JAVASCRIPT

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java.

Heri Fernando Londoño

INCLUIR JAVASCRIPT EN XHTML

La integración de JavaScript y XHTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web.

Heri Fernando Londoño

Incluir JavaScript en el mismo documento XHTML

El código JavaScript se encierra entre etiquetas <script type="text/javascript"> </script>. Se recomienda definir el código JavaScript dentro de la cabecera del documento.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de código JavaScript en el propio documento</title>
<script type="text/javascript">
    alert("Un mensaje de prueba");
</script>
</head>
<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Definir JavaScript en un archivo externo

Las instrucciones JavaScript se pueden incluir en un archivo externo de tipo JavaScript que los documentos XHTML enlazan mediante las etiquetas <script type="text/javascript" src="archivo.js"></script>.

Se pueden crear todos los archivos JavaScript que sean necesarios y cada documento XHTML puede enlazar tantos archivos JavaScript como necesite.

Cada etiqueta <script> solamente puede enlazar un único archivo, pero en una misma página se pueden incluir tantas etiquetas <script> como sean necesarias.

Los archivos de tipo JavaScript son documentos normales de texto con la extensión .js, que se pueden crear con cualquier editor de texto.

Incluir JavaScript en los elementos XHTML

Este último método es el menos utilizado, ya que consiste en incluir trozos de JavaScript dentro del código XHTML de la página.

El mayor inconveniente de este método es que ensucia innecesariamente el código XHTML de la página y complica el mantenimiento del código JavaScript.

En general, este método sólo se utiliza para definir algunos eventos y en algunos otros casos especiales.

```
<body>
<p onclick="alert('Un mensaje de prueba')">Un párrafo de texto.</p>
</body>
```

Etiqueta noscript

Algunos navegadores no disponen de soporte completo de JavaScript, o permiten bloquearlo parcialmente o completamente.

En estos casos, es habitual que se incluya un mensaje de aviso al usuario indicándole que debería activar JavaScript para disfrutar completamente de la página.

El lenguaje HTML define la etiqueta <noscript> para mostrar un mensaje al usuario cuando su navegador no puede ejecutar JavaScript.

La etiqueta <noscript> se debe incluir en el interior de la etiqueta <body>.

```
<noscript>
<p>La página que estás viendo requiere para su funcionamiento el uso de
JavaScript.
Si lo has deshabilitado intencionadamente, por favor vuelve a activarlo.</p>
</noscript>
```

Sintaxis

- No se tienen en cuenta los espacios en blanco y las nuevas líneas.
- Se distinguen las mayúsculas y minúsculas
- No se define el tipo de las variables
- No es necesario terminar cada sentencia con el carácter de punto y coma (;)
- Se pueden incluir comentarios.

Variables

Las variables en JavaScript se crean mediante la palabra reservada var. Esta solamente se debe indicar al definir por primera vez la variable.

En JavaScript no es necesario declarar las variables. En otras palabras, se pueden utilizar variables que no se han definido anteriormente mediante la palabra reservada var.

En cualquier caso, se recomienda declarar todas las variables que se vayan a utilizar.

El nombre de una variable también se conoce como identificador y debe cumplir las siguientes normas:

- Sólo puede estar formado por letras, números y los símbolos $ (dólar) y _ (guión bajo).
- El primer carácter no puede ser un número.
- Se diferencian mayúsculas de minúsculas.
- No puede ser palaba reservada.

Heri Fernando Londoño

Tipos de variables

- **Numéricas.** Se utilizan para almacenar valores numéricos enteros o decimales. Los números decimales utilizan el carácter . (punto) para separar la parte entera y la parte decimal.
- **Cadenas de texto.** Se utilizan para almacenar caracteres, palabras y/o frases de texto. Para asignar el valor a la variable, se encierra el valor entre comillas dobles o simples, para delimitar su comienzo y su final.
- **Arrays.**

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];
var Vector = new Array();
Vector[0] = "rojo";
Vector[1] = "verde";
Vector[2] = "azul";
var Colores = new Array("rojo","verde","azul");
```

- **Booleanos.** true o false

Heri Fernando Londoño

Funciones con arreglos

toString(). Convierte un array en un string, con los elementos separados por coma.

pop(). Elimina y retorna el ultimo elemento de un array.

push(). Agrega un elemento al final del arreglo y retorna la longitud del array.

shift(). Elimina y retorna el primer elemento del array.

unshift. Agrega un elemento al inicio del array, y retorna la longitud del array.

sort(). Ordena un array alfabéticamente.

reverse(). Ordena los elementos en orden contrario a sort.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.toString();

fruits.pop();

fruits.push("Kiwi");

fruits.shift();

fruits.unshift("Lemon");

fruits.sort();

fruits.reverse();
```

Heri Fernando Londoño

Tipos de variables

- **Objetos.** Son variables que pueden contener muchos valores.

Los valores son definidos como pares nombre: valor, y en caso de necesitar ingresar varios pares, estos se separan por coma.

```
var car = {type:"Fiat", model:"500", color:"white"};
```

```
var person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

```
objectName.propertyName
```

```
objectName["propertyName"]
```

Tipos de variables

- **Objetos.**

Los objetos también pueden contener métodos. Estos son acciones que funcionan sobre los objetos y se almacenan en las propiedades como definiciones de funciones.

```
var person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

```
name = person.fullName();
```

El operador typeof

El operador typeof se emplea para determinar el tipo de dato que almacena una variable. Su uso es muy sencillo, ya que sólo es necesario indicar el nombre de la variable cuyo tipo se quiere averiguar:

```
var variable1 = 7;
typeof variable1; // "number"
var variable2 = "hola mundo";
typeof variable2; // "string"
```

Los posibles valores de retorno del operador son: undefined, boolean, number, string para cada uno de los tipos primitivos y object para los valores de referencia y también para los valores de tipo null.

Constantes Matemáticas

Constante	Valor	Significado
Math.E	2.718.281.828.459.040	Constante de Euler, base de los logaritmos naturales y también llamado número e
Math.LN2	0.6931471805599453	Logaritmo natural de 2
Math.LN10	2.302.585.092.994.040	Logaritmo natural de 10
Math.LOG2E	14.426.950.408.889.600	Logaritmo en base 2 de Math.E
Math.LOG10E	0.4342944819032518	Logaritmo en base 10 de Math.E
Math.PI	3.141.592.653.589.790	Pi, relación entre el radio de una circunferencia y su diámetro
Math.SQRT1_2	0.7071067811865476	Raíz cuadrada de 1/2
Math.SQRT2	14.142.135.623.730.900	Raíz cuadrada de 2

Heri Fernando Londoño

Conversión entre tipos de variables

- **toString(),** permite convertir variables de cualquier tipo a variables de cadena de texto.

```
var variable1 = true;
variable1.toString(); // devuelve "true“
```

- **parseInt() y parseFloat(),** convierten la variable que se le indica en un número entero o un número decimal respectivamente.

```
var variable2 = "34";
parseInt(variable2); // devuelve 34
var variable4 = "34.23";
parseFloat(variable4); // devuelve 34.23
```

Operadores

- **Asignación.** =
- **Incremento y decremento.** ++, --
- **Lógicos.** !, &&, ||
- **Matemáticos.** +, -, *, /, %
- **Relacionales.** ==, !=, >, <, >=, <=

Estructuras de control de flujo

- Estructura if
- Estructura if...else
- Estructura switch
- Estructura for
- Estructura for...in

```
for(indice in array) {
...
}
```

- Estructura while
- Estructura do while

Funciones útiles

- **length.** Calcula la longitud de una cadena de texto o un array.

```
var mensaje = "Hola Mundo";
var numeroLetras = mensaje.length;
```

- **toUpperCase().** Transforma los caracteres de la cadena en mayúsculas.

```
var mensaje1 = "Hola";
var mensaje2 = mensaje1.toUpperCase();
```

- **toLowerCase().** Transforma los caracteres de la cadena en minúsculas.

```
var mensaje1 = "HolA";
var mensaje2 = mensaje1.toLowerCase();
```

- **charAt(posicion).** Obtiene el carácter que se encuentra en la posición indicada:

```
var mensaje = "Hola";
var letra = mensaje.charAt(0);
```

Funciones útiles

- **indexOf(caracter).** Devuelve la primera posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

```
var posicion = mensaje.indexOf('a');
```

- **lastIndexOf(caracter).**
- **substring(inicio, final).** Extrae una porción de una cadena de texto. El segundo parámetro es opcional.

```
var porcion = mensaje.substring(1, 8);
```

- **split(separador).** Convierte una cadena de texto en un array. La función parte la cadena de texto determinando sus trozos a partir del carácter separador indicado.

```
var mensaje = "Hola Mundo, soy una cadena de texto!";
var palabras = mensaje.split(" ");
```

Funciones útiles

- **join(separador).** Une todos los elementos de un array en una cadena de texto de acuerdo al separador.

var array = ["hola", "mundo"];

mensaje = array.join(" ");

- **pop().** Elimina y retorna el último elemento del array.

var ultimo = array.pop();

- **push().** Añade un elemento al final del array. array.push(4);
- **shift().** Elimina y retorna el primer elemento del array.
- **unshift().** Añade un elemento al principio del array.
- **reverse().** Modifica un array colocando sus elementos en el orden inverso a su posición original. array.reverse();
- **toFixed(digitos).** Devuelve el número original con tantos decimales como los indicados por el parámetro digitos y realiza los redondeos necesarios. numero1.toFixed(2); // 4564.35

La clase Date

Permite representar y manipular valores relacionados con fechas y horas.

var fecha = new Date();

- **getTime().** Devuelve el número de milisegundos transcurridos desde la referencia de tiempos (1 de Enero de 1970).
- **getMonth().** Devuelve el número del mes de la fecha (0 para Enero y 11 para Diciembre).
- **getFullYear().** Devuelve el año de la fecha como un número de 4 cifras.
- **getYear().** Devuelve el año de la fecha como un número de 2 cifras.
- **getDate().** Devuelve el número del día del mes.
- **getDay().** Devuelve el número del día de la semana (0 Domingo, 1 Lunes, ..., 6 Sábado).
- **getHours(), getMinutes(), getSeconds(), getMilliseconds().** Devuelve respectivamente las horas, minutos, segundos y milisegundos de la hora correspondiente a la fecha.

Funciones

Las funciones en JavaScript se definen mediante la palabra reservada function, seguida del nombre de la función. Su definición formal es la siguiente:

```
function nombre_funcion(argumento1, argumento2) {
   ...
   return valor;
}
```

Después del nombre de la función, se incluyen dos paréntesis en los cuales se reciben los argumentos.

Por último, los símbolos { y } se utilizan para encerrar todas las instrucciones que pertenecen a la función.

Ámbito de las variables

Si una variable se declara fuera de cualquier función, automáticamente se transforma en variable global independientemente de si se define utilizando la palabra reservada var o no.

Si en el interior de una función, las variables se declaran mediante var se consideran locales y las variables que no se han declarado mediante var, se transforman automáticamente en variables globales.

DOM

Document Object Model o DOM permite a los programadores web acceder y manipular las páginas XHTML como si fueran documentos XML.

En la programación de aplicaciones web con JavaScript es necesaria la manipulación de las páginas web, obtener valores, agregar elementos.

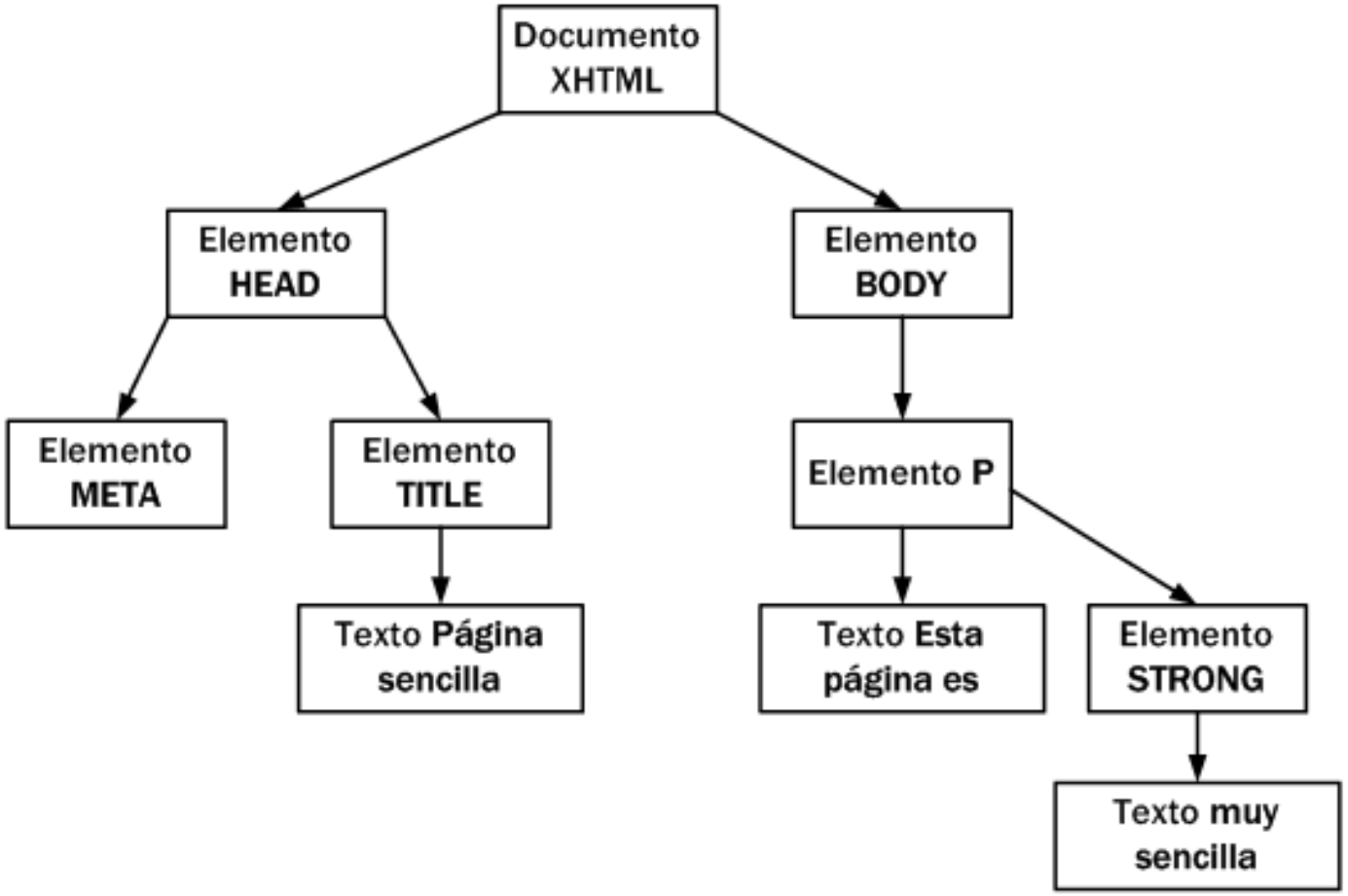
Todas estas tareas habituales son muy sencillas de realizar gracias a DOM.

Sin embargo, para poder utilizar las utilidades de DOM, es necesario "transformar" la página original. Por ello, los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular.

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".

Heri Fernando Londoño

Árbol de nodos



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.

Los otros tipos de nodos existentes son **DocumentType, CDataSection, DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation**.

Acceso directo a los nodos

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.

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

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. En este caso 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.

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

var primerParrafo = parrafos[0];

Acceso directo a los nodos

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.

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

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

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

Acceso directo a los nodos

getElementById()

La función getElementById() es la más utilizada cuando se desarrollan aplicaciones web dinámicas.

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>
```

Internet Explorer 6.0 interpreta incorrectamente esta función, ya que devuelve también aquellos elementos cuyo atributo name coincida con el parámetro proporcionado a la función.

Acceso directo a los atributos XHTML

Mediante DOM, es posible acceder a todos los atributos XHTML y todas las propiedades CSS de cualquier elemento de la página.

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.

```
var enlace = document.getElementById("enlace");
alert(enlace.href);
```

```
<a id="enlace" href="http://www...com">Enlace</a>
```

Para obtener el valor de cualquier propiedad CSS del nodo, se debe utilizar el atributo style.

```
var imagen = document.getElementById("imagen");
alert(imagen.style.margin);
```

```
<img id="imagen" style="margin:0; border:0;" src="logo.png" />
```

Eventos

Evento	Descripción	Elementos
onblur	Deseleccionar el elemento	<button>, <input>, <label>, <select>,<textarea>, <body>
onchange	Deseleccionar un elemento que se ha modificado	<input>, <select>, <textarea>
onclick	Pinchar y soltar el ratón	Todos los elementos
ondblclick	Pinchar dos veces seguidas con el ratón	Todos los elementos
onfocus	Seleccionar un elemento	<button>, <input>, <label>, <select>,<textarea>, <body>
onkeydown	Pulsar una tecla (sin soltar)	Elementos de formulario y <body>
onkeypress	Pulsar una tecla	Elementos de formulario y <body>
onkeyup	Soltar una tecla pulsada	Elementos de formulario y <body>
onload	La página se ha cargado completamente	<body>

Heri Fernando Londoño

Eventos

Evento	Descripción	Elementos
onmousedown	Pulsar (sin soltar) un botón del ratón	Todos los elementos
onmousemove	Mover el ratón	Todos los elementos
onmouseout	El ratón *"sale"* del elemento (pasa por encima de otro elemento)	Todos los elementos
onmouseover	El ratón *"entra"* en el elemento (pasa por encima del elemento)	Todos los elementos
onmouseup	Soltar el botón que estaba pulsado en el ratón	Todos los elementos
onreset	Inicializar el formulario (borrar todos sus datos)	<form>
onresize	Se ha modificado el tamaño de la ventana del navegador	<body>
onselect	Seleccionar un texto	<input>, <textarea>
onsubmit	Enviar el formulario	<form>
onunload	Se abandona la página (por ejemplo al cerrar el navegador)	<body>

Manejadores de eventos

Para que los eventos resulten útiles, se deben asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento se ejecuta el código indicado, por lo que la aplicación puede responder ante cualquier evento que se produzca durante su ejecución.

Las funciones o código JavaScript que se definen para cada evento se denominan "manejador de eventos" y como JavaScript es un lenguaje muy flexible, existen varias formas diferentes de indicar los manejadores:

- Manejadores como atributos de los elementos XHTML.
- Manejadores como funciones JavaScript externas.
- Manejadores "semánticos".

Manejadores de eventos como atributos XHTML

Se trata del método más sencillo y a la vez menos profesional de indicar el código JavaScript que se debe ejecutar cuando se produzca un evento. En este caso, el código se incluye en un atributo del propio elemento XHTML.

En el siguiente ejemplo, se quiere mostrar un mensaje cuando el usuario de clic con el ratón sobre un botón:

```
<input type="button" value=“Cliqueame" onclick="alert('Gracias por cliquearme');" />
```

Manejadores de eventos como atributos XHTML

```
<div onclick="alert('Has pinchado con el ratón');" onmouseover="alert('Acabas de pasar el ratón por encima');">
Puedes pinchar sobre este elemento o simplemente pasar el ratón por encima
</div>

<body onload="alert('La página se ha cargado completamente');">
...
</body>
```

Manejadores de eventos como funciones externas

Esta técnica consiste en extraer todas las instrucciones de JavaScript y agruparlas en una función externa. Una vez definida la función, en el atributo del elemento XHTML se incluye el nombre de la función, para indicar que es la función que se ejecuta cuando se produce el evento.

La llamada a la función se realiza de la forma habitual, indicando su nombre seguido de los paréntesis y de forma opcional, incluyendo todos los argumentos y parámetros que se necesiten.

```
function muestraMensaje() {
alert('Gracias por cliquearme');
}
<input type="button" value="Cliqueame" onclick="muestraMensaje()" />
```

Manejadores de eventos semánticos

Esta técnica es una evolución del método de las funciones externas, ya que se basa en utilizar las propiedades DOM de los elementos XHTML para asignar todas las funciones externas que actúan de manejadores de eventos. Así, el siguiente ejemplo:

La técnica de los manejadores semánticos consiste en:

1. Asignar un identificador único al elemento XHTML mediante el atributo id.
2. Crear una función de JavaScript encargada de manejar el evento.
3. Asignar la función externa al evento correspondiente en el elemento deseado.

```
function muestraMensaje() {
alert('Gracias por cliquearme');
}
document.getElementById("pinchable").onclick = muestraMensaje;
<input id="pinchable" type="button" value="Pinchame y verás" />
```

Manejadores de eventos semánticos

Lo más importante (y la causa más común de errores) es indicar solamente el nombre de la función, es decir, prescindir de los paréntesis al asignar la función:

Si se añaden los paréntesis después del nombre de la función, en realidad se está ejecutando la función y guardando el valor devuelto por la función en la propiedad onclick de elemento.

```
// Asignar una función externa a un evento de un elemento
document.getElementById("pinchable").onclick = muestraMensaje;
// Ejecutar una función y guardar su resultado en una propiedad de un elemento
document.getElementById("pinchable").onclick = muestraMensaje();
```

Manejadores de eventos semánticos

El único inconveniente de este método es que la página se debe cargar completamente antes de que se puedan utilizar las funciones DOM que asignan los manejadores a los elementos XHTML. Una de las formas más sencillas de asegurar que cierto código se va a ejecutar después de que la página se cargue por completo es utilizar el evento onload:

```
window.onload = function() {
   document.getElementById("pinchable").onclick = muestraMensaje;
}
```

Formularios

JavaScript dispone de numerosas propiedades y funciones que facilitan la programación de aplicaciones que manejan formularios. En primer lugar, cuando se carga una página web, el navegador crea automáticamente un array llamado forms y que contiene la referencia a todos los formularios de la página.

Para acceder al array forms, se utiliza el objeto document, por lo que document.forms es el array que contiene todos los formularios de la página.

document.forms[0];

Además del array de formularios, el navegador crea automáticamente un array llamado elements por cada uno de los formularios de la página. Cada array elements contiene la referencia a todos los elementos (cuadros de texto, botones, listas desplegables, etc.) de ese formulario.

document.forms[0].elements[0];

Esta forma de acceder tiene un inconveniente muy grave.

¿Qué sucede si cambia el diseño de la página y en el código HTML se cambia el orden de los formularios originales o se añaden nuevos formularios?

Heri Fernando Londoño

Formularios

Una forma de evitar los problemas del método anterior consiste en acceder a los formularios de una página a través de su nombre (atributo name) o a través de su atributo id. El objeto document permite acceder directamente a cualquier formulario mediante su atributo name:

```
var formularioPrincipal = document.formulario;
<form name="formulario" >
...
</form>
```

Los elementos de los formularios también se pueden acceder directamente mediante su atributo name:

```
var formularioPrincipal = document.formulario;
var primerElemento = document.formulario.elemento;
<form name="formulario">
<input type="text" name="elemento" />
</form>
```

Formularios

Obviamente, también se puede acceder a los formularios y a sus elementos utilizando las funciones DOM de acceso directo a los nodos.

```
var formularioPrincipal = document.getElementById("formulario");
var primerElemento = document.getElementById("elemento");
<form name="formulario" id="formulario" >
<input type="text" name="elemento" id="elemento" />
</form>
```

Propiedades de los Formularios

- **type**: indica el tipo de elemento que se trata. Para los elementos de tipo <input> (text, button, checkbox, etc.) coincide con el valor de su atributo type. Para las listas desplegables normales (elemento <select>) su valor es select-one, lo que permite diferenciarlas de las listas que permiten seleccionar varios elementos a la vez y cuyo tipo es select-multiple. Por último, en los elementos de tipo <textarea>, el valor de type es textarea.
- **form**: es una referencia directa al formulario al que pertenece el elemento.

document.getElementById("id_del_elemento").form

- **name**: obtiene el valor del atributo name de XHTML. Solamente se puede leer su valor, por lo que no se puede modificar.
- **value**: permite leer y modificar el valor del atributo value de XHTML. Para los campos de texto (<input type="text"> y <textarea>) obtiene el texto que ha escrito el usuario. Para los botones obtiene el texto que se muestra en el botón. Para los elementos checkbox y radiobutton no es muy útil.

Obtener el valor de los campos de formulario

- **Cuadro de texto y textarea.** value.

```
<input type="text" id="texto" />
var valor = document.getElementById("texto").value;
<textarea id="parrafo"></textarea>
var valor = document.getElementById(“parrafo").value;
```

- **Radiobutton y Checkbox.** Lo importante es conocer cuál se ha seleccionado se puede logar a través de la propiedad checked.
- **Select.**

```
var valor = document.getElementById("opciones").value;
```

Creación de elementos HTML simples

`document.getElementById(<id>);`	Retorna nodo según su id, si no encuentra retorna "undefined".
`document.createElement(<tag>);`	Crea un nodo de tipo elemento según el tag ingresado.
`document.createTextNode(<texto>);`	Crea un nodo de tipo texto con el texto ingresado.
`<nodo>.hasChild();`	Retorna "true" si el nodo tiene nodo/s hijos, de lo contrario retorna "false".
`<nodo>.parentNode();`	Retorna el nodo padre del nodo actual.
`<nodo>.appendChild(<nodo>);`	Agrega un nodo hijo al nodo actual.
`<nodo>.removeChild(<nodo>);`	Elimina el nodo recibido por parametro del nodo actual.
`<nodo>.removeAttribute(<atributo>);`	Elimina el atributo ingresado del nodo actual.
`<nodo>.setAttribute(<atributo>, <valor>);`	Asigna atributo al nodo actual.
`<nodo>.childNodes;`	Retorna arreglo con todos los nodos hijos del nodo actual.

Creación de elementos HTML simples

Algunos métodos utilizados para la creación de nodos dentro del DOM son.

createAttribute(nombre) Crea un nodo de tipo atributo con el nombre indicado.

createComment(texto) Crea un nodo de tipo comentario que contiene el valor indicado.

createElement(nombre_etiqueta) Crea un elemento del tipo indicado en el parámetro nombre_etiqueta.

createTextNode(texto) Crea un nodo de tipo texto con el valor indicado como parámetro

appendChild(nodo). Agrega un nodo hijo.

Los métodos createElement y createTextNode requieren llevar delante como referencia el objeto document. El metodo appendChild, lleva por referencia su elemento padre.

Heri Fernando Londoño

Creación de elementos HTML simples

Un elemento HTML sencillo 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>.

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.

Heri Fernando Londoño

Creación de elementos HTML simples

```
<a href="#" onclick="nuevo()">Nuevo Parrafo</a>
<div id="parrafos">
    <p>Parrafo inicial</p>
</div>
```

```
ind = 1;
function nuevo(){
    var div = document.getElementById("parrafos");
    var parrafo = document.createElement("p");
    var cont = document.createTextNode("Parrafo " + ind);
    parrafo.appendChild(cont);
    div.appendChild(parrafo);
    ind++;
}
```

Eliminación de nodos

Para eliminar un nodo del árbol DOM de la página solamente es necesario utilizar la función removeChild():

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

```
<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.

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.