JAVASCRIPT

**William Steven Pedraza Suarez 111515**

*Ingeniería de Sistemas, Fundación Universitaria San Martín,*

*Bogotá D.C.*

wp111517@ingenieria.sanmartin.edu.co

**Abstract –.**

I. INTRODUCCIÓN

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

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.

II. SINTAXIS

La sintaxis de un lenguaje de programación se define como el conjunto de reglas que deben seguirse al escribir el código fuente de los programas para considerarse como correctos para ese lenguaje de programación.

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

* **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.**

Una variable x puede tomar cualquier valor (String, int, doublé, boolean).

* **No es necesario terminar cada sentencia con el carácter de punto y coma (;).**
* **Se pueden incluir comentarios.**

// Para una línea

/\* Para un párrafo \*/

* **Palabras reservadas:**

Se utilizan para construir las sentencias de JavaScript por tanto no pueden ser utilizadas libremente.

break, case, catch, continue, default, delete, do, else, finally, for, function,if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

* **Inclusion del JavaScript:**

En el mismo documento, se recomienda colocarlo en el head:

<script type="text/javascript">

alert("Un mensaje de prueba");

</script>  
En un archivo Externo:

<script type="text/javascript" src="/js/codigo.js">

</script>

En los elementos del HTML:

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

III. ESTRUCTURAS DE CONTROL DE FLUJO

Son instrucciones del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*. También existen instrucciones del tipo *"repite esto mientras se cumpla esta condición"*.

Si se utilizan estructuras de control de flujo, los programas dejan de ser una sucesión lineal de instrucciones para convertirse en programas *inteligentes* que pueden tomar decisiones en función del valor de las variables.

* **Estructura if.**

Es La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación,  Se emplea para tomar decisiones en función de una condición. Su definición formal es:

if(condicion) {

...

}

La condición que controla el if() puede combinar los diferentes operadores lógicos y relacionales:

var condicion = true;

var muestra = false;

if(condicion) {

alert("Hola Mundo");

}

if(condicion == true) {

...

}

if(!condicion) {

alert("No entra a este bloque");

}

if(!muestra && condicion) {

alert("muestra el mensaje");

}

* **Estructura if...else.**

En ocasiones, las decisiones que se deben realizar no son del tipo *"si se cumple la condición, hazlo; si no se cumple, no hagas nada"*. Normalmente las condiciones suelen ser del tipo *"si se cumple esta condición, hazlo; si no se cumple, haz esto otro"*.

Para este segundo tipo de decisiones, existe una variante de la estructura if llamada if...else. Su definición formal es la siguiente:

if(condicion) {

...

}

else {

...

}

Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro del if(). Si la condición no se cumple (es decir, si su valor es false) se ejecutan todas las instrucciones contenidas en else { }. **Ej.**

var edad = 18;

if(edad >= 18) {

alert("Eres mayor de edad");

}

else {

alert("Todavía eres menor de edad");

}

La estructura if...else se puede encadenar para realizar varias comprobaciones seguidas:

if(edad < 12) {

alert("Todavía eres muy pequeño");

}

else if(edad < 19) {

alert("Eres un adolescente");

}

else if(edad < 35) {

alert("Aun sigues siendo joven");

}

else {

alert("Piensa en cuidarte un poco más");

}

* **Estructura for.**

Las estructuras if y if...else no son muy eficientes cuando se desea ejecutar de forma repetitiva una instrucción.

La estructura for permite realizar este tipo de repeticiones (también llamadas bucles) de una forma muy sencilla. No obstante, su definición formal no es tan sencilla como la de if():

for(inicializacion; condicion; actualizacion) {

...

}

La idea del funcionamiento de un bucle for es la siguiente: *"mientras la condición indicada se siga cumpliendo, repite la ejecución de las instrucciones definidas dentro del for. Además, después de cada repetición, actualiza el valor de las variables que se utilizan en la condición"*.

La "inicialización" es la zona en la que se establece los valores iniciales de las variables que controlan la repetición.

La "condición" es el único elemento que decide si continua o se detiene la repetición.

La "actualización" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

**Ej.**

var mensaje = "Estoy en un bucle";

for(var i = 0; i < 5; i++) {

alert(mensaje);

}

Normalmente, la variable que controla los bucles for se llama i, ya que recuerda a la palabra índice y su nombre tan corto ahorra mucho tiempo y espacio.

* **Estructura for...in**

Una estructura de control derivada de for es la estructura for...in. Su definición exacta implica el uso de objetos, se va a presentar la estructura for...in adaptada a su uso en arrays. Su definición formal adaptada a los arrays es:

for(indice in array) {

...

}

Si se quieren recorrer todos los elementos que forman un array, la estructura for...in es la forma más eficiente de hacerlo, como se muestra en el siguiente ejemplo:

var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"];

for(i in dias) {

alert(dias[i]);

}

Esta estructura de control es la más adecuada para recorrer arrays (y objetos), ya que evita tener que indicar la inicialización y las condiciones del bucle for simple y funciona correctamente cualquiera que sea la longitud del array. De hecho, sigue funcionando igual aunque varíe el número de elementos del array.

IV 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.

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.

* **Arbol de nodos**

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.

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"*.

**Ej.**

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<title>Página sencilla</title>

</head>

<body>

<p>Esta página es

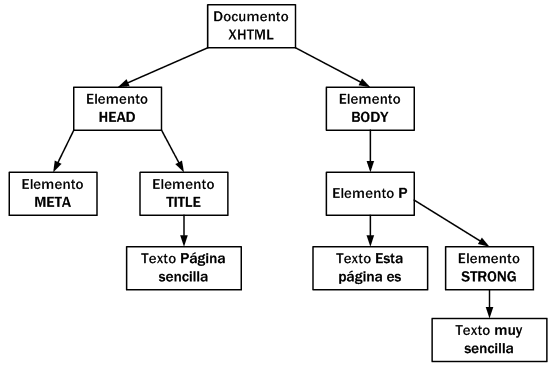
<strong>muy sencilla</strong>

</p>

</body>

</html>

Se transforma en el siguiente árbol de nodos:



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 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.

* **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 que no se van a considerar son DocumentType, CDataSection,DocumentFragment, Entity, EntityReference, ProcessingInstruction y Notation.

VI CONCLUSIONES

REFERENCIAS

[1] <http://librosweb.es/javascript/>

[2]<http://www.maestrosdelweb.com/editorial/%C2%BFque-es-javascript/>

[3]<http://si.ua.es/es/documentacion/ajax-dom/documentos/pdf/documentacion-en-pdf.pdf>