La alegría de crecer juntos

Plaza de Castilla · Madrid
COLEGIO NUESTRA SEÑORA DEL PILAR
Salesianas

C.F.G.S.: DESARROLLO DE APLICACIONES WEB
Módulo: DESARROLLO WEB EN ENTORNO CLIENTE

# 10 DOCUMENT OBJECT MODEL: DOM

## 1. Introducción al DOM

Hasta ahora hemos accedido a algunos elementos de la página web mediante un objeto que se llamaba document. El objeto document es el objeto de más alto nivel dentro del Document Object Model (Modelo de Objetos del Documento) o DOM.

El DOM es un estándar de W3C que define como acceder a los documentos HTML y XML, describiendo el contenido del documento como un conjunto de objetos sobre los que un programa Javascript puede actuar.

El DOM es una interfaz de programación de aplicaciones (API) de la plataforma de W3C y cuenta con un lenguaje que permite a los programas y scripts acceder y actualizar dinámicamente el contenido, estructura y estilo de un documento permitiendo obtener, modificar, añadir o eliminar elementos HTML modificando estructura, estilo y contenido.

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.

- ✓ *DOM Document Object Model define el contenido de un documento HTML como un conjunto de objetos sobre los que JavaScript puede actuar.*
- ✓ *Podremos acceder y actualizar dinámicamente el contenido, estructura y estilo de un documento HTML*

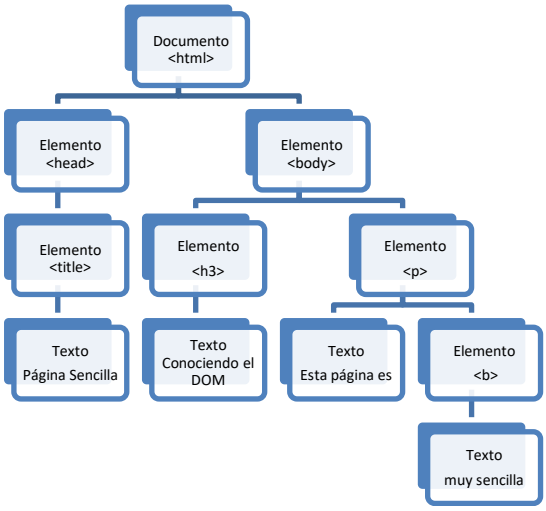
## 2. Árbol de nodos

**El modelo DOM HTML se define a través de una estructura de árbol**. Para poder manejar las utilidades del DOM, el navegador web transforma automáticamente la página web en una estructura de árbol en la que cada elemento del documento se considera un nodo. Estos nodos están interconectados y 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".

Ejemplo de una estructura de nodos generada por DOM a partir de una sencilla página HTML:

```
<html>
        <head>
                <title>Página sencilla</title>
        </head>
        <body>
                <h3>Conociendo el DOM </h3>
                <p>Esta página es <b>muy sencilla</b></p>
        </body>
</html>
```

Se transforma en el siguiente árbol de nodos en el que 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.

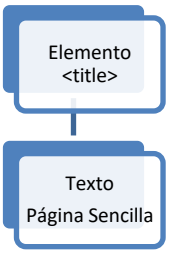



La **raíz** del árbol de nodos de cualquier página HTML siempre es la misma: un **nodo de tipo especial denominado "Documento".** A partir de ese nodo raíz, cada **etiqueta HTML** se transforma en un **nodo de tipo "Elemento"**.

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 HTML se transforma en un nodo que deriva del nodo correspondiente a su "etiqueta padre".

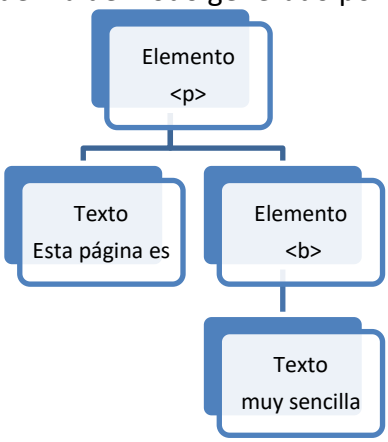
La transformación de las etiquetas HTML habituales suele generar dos nodos: el primero es el nodo de tipo "Elemento" (correspondiente a la propia etiqueta HTML) y el segundo es un nodo de tipo "Texto" que contiene el texto encerrado por esa etiqueta HTML.

