucesión de caracteres, por lo que es un formato muy difícil de manipular. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

DOM transforma todos los documentos XHTML en un conjunto de elementos llamados *nodos*, que están interconectados y que representan los contenidos de las páginas web y las relaciones entre ellos. Por su aspecto, la unión de todos los nodos se llama *"árbol de nodos"*.

La siguiente página XHTML sencilla:

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>

</html>
```

Se transforma en el siguiente árbol de nodos:

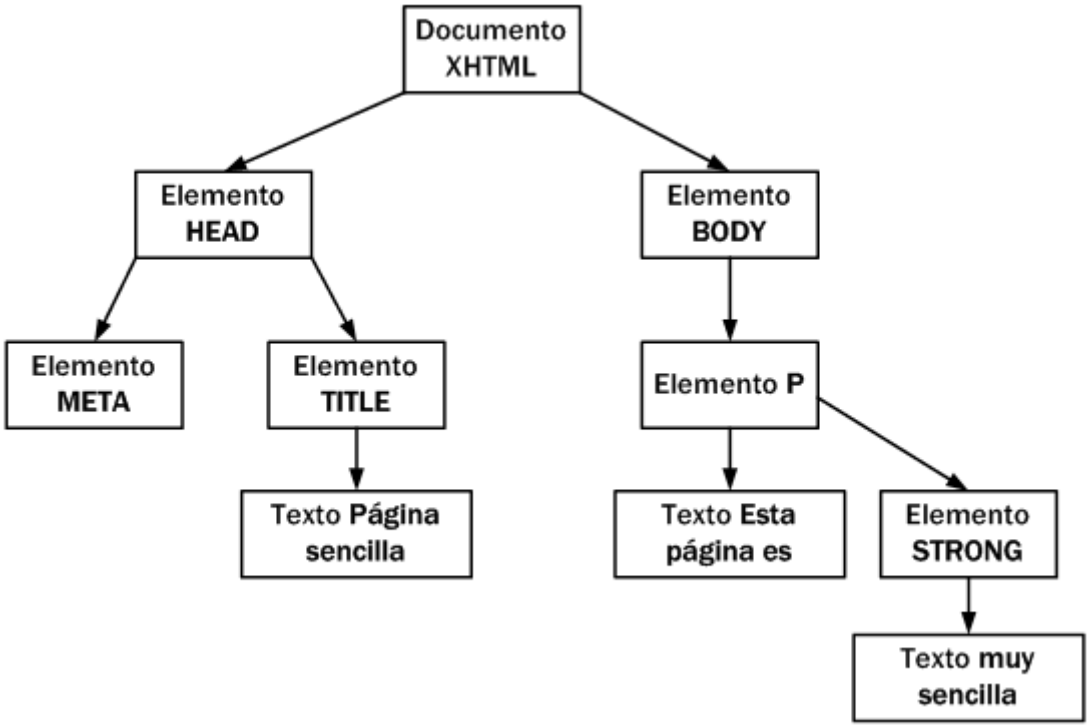


**Figura 1.1** Árbol de nodos generado automáticamente por DOM a partir del código XHTML de la página

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo (que se verá más adelante) y su contenido.

La raíz del árbol de nodos de cualquier página XHTML siempre es la misma: un nodo de tipo especial denominado "Documento".

A partir de ese nodo raíz, cada etiqueta XHTML se transforma en un nodo de tipo "Elemento". La conversión de etiquetas en nodos se realiza de forma jerárquica. De esta forma, del nodo raíz solamente pueden derivar los nodos HEAD y BODY. A partir de esta derivación inicial, cada etiqueta XHTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre".
La transformación de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta XHTML.

Así, la siguiente etiqueta XHTML:

```
<title>Página sencilla</title>
```

Genera los siguientes dos nodos:

**Figura 1.2** Nodos generados automáticamente por DOM para una etiqueta XHTML sencilla De la misma forma, la siguiente etiqueta XHTML:

