HASTA EL MOMENTO

Introducción

Creación de páginas web estáticas

Herramientas que permiten crear páginas dinámicas con mayor interacción con el usuario

JavaScript

Introducción

Si analizamos los lenguajes y tecnologías utilizadas para el desarrollo web encontramos:

JAVASCRIPT

JavaScript

Es un lenguaje de programación utilizado para crear páginas web dinámicas

Una páginas web dinámica es aquella que incorpora efectos como:

- Aparición y desaparición de elementos
- Generación de acciones al presionar botones
- Mostrar mensajes y avisos al usuario
- Animaciones

JavaScript

Es un lenguaje interpretado, no se necesita compilar el código

Es Front-end, funciona del lado del cliente

Aplicaciones más dinámicas

Los programas JS se pueden probar directamente sobre el navegador sin la necesidad de instalar otros programas

No está relacionado con el lenguaje de programación Java, que fue desarrollado de manera independiente

Es multiplataforma y open-source

Es un lenguaje de scripting multiplataforma y orientado a objetos

JavaScript

- Es dinámico:

  - Tipado dinámico: No es necesario declarar los tipos de datos

  - Las propiedades y valores de los objetos pueden ser creados, modificados y eliminados en tiempo de ejecución

- Es funcional: Las funciones son objetos en sí mismos, poseen propiedades y métodos.

JavaScript

Se puede integrar con lenguajes del lado del servidor como PHP, JSP o ASP

Hoy en día JavaScript permite

- ✓ Ejecución del lado del cliente
- ✓ Ejecución del lado del servidor

JavaScript

✓ Ejecución del lado del cliente

Proporciona objetos para controlar el navegador y su modelo de datos

Entre sus usos encontramos:

- Modificar dinámicamente el contenido de la página
- Procesar los valores de un form, validarlos a medida que son ingresados, autocompletar campos del formulario
- Ejecutar alguna acción cuando ocurre algún evento
- Generar animaciones de los elementos de la página

JavaScript

✓ Ejecución del lado del servidor

Hace ya varios años, se ha empezado a usar en desarrollos web como lenguaje del lado del servidor para la creación de aplicaciones web completas

Para ello, se requiere instalar en el servidor herramientas que permiten su uso:

- NodeJS: Es la más utilizada, y principalmente se complementa con AngularJS
- Jaxer
- RingoJS

JavaScript

Hoy en día
es el lenguaje de
programación del lado
del cliente más utilizado
en los desarrollos web y
es aceptado por todos
los navegadores

JavaScript – Inclusión en documentos

Se utiliza el elemento <script> de HTML

Los scripts pueden aparecer en la sección del header como en el body y se puede usar más de una etiqueta

Tiene dos atributos configurables “src” y “type” de acuerdo al origen del script

Se pueden incluir de dos modos:

- En el mismo documento
- En un archivo externo

JavaScript – Inclusión en documentos

- Inclusión en el mismo documento

El código JavaScript se encierra entre las etiquetas <script> y se incluye en cualquier parte del documento

Se recomienda definir el código en la sección del header

Se debe configurar el atributo “type” “text/javascript”

Se utiliza para un bloque de código pequeño o