Así, la siguiente etiqueta HTML:
<title>Página sencilla</title>
Genera los siguientes dos nodos:




De la misma forma, la siguiente etiqueta HTML:
Esta página es muy sencilla
Genera los siguientes nodos:

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




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

- Las etiquetas HTML que contienen texto 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 HTML se encuentra dentro de otra, se sigue el mismo procedimiento anterior, pero los nodos generados serán nodos hijo de su etiqueta padre.

✓ *El navegador transforma automáticamente el documento HTML en una estructura de árbol en la que cada elemento del documento se considera un nodo*

- ✓ *La raíz siempre es la misma, un nodo de tipo especial denominado "Documento"*
- ✓ *A partir del nodo raíz, cada etiqueta HTML se transforma en un nodo tipo "Elemento" y este a su vez tiene por hijo un nodo de tipo "Texto" con el texto encerrado en la etiqueta.*
- ✓ *Mediante las funciones proporcionadas por DOM podremos acceder a cualquier nodo de la página de forma sencilla e inmediata.*

Para ordenar la estructura del árbol existen una serie de reglas que son de obligado cumplimiento:

- En el árbol de nodos, al nodo superior (document) se le llama raíz
- Cada nodo, exceptuando el nodo raíz, tiene un padre
- Un nodo puede tener cualquier número de hijos
- Una hoja es un nodo sin hijos
- Los nodos que comparten el mismo padre son hermanos.

Estas normas siempre son respetadas para que la jerarquía tenga una integridad y poder asegurar el orden y acceso a los diferentes elementos.

## 3. Tipos de datos específicos del DOM

- ✓ *El manejo del DOM se reduce en última instancia al manejo de objetos que nos ofrecen propiedades y métodos.*

Los valores de estas propiedades y las respuestas de los métodos pueden ser:

- Los tipos intrínsecos de JavaScript que ya conocemos (String, Number, …)
- Otros objetos cuya estructura está estandarizada en el DOM. Estas estructuras pueden considerarse como **tipos de datos específicos del DOM** y algunas de las más interesantes son:
  - ***Node**: Es un tipo de datos que hace referencia a un nodo sea cual sea su tipo*
  - ***Element**: Es un tipo de datos que hace referencia a un nodo de tipo elemento*
  - ***Attribute**: Es un tipo de datos que hace referencia a un nodo de tipo atributo.*
  - ***NodeList**: Es una colección ordenada de objetos de tipo nodo. Una colección nos permite acceder a sus elementos como si fuera un array, es decir, indicando el índice entre corchetes, pero no admite ciertos métodos que sí están disponibles en los arrays.*

- ***HTMLCollection**: Un nodeList de nodos de tipo elemento en la que se puede acceder a sus elementos por el índice o por el name o id de ese elemento usando en ambos casos la sintaxis de corchetes.*

## 4. El objeto Node

Una vez que DOM ha creado de forma automática el árbol completo de nodos de la página, ya es posible utilizar sus funciones para obtener información sobre los nodos o manipular su contenido. JavaScript crea el objeto Node para definir las propiedades y métodos necesarios para procesar y manipular los documentos.

Node proporciona, entre otros, las siguientes propiedades y métodos:

| Propiedad/Método | Valor devuelto | Descripción |
|---|---|---|
| childNodes | NodeList | Lista de todos los nodos hijo del nodo actual |
| firstChild | Node | Referencia del primer nodo de la lista childNodes |
| lastChild | Node | Referencia del último nodo de la lista childNodes |
| previousSibling | Node | Referencia del nodo hermano anterior o null si este nodo es el primer hermano |
| nextSibling | Node | Referencia del nodo hermano siguiente o null si este nodo es el último hermano |
| hasChildNodes() | Bolean | Devuelve true si el nodo actual tiene uno o más nodos hijo |
| appendChild(nodo) | Node | Añade un nuevo nodo al final de la lista childNodes |
| removeChild(nodo) | Node | Elimina un nodo de la lista childNodes |
| replaceChild(nuevoNodo, anteriorNodo) | Node | Reemplaza el nodo anteriorNodo por el nodo nuevoNodo |
| insertBefore(nuevoNodo, anteriorNodo) | Node | Inserta el nodo nuevoNodo antes que la posición del nodo anteriorNododentro de la lista childNodes |