```
<p>Esta página es <strong>muy sencilla</strong></p>
```

Genera los siguientes nodos:

- Nodo de tipo *"Elemento"* correspondiente a la etiqueta `<p>`.
- Nodo de tipo *"Texto*" con el contenido textual de la etiqueta `<p>`.
- Como el contenido de `<p>` incluye en su interior otra etiqueta XHTML, la etiqueta interior se transforma en un nodo de tipo *"Elemento"* que representa la etiqueta `<strong>` y que deriva del nodo anterior.
- El contenido de la etiqueta `<strong>` genera a su vez otro nodo de tipo *"Texto"* que deriva del nodo generado por `<strong>`.

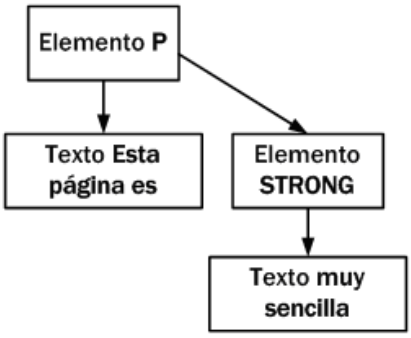


**Figura 1.3** Nodos generados automáticamente por DOM para una etiqueta XHTML con otras etiquetas XHTML en su interior

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.

Como se puede suponer, las páginas XHTML habituales producen árboles con miles de nodos. Aun así, el proceso de transformación es rápido y automático, siendo las funciones proporcionadas por DOM (que se verán más adelante) las únicas que permiten acceder a cualquier nodo de la página de forma sencilla e inmediata.

## 1.2. Tipos de nodos

La especificación completa de DOM define 12 tipos de nodos, aunque las páginas XHTML habituales se pueden manipular manejando solamente cuatro o cinco tipos de nodos:

- `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 un nodo de este tipo 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.

**Ejercicio**: Crear el árbol de nodos de la siguiente página web

```
<HTML>
<HEAD>
    <TITLE> Título de la página</TITLE>
</HEAD>
<BODY>
    <H1> Encabezado </H1>
    <P> Un párrafo </P>
    <P> Un párrafo con <A HREF="pagina2.html"> un vínculo </A> </P>
    <UL>
        <LI> Primer elemento de la lista </LI>
        <LI> Segundo elemento de la lista </LI>
    </UL>
    <P> Otro párrafo </P>
</BODY>
</HTML>
```

## 1.3. Explorar el árbol del DOM

Partiendo de un nodo del DOM (bien sea un nodo de elemento o bien el propio nodo raíz document) podemos acceder a cualquier otro nodo utilizando las propiedades y métodos que se describen en la siguiente tabla para que sirva como referencia. Tenga en cuenta que estas propiedades/métodos los heredan los elementos de la interfaz Node y, consecuentemente, no sólo tienen en cuenta los nodos de tipo elemento, sino también todos los demás (como los de tipo texto).

| **Propiedad/método** | **Descripción** |
|---|---|
| childNodes | Es una propiedad de sólo lectura que contiene un nodeList (recuerde que un nodeList es una colección (lista ordenada) de objetos de tipo nodo) de todos los nodos hijo (no nietos ni demás descendientes) del nodo actual. Tiene en cuenta los nodos de elemento y los de texto, pero no los de atributo. |
| firstChild | Es una propiedad de sólo lectura que contiene el primer nodo hijo del elemento actual. Puede ser otro nodo de tipo elemento o bien un nodo de tipo texto (recuerde que los espacios se consideran nodos de tipo texto). Devuelve `null` si no tiene hijos. |
| lastChild | Es una propiedad de sólo lectura que contiene el último nodo hijo del elemento actual. Puede ser otro nodo de tipo elemento o bien un nodo de tipo texto (recuerde que los espacios se consideran nodos de tipo texto). Devuelve `null` si no tiene hijos. |
| nextSibling | Es una propiedad de sólo lectura que contiene el siguiente nodo hermano del actual. Devuelve `null` si no tiene hermanos menores. |
| previousSibling | Es una propiedad de sólo lectura que contiene el anterior nodo hermano del actual. Devuelve `null` si no tiene hermanos mayores. |
| parentNode | Es una propiedad de sólo lectura que contiene el nodo padre del actual. Devuelve `null` si no tiene padre. |
| hasChildNodes() | Es un método que nos indica si el elemento al que se aplica posee nodos hijo. |

| | |
|---|---|
| nodeName | Devuelve el nombre del nodo (tag Name, nombre del atributo,…) |
| NodeType | Devuelve el tipo de nodo (Element, Attr, text, Comment,…) |
| NodeValue | Devuelve el valor del atributo(null para Element, texto para Text, valor atributo para Attr) |

Ejemplos: ejemplo_explorar_Dom.html, ejemplo_explorar_Dom2.html

## 1.4. Acceso directo a los nodos

Una vez construido automáticamente el árbol completo de nodos DOM, ya es posible utilizar las funciones DOM para acceder de forma directa a cualquier nodo del árbol. Como acceder a un nodo del árbol es equivalente a acceder a "un trozo" de la página, una vez construido el árbol, ya es posible manipular de forma sencilla la página: acceder al valor de un elemento, establecer el valor de un elemento, mover un elemento de la página, crear y añadir nuevos elementos, etc.

DOM proporciona dos métodos alternativos para acceder a un nodo específico: acceso a través de sus nodos padre y acceso directo.

Las funciones que proporciona DOM para acceder a un nodo a través de sus nodos padre consisten en acceder al nodo raíz de la página y después a sus nodos hijos y a los nodos hijos de esos hijos y así sucesivamente hasta el último nodo de la rama terminada por el nodo buscado.
Sin embargo, cuando se quiere acceder a un nodo específico, es mucho más rápido acceder directamente a ese nodo y no llegar hasta él descendiendo a través de todos sus nodos padre.

Por último, 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.

### 1.4.1. getElementsByTagName()

La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.

El siguiente ejemplo muestra cómo obtener todos los párrafos de una página XHTML:

```
var parrafos = document.getElementsByTagName("p");
```

El valor que se indica delante del nombre de la función (en este caso, `document`) es el nodo a partir del cual se realiza la búsqueda de los elementos. En este caso, como se quieren obtener todos los párrafos de la página, se utiliza el valor `document` como punto de partida de la búsqueda.

El valor que devuelve la función es un array con todos los nodos que cumplen la condición de que su etiqueta coincide con el parámetro proporcionado. El valor devuelto es un array de nodos DOM, no un array de cadenas de texto o un array de objetos normales. Por lo tanto, se debe procesar cada valor del array de la forma que se muestra en las siguientes secciones.

De este modo, se puede obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

De la misma forma, se podrían recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++)

  { var parrafo = parrafos[i];

}
```

La función `getElementsByTagName()` se puede aplicar de forma recursiva sobre cada uno de los nodos devueltos por la función. En el siguiente ejemplo, se obtienen todos los enlaces del primer párrafo de la página:

```
var parrafos =

document.getElementsByTagName("p");

 var primerParrafo = parrafos[0];

var enlaces = primerParrafo.getElementsByTagName("a");
```

### 1.4.2. getElementsByName()

La función `getElementsByName()` es similar a la anterior, pero en este caso se buscan los elementos cuyo atributo `name` sea igual al parámetro proporcionado. En el siguiente ejemplo, se obtiene directamente el único párrafo con el nombre indicado:

```
var parrafoEspecial = document.getElementsByName("especial");


<p name="prueba">...</p>

<p name="especial">...</p>

<p>...</p>
```

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. En el caso de los elementos HTML *radiobutton*, el atributo `name` es común a todos los *radiobutton* que están relacionados, por lo que la función devuelve una colección de elementos.

### 1.4.3. getElementById()

*La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas. Se trata de la función preferida para acceder directamente a un nodo y poder leer o modificar sus propiedades.*

*La función 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.*

```
   var cabecera = document.getElementById("cabecera");

   <div id="cabecera">
     <a href="/" id="logo">...</a>
   </div>
```

## 1.5. Creación y eliminación de nodos

*Acceder a los nodos y a sus propiedades es sólo una parte de las manipulaciones habituales en las páginas. Las otras operaciones habituales son las de crear y eliminar nodos del árbol DOM, es decir, crear y eliminar "trozos" de la página web.*

En la *siguiente* tabla se describen los métodos que nos permiten manipular el DOM mediante objetos de tipo elemento; los dos primeros pertenecen al objeto document, y el resto puede aplicarse sobre objetos de tipo elemento.

| **Método** | **Descripción** |
|---|---|
| document.createElement(*etiqueta*) | Devuelve un objeto de tipo nodo de elemento y, más concretamente, del tipo de elemento correspondiente a la *etiqueta* HTML. Este método crea el nodo, pero no lo inserta en ninguna posición del DOM, por lo que no será visible hasta que lo ubiquemos. |
| document.createTextNode(*texto*) | Devuelve un objeto de tipo nodo de texto con el *texto* indicado. Este método crea el nodo, pero no lo inserta en ninguna posición del DOM, por lo que no será visible hasta que lo ubiquemos. |
| appendChild(*nodo*) | Inserta el *nodo* en el DOM como hijo menor del elemento actual. |
| insertBefore(*nodo1*,*nodo2*) | Inserta el *nodo1* como hermano inmediatamente mayor de *nodo2* dentro del elemento actual. |
| removeChild(*nodo*) | Elimina el nodo del DOM y lo devuelve como una referencia que podemos recoger en una variable para rescatarlo posteriormente. |
| remove() | Elimina el nodo<br>Ej: element.remove() |
| replaceChild(*nuevo*,*viejo*) | Sustituye el nodo *viejo* por el *nuevo* dentro del elemento actual. |
| cloneNode(*incluirDescencientes*) | Devuelve un elemento que es la copia exacta del actual, salvo porque no incluye sus detectores de eventos.<br>*incluirDescendientes* es un valor booleano; si es `true` duplica el elemento y todos sus nodos descendientes; si es `false` duplica el nodo de elemento sin ninguno de sus descendientes (ni siquiera los nodos de tipo texto). Tenga en cuenta que este argumento es obligatorio en algunos navegadores, mientras que en otros se considera por defecto false. |

### a) Ejemplo añadir nodos:

Como se ha visto, un elemento XHTML sencillo, como por ejemplo un párrafo, genera dos nodos: el primer nodo es de tipo `Element` y representa la etiqueta `<p>` y el segundo nodo es de tipo `Text` y representa el contenido textual de la etiqueta `<p>`.

Por este motivo, crear y añadir a la página un nuevo elemento XHTML sencillo consta de cuatro pasos diferentes:

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`.
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.

De este modo, si se quiere añadir un párrafo simple al final de una página XHTML, es necesario incluir el siguiente código JavaScript:

```
// Crear nodo de tipo Element

var parrafo = document.createElement("p");

// Crear nodo de tipo Text

var contenido = document.createTextNode("Hola Mundo!");
// Añadir el nodo Text como hijo del nodo

Element parrafo.appendChild(contenido);


// Añadir el nodo Element como hijo de la página

document.body.appendChild(parrafo);
```

### b) Ejemplo eliminar nodo:

```
var parrafo = document.getElementById("provisional");
parrafo.parentNode.removeChild(parrafo);
ó
parrafo.remove()

<p id="provisional">...</p>
```

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`.

### c) Ejemplo modificar nodo:

```
var nuevoDiv = document.createElement("div");
var nuevoTexto = document.createTextNode("Cadena que sustituye a Hola mundo");
nuevoDiv.appendChild(nuevoTexto);
var originalDiv = document.getElementByTagName("div")[0];
originalDiv.parentNode.replaceChild(nuevoDiv, originalDiv);
```

La estructura de la página quedará:

```
<html>
<head><title>Ejemplo de modificación de un nodo </head>
<body>
        <div> Cadena que sustituye a Hola Mundo</div>
</body>
</html>
```

### d) Ejemplo insertar nodo antes:

```
Var divAntes = document.createElement(“div”);
Var textoAnt = document.createTextNode(“div antes que el original”);
divAntes.appendChild(textoAnt)
var divOriginal = document.getElementsByTagName(“div”)[0];
document.body.insertBefore(divAntes, divOriginal);
```

La estructura de la página quedaría:

```
<html>
<head><title>Ejemplo de insertar antes de un nodo</title> </head>
<body>
        <div> div antes que el original</div>
        <div> Cadena que sustituye a Hola Mundo </div>
</body>
</html>
```

## 1.6. Gestionar los atributos de un elemento

Podemos gestionar cualquier atributo a través de los métodos **getAttribute(), setAttribute(), removeAttribute() y hasAttribute()**.

El objeto Element posee propiedades asociadas a los principales atributos que puede tener un elemento genérico HTML (id, className, name, style, y title), y los objetos de tipo HTMLxxxElement posee propiedades que puede tener cada tipo de elemento en particular.

En la siguiente tabla se describen las propiedades genéricas.

| **Propiedad** | **Descripción** |
|---|---|
| id | Lee o establece el atributo `id` del elemento. Recuerde que el id de cada elemento debe ser exclusivo (no puede haber dos elementos con el mismo id). |
| className | Lee o establece el atributo `class` del elemento. Pueden asignarse varias clases a un elemento separando sus nombres con espacios. |
| name | Lee o establece el atributo `name` del elemento. |
| title | Lee o establece el atributo `title` del elemento. El valor de este atributo se muestra como un mensaje de herramienta (*tooltip*) al colocar el puntero del ratón sobre el elemento. |
| style | Es una referencia a un objeto cuyas propiedades coinciden con las propiedades CSS configuradas para el elemento en su atributo `style` . |

Por ejemplo, en el siguiente código tenemos un botón que cambia la clase del párrafo para aplicarle un estilo diferente y otro que actúa directamente sobre su atributo style.

```
<!DOCTYPE html>
<html>
	<head>
		<title>DOM</title>
		<style>
			.claseParrafo {color:red;}
		</style>
	</head>
	<body>
		<p id='parrafo'>El estilo de este párrafo está controlado por los botones.</p>

		<button type='button'
onclick='document.getElementById("parrafo").className="claseParrafo";'>Cambiar clase</button>
		<button type='button'
onclick='document.getElementById("parrafo").style.color="green";'>Cambiar atributo style</button>
	</body>
</html>
```

También podemos usar los métodos DOM getAttribute(nombre), setAttribute(nombre,valor), removeAttribute(nombre) y hasAttribute(nombre) del objeto Element para, respectivamente, leer el valor, crear/modificar, eliminar y averiguar si existe un atributo. Si intentamos obtener con getAttribute(nombre) un atributo que no existe obtendremos el valor null; es conveniente comprobar previamente la existencia del atributo con hasAttribute(nombre). Por ejemplo:

```
<!DOCTYPE html>
<html>
	<head>
		<title>DOM</title>
		<style>
			.claseParrafo {color:red;}
		</style>
		<script>
			function crearClase(){
				document.getElementById('parrafo').setAttribute('class','claseParrafo');
			}
			function leerClase(){
				if(document.getElementById('parrafo').hasAttribute('class')){
					alert(document.getElementById('parrafo').getAttribute('class'));
				}else{
					alert('No tiene clase');
				}
			}
		</script>
	</head>
	<body>
		<p id='parrafo'>El estilo de este párrafo está controlado por los botones.</p>

		<button type='button' onclick='crearClase();'>Crear clase</button>
		<button type='button' onclick='leerClase();'>Leer clase</button>
	</body>
</html>
```

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. A continuación se muestran algunos ejemplos:

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

- `border-top-style` se transforma en `borderTopStyle`
- `list-style-image` se transforma en `listStyleImage`

El único atributo XHTML que no tiene el mismo nombre en XHTML y en las propiedades DOM es el atributo class. Como la palabra class está reservada por JavaScript, no es posible utilizarla para acceder al atributo class del elemento XHTML. En su lugar, DOM utiliza el nombre className para acceder al atributo class de XHTML

En lugar de buscar por atributos del elemento, también podemos usar como criterio selectores CSS (recuerde que los selectores CSS pueden ser de etiqueta, de id y de clase, y que se pueden agrupar con una coma, o combinar mediante los operadores de descendiente (espacio), hijo (>) y hermano menor (+)) gracias a los métodos **querySelector(cadenaSelector)** y **querySelectorAll(cadenaSelector);** el primero devuelve el primer elemento que coincide con el selector, mientras que el segundo devuelve todos los elementos que coinciden con el selector.

Ejemplo:

```
document.querySelector("p").style.backgroundColor = "red";
//cambia el color de fondo del primer párrafo

document.getElementsByTagName('table')[1].querySelectorAll ('tr.encabezado');
//seleccionar dentro de la segunda tabla, todas las filas que pertenezcan a la clase encabezado

document.querySelector("#demo").innerHTML = "Hello World!";
//cambia el texto del elemento con id=”demo”

var imagenes = document.querySelectorAll(".imagen");
//todos los elementos que pertenezcan a la clase imagen

var button = document.querySelector("input[type=button]");
//todos los elementos input cuya propiedad type es button
```

HTML DOM querySelector() Method (w3schools.com)

http://www.w3schools.com/cssref/css_selectors.asp

## Ejercicio

Dado el siguiente código html

```
<body><input type="button" id="boton1" value="Crea" ></body>
```

Programar la creación dinámica de una página web al pulsar dicho botón con los siguientes elementos: