

[Ir directamente al contenido de esta página](#)

codexempla.org

Buscar:

Buscar

- [Inicio](#)
- [Curso](#)
- [Artículos](#)
- [Recursos](#)
- [Acerca de](#)

[Contenidos](#) [Activar](#) [Desactivar](#)

1. Breve introducción a XML
  1. [Un vistazo a SGML](#)
  2. [Sintaxis de XML](#)
  3. [XML bien formado y XML válido](#)
  4. [Referencias de entidad](#)
  5. [¿XML va a reemplazar al HTML?](#)
2. Capa de estructura del documento
  1. [DTD y espacios de nombres](#)
  2. [Diferencias entre HTML y XHTML](#)
  3. [Elementos y atributos de XHTML 1.1](#)
    1. [Elementos estructurales](#)
    2. [Elementos de texto](#)
    3. [Elementos de lista](#)
    4. [Elementos de vinculación](#)
    5. [Elementos de objetos](#)
    6. [Elementos de tabla](#)
    7. [Elementos de formulario](#)
    8. [Elementos de \*scripting\*](#)
  4. [Validación](#)
  5. [Código semántico](#)
  6. [Técnicas de accesibilidad](#)
    1. [Técnicas generales](#)
    2. [Imágenes accesibles](#)
    3. [Tablas accesibles](#)
    4. [Formularios accesibles](#)
3. Capa de presentación del documento
  1. [CSS 2.1: Introducción](#)
  2. [Sintaxis de CSS 2.1: propiedades, declaraciones y reglas](#)
  3. [Selectores y pseudoselectores](#)
  4. [Relación de propiedades para medios visuales](#)
    1. [Fuentes y texto](#)
    2. [Cajas](#)
    3. [Colores y fondos](#)
    4. [Posición y flotado](#)
    5. [Listas](#)
    6. [Tablas](#)
    7. [Contenido generado](#)
    8. [Interfaz de usuario](#)
  5. [Relación de propiedades para medios de impresión](#)
  6. [Herencia y cascada](#)
  7. [Más sobre posición y flotado](#)
  8. [El modelo de caja del W3C y el de Internet Explorer](#)
  9. [Clarificación del uso de `id` y `class`](#)
  10. [Sabios consejos para optimizar una hoja de estilo](#)
  11. [Validación](#)
  12. [Cuestiones de accesibilidad](#)
4. Capa de comportamiento del documento
  1. [Lenguajes de \*script\* y JavaScript](#)
  2. [Sintaxis básica de JavaScript](#)

1. [Variables](#)
2. [Operadores](#)
3. [Estructuras condicionales](#)
4. [Bucles](#)
5. [Funciones](#)
6. [Objetos nativos e incorporados](#)
7. [Eventos](#)
3. [El DOM](#)
  1. [getElementById y getElementsByTagName](#)
  2. [parentNode, firstChild, lastChild, nextSibling y previousSibling](#)
  3. [createElement y createTextNode](#)
  4. [appendChild, insertBefore, replaceChild, removeChild y cloneNode](#)
  5. [addEventListener](#)
4. [El BOM](#)
  1. [Window y Document](#)
  2. [Navigator y Screen](#)
5. [Técnicas de accesibilidad: el JavaScript no obstrusivo](#)

## appendChild, insertBefore, replaceChild, removeChild y cloneNode

### Tabla de contenidos

1. [Introducción](#)
2. [appendChild](#)
3. [insertBefore](#)
  1. [¿insertAfter?](#)
4. [replaceChild](#)
5. [removeChild](#)
6. [cloneNode](#)

### Introducción

Acabamos de ver cómo **crear elementos** y **nodos de texto**, pero también hemos visto que estos **nodos creados** se **mantienen** en el «limbo» del **documento** hasta que los **incluimos** en el **árbol** del **mismo**.

Para trabajar **incorporando**, **modificando** o **eliminando** **nodos**, contamos con los **métodos** que vamos a ver a continuación.

#### appendChild

Por medio de **appendChild** podemos **incluir** en un **nodo** un **nuevo hijo**, de esta manera:

```
elemento_padre.appendChild(nuevo_nodo);
```

El **nuevo nodo** se **incluye** inmediatamente **después** de los **hijos** ya **existentes** —si hay alguno— y el **nodo padre** **cuenta con** una **nueva rama**.

Por ejemplo, el siguiente código:

```
var lista = document.createElement('ul');
var item = document.createElement('li');
lista.appendChild(item);
```

crea un **elemento ul** y un **elemento li**, y **convierte** el **segundo** en **hijo** del **primero**.

#### insertBefore


**insertBefore** nos **permite** **elegir** un **nodo** del **documento** e **incluir** **otro** **antes** **que** **él**.

Su sintaxis es:

```
elemento_padre.insertBefore(nuevo_nodo, nodo_de_referencia);
```

Si tuviéramos un fragmento de un documento como éste:

```
<div id="padre">padre
  <p>Un párrafo.</p>hijo
  <p>Otro párrafo.</p>hijo
</div>
```



y quisiéramos añadir un nuevo párrafo antes del segundo, lo haríamos así:

```
// Creamos el nuevo párrafo
var nuevo_parrafo = document.createElement('p').appendChild(document.createTextNode('Nuevo párrafo.'));

// Recojemos en una variable el segundo párrafo
var segundo_p = document.getElementById('padre').getElementsByTagName('p')[1];

// Y ahora lo insertamos
document.getElementById('padre').insertBefore(nuevo_parrafo, segundo_p);
```

## ¿insertAfter?

Cuando se empieza a trabajar con el DOM, uno echa de menos un método que permita incluir un nodo detrás de otro. Como no lo hay, todo programador de JavaScript acaba creándose una función propia que haga eso exactamente. Ésta es la mía:

```
function insertAfter(e, i) {
  if (e.nextSibling) {
    e.parentNode.insertBefore(i, e.nextSibling);
  } else {
    e.parentNode.appendChild(i);
  }
}
```

Los parámetros son:

- **e**: el nodo tras el que se quiere insertar otro.
- **i**: el nodo que se quiere insertar.

Espero que sea de ayuda.

## replaceChild

Para reemplazar un nodo por otro contamos con `replaceChild`, cuya sintaxis es:

```
elemento_padre.replaceChild(nuevo_nodo, nodo_a_reemplazar);
```

Con el mismo marcado que para el ejemplo de `insertBefore`, si quisiéramos sustituir el segundo párrafo por el que creamos, lo haríamos así:

```
document.getElementById('padre').replaceChild(nuevo_parrafo, segundo_p);
```

## removeChild

Dado que podemos incluir nuevos hijos en un nodo, tiene sentido que podamos eliminarlos. Para ello existe el método `removeChild`.

La sintaxis es:

```
elemento_padre.removeChild(nodo_a_eliminar);
```

Con el mismo ejemplo anterior, **eliminar** el **segundo párrafo** 'sería algo tan sencillo como:

```
document.getElementById('padre').removeChild(segundo_p);
```

## cloneNode

Por último, podemos **crear** un **clon** de un **nodo** por **medio** de **cloneNode**:

```
elemento_a_clonar.cloneNode(booleano);
```

El **booleano** que se pasa **como parámetro define** si se quiere **clonar** el **elemento** —con el **valor false**—, o bien si se quiere **clonar con su contenido** —con el **valor true**—, es decir, el **elemento** y **todos** sus **descendientes**.

Si quisiéramos **clonar** nuestro **div** de ejemplo con el siguiente código:

```
var clon = document.getElementById('padre').cloneNode(false);
```

**clon** contendría un **elemento div**, pero de esta manera:

```
var clon = document.getElementById('padre').cloneNode(true);
```

contendría un **elemento div** con **dos párrafos** que **contendrían** los **mismos nodos** de **texto** que el **original**.

Y vista toda la teoría hasta aquí, [veamos unos cuantos ejemplos en funcionamiento de todos estos métodos](#).

- ☐
- [Curso: TdC > El DOM > appendChild, insertBefore, replaceChild...](#) [59/64]
- ☐

## Contacto

Nombre

Correo electrónico [Necesario]

Mensaje [Necesario]

Enviar

En virtud de la Ley Orgánica 15/1999 de Protección de Datos de Carácter Personal le informo de que los datos que proporcione no serán empleados para otro fin que el de responder a su mensaje. En especial, me comprometo a no cederlos a terceros ni a emplearlos para enviar información no solicitada.

## Del blog de Digital Icon

- [MapBox: una alternativa al mapa de Google](#) 18/02/2013

Probamos MapBox, una alternativa a Google Maps a la hora de incluir un mapa en una página web.

- [0000-00-00](#) 07/12/2012

Un dato importante sobre el resultado de evaluar la fecha *0000-00-00* en PHP.

- [Responsive images sin JavaScript ni PHP](#) 05/11/2012

Experimentamos con un método para servir imágenes de tamaños dependientes de la pantalla del usuario sin emplear JavaScript ni PHP.

([cc](#)) [CodexExempla.org](#), 2007–2018 [Mapa del sitio](#) | [XHTML](#) | [CSS](#) | [AA](#)