*Ejemplo 01 recorriendo el DOM.html*

```html
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>DOM</title>
        <link rel="stylesheet" href="main.css">

<script>
function comprobar(){
nodoDiv=document.getElementById("id1");
console.log('Nodo div tiene hijos ' + nodoDiv.hasChildNodes()); // true o false según tenga o no nodos hijos
hijos=nodoDiv.childNodes;  // son los nodos hijos de nodoDiv
console.log('Tiene ' +hijos.length); // cuantos tiene
console.log('Primero '+ nodoDiv.firstChild.innerHTML); // es el hijo mayor o primero
console.log('Último '+nodoDiv.lastChild.innerHTML);  // es el hijo menor o último
p=document.getElementById("p2");
console.log('Hermano izdo de p2 '+p.previousSibling.innerHTML); // es el hermano izquierdo o anterior de p
console.log('Hermano drcho de p2 '+p.nextSibling.innerHTML); // es el hermano derecho o siguiente de p
console.log('Padre de p2 '+p.parentNode.id); // es el nodo padre de p
}
</script>
</head>
<body onload="comprobar()">
<div id="id1"><p> Primero </p><p id="p2"> Segundo </p><p> Tercero </p></div>
</body>
</html>
```

## 5. Acceder y localizar elementos concretos dentro del documento

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 modos alternativos para acceder a un nodo específico:
- Acceso a través de sus nodos padre
- 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.

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 HTML se cargue por completo

Básicamente existen 4 formas de acceder a un elemento de un documento HTML:

## 5.1. A través de una propiedad directa:

Algunos elementos únicos (que solo puede existir uno dentro de la etiqueta <html>), como head y body, son accesibles directamente a través de las propiedades head y body, respectivamente, del objeto document.

*Ejemplo 02.html*

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>DOM</title>
        <link rel="stylesheet" href="main.css">

<script>
function ver(){
var nodobody=document.body;
var nodohead=document.head;
alert("body : tipo "+nodobody.nodeType+" nombre "+nodobody.nodeName); // tipo 1 nombre BODY
alert("head : tipo "+nodohead.nodeType+" nombre "+nodohead.nodeName);
// tipo 1 nombre HEAD
}
</script>
<title>Página sencilla</title>
</head>
<body onload="ver()">
<h3>Conociendo el DOM </h3>
<p>Esta página es <b>muy sencilla</b></p>
</body>
</html>
```

## 5.2. A través de una colección (HTMLCollection):

Para otros elementos que no tienen por qué ser únicos, el objeto document nos ofrece propiedades que devuelven una colección ordenada de esos elementos, como forms, links, images, scripts, styleSheets, y frames (para iFrames).

**document.forms** -> colección con todos los formularios del documento

document.forms.length nos dice cuantos
document.forms[i].name sus nombres
document.forms[i].elements HTMLCollection de sus elementos
document.forms[i].elements.length nos dice cuantos elementos tiene
document.forms[i].elements[j].name
document.forms[i].elements[j].value

Tambien admite la nomenclatura:
document.forms[“name_del_form”][“name_del_elemento”].value
document.forms[].reset() inicializa los campos del formulario

*Ejemplo 03 document.forms.html*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>DOM</title>
        <link rel="stylesheet" href="main.css">
<script>
function mostrar(){
for (i=0;i<document.forms.length;i++){
    console.log (document.forms[i].name+' '+document.forms[i].elements);
    for (j=0;j<document.forms[i].elements.length;j++)
        console.log (document.forms[i].elements[j].name+' '+document.forms[i].elements[j].value);
    }
console.log (document.forms["form1"]["campo2"].value);
}
function limpiar(){
document.forms[0].reset();
}
</script>
</head>
<body onload="mostrar()">
<form name="form1">
        Un campo <input type="text" name="campo1" value="20">
        <br><br>
        Otro campo <input type="text" name="campo2" value="40">
        <br><br>
        Y otro más<input type="text" name="campo3" value="50">
        <br><br>
</form>
<form name="form2">
        <input type="button" value="limpiar" onclick="limpiar()">
        </form>
<form name="form3">
</form>
</body>
</html>
```

**document.images** -> colección con todas las imágenes del documento

document.images.length nos dice cuantas
document.images[i].src los archivos con sus rutas

http://www.w3schools.com/jsref/dom_obj_image.asp

**document.links** -> colección de todos los enlaces del documento
document.links.length nos dice cuantos
document.links[i] son los enlaces

*Ejemplo 04 document.links.html*

### 5.3. Acceso directo mediante los métodos

- getElementById()
  Devuelve el elemento cuyo id es uno en concreto

- getElementsByName()
  Devuelve los elementos que comparten el mismo valor del atributo name

- getElementsByTagName()
  Devuelve los elementos HTML de un tipo concreto (similar a las propiedades forms, images, links, …, pero devolviendo un nodeList en lugar de una HTMLCollection)

- getElementsByClassName(),
  Devuelve los elementos que pertenecen a una clase determinada

### 5.3.1. getElementById()

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

La función getElementById() devuelve el elemento HTML 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.

### 5.3.2. getElementsByName()

La función getElementsByName() busca los elementos cuyo atributo name sea igual al parámetro proporcionado.

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, pero podría no serlo.

En cualquier caso debemos tratar los elementos devueltos como los componentes de un array de elementos.

En el caso de los elementos HTML radio button, el atributo name es común a todos los radio button que están relacionados.

*Ejemplo 05 getElementById getElementsByName.html*

### 5.3.3. getElementsByTagName()

La función getElementsByTagName(nombreEtiqueta) obtiene todos los elementos de la página HTML 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 (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.

*Ejemplo 06 document.getElementsByTagName.html*

```
<!DOCTYPE html>
<html>

    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>DOM</title>
        <link rel="stylesheet" href="main.css">

<script>
function colorear(){
var parrafos=document.getElementsByTagName("p");
for (i=0;i<parrafos.length;i++)
    parrafos[i].style.color = "orange";
}
function enlaces(){
var parrafo=document.getElementsByTagName("p")[0];
var enlaces=parrafo.getElementsByTagName("a");
for (i=0;i<enlaces.length;i++)
    alert(enlaces[i].innerHTML);
}
</script>
</head>
<body>
<p>PRIMERO
 <br> <a href="http://www.google.es/"> Google </a>
 <br> <a href="http://www.elpais.es/"> El País </a> </p>
No párrafo
<p>SEGUNDO
 <br> <a href="http://www.marca.es/"> Marca </a>
</p>
No párrafo<br>
No párrafo
<p>TERCERO</p>
<input type="button" value="Colorear párrafos" onclick="colorear()"><br><br>
<input type="button" value="Enlaces primer párrafo" onclick="enlaces()">
</body>
</html>
```

El valor devuelto es un array de nodos DOM. 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.

De manera parecida podemos trabajar con la función:
getElementsByClassName()

### 5.3.4. Gestionar los atributos de un elemento

Mediante los siguientes métodos podremos gestionar los atributos de un elemento:

- getAttribute (nombre) lee el valor del atributo nombre
- setAttribute(nombre,valor) crea/modifica el atributo nombre
- removeAttribute(nombre) elimina el atributo nombre
- hasAttribute(nombre) averigua si existe el atributo nombre

*Ejemplo 07 gestionar atributos.html*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>DOM</title>
        <link rel="stylesheet" href="main.css">
<style>
.claseParrafo {color:red;}
</style>
<script>
function crearClase(){
document.getElementById('parrafo').setAttribute('class','claseParrafo');
}
function borrarClase(){
document.getElementById('parrafo').removeAttribute('class');
}
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>
<button type='button' onclick='borrarClase();'>Borrar clase</button>
</body>
</html>
```

# 6. Modificar el DOM

Existen dos técnicas diferentes para modificar el DOM, ambas son capaces de producir los mismos resultados, pero según el contexto una puede resultar más cómoda que la otra:

## 6.1. Utilizar código HTML.

Esta técnica se basa en el uso de cadenas de caracteres que contienen código HTML.

**La manera más utilizada es a través de la propiedad innerHTML de un elemento.**

Al utilizar código HTML el navegador tiene que interpretar ese código traduciéndolo en nodos DOM y luego actualizar esos nodos en la presentación de la página.

## 6.2. Utilizar objetos de tipo elemento.

Esta técnica se basa en el **uso de objetos de tipo nodo, en la creación y eliminación de nodos.**

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.

- 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 último 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.

- replaceChild(nuevo,viejo)
  Sustituye el nodo *viejo* por el *nuevo* dentro del elemento actual.

*EJEMPLO*

```
<html>
<head>
 <script>
 var n=1;
 function añadir(){
  let p=document.createElement('p');
  let texto=document.createTextNode('Este párrafo no existía en el HTML original '+n);
  p.appendChild(texto);
  document.body.appendChild(p);
  n++;
  }
  function eliminar(){
            var p=document.getElementsByTagName('p')[0];
            document.body.removeChild(p);
  }
  function cambiar(){
let p1=document.getElementsByTagName('p')[0];
let p=document.createElement('p');
let texto=document.createTextNode('Este párrafo sustituye al anterior');
p.appendChild(texto);
document.body.replaceChild(p,p1);
 }
 function intercalar(){
let p1=document.getElementsByTagName('p')[1];
let p=document.createElement('p');
let texto=document.createTextNode('Este se intercala');
p.appendChild(texto);
document.body.insertBefore(p,p1);
 }
  </script>
</head>
<body>
<input type="button" value="Añadir nodo" onclick="añadir()">
  <input type="button" value="Elimina nodo" onclick="eliminar()">
  <input type="button" value="Cambiar nodo" onclick="cambiar()">
  <input type="button" value="Poner nodo entre 1º y 2º" onclick="intercalar()">
  <p>Primer párrafo</p>
  <p>Segundo párrafo</p>
  <p>Tercer párrafo</p>

</body>
</html>
```

```
<script>
var n=1;
function añadir(){
  var nuevo=document.createElement('p');
  nuevo.innerHTML='Este párrafo no existía en el HTML original '+n;
  padre=document.getElementById("principal");
  padre.appendChild(nuevo);
  n++;
  }
function eliminar(){
    // borra siempre el primer parrafo
    var pborrar=document.getElementsByTagName('p')[0];
    padre=pborrar.parentNode;
    padre.removeChild(pborrar);
    // busca el padre de ese párrafo y le dice que le borre el hijo
}
 function cambiar(){
    var pReemplazo=document.getElementsByTagName('p')[0];
    var pCreo=document.createElement('p');
    pCreo.innerHTML='Este párrafo sustituye al anterior';
    padre=pReemplazo.parentNode;
    padre.replaceChild(pCreo,pReemplazo);
}
function intercalar(){
    var pSeQuedaDetras=document.getElementsByTagName('p')[1];
    var pCreo=document.createElement('p');
    pCreo.innerHTML='Este se intercala';
    padre=pSeQuedaDetras.parentNode;
    padre.insertBefore(pCreo,pSeQuedaDetras);
}
 </script>
</head>
<body>
<input type="button" value="Añadir nodo al final del primer div" onclick="añadir()">
<input type="button" value="Elimina nodo (siempre el primer parrafo)" onclick="eliminar()">
<input type="button" value="Cambiar nodo (cambia el primer párrafo)" onclick="cambiar()">
<input type="button" value="Poner nodo entre 1º y 2º" onclick="intercalar()">
<div id="principal">
    <p>Primer párrafo</p>
    <p>Segundo párrafo</p>
    <p>Tercer párrafo</p>
</div>
 <div id="otro">
    <p>Primer párrafo de otro</p>
    <p>Segundo párrafo de otro </p>
    <p>Tercer párrafo de otro</p>
 </div>
</body>
```

*Ejemplo 08 crear y eliminar nodos.html*

*Ejemplo 09 crear y eliminar nodos.html*

*Ejemplo 10 Lista ordenada.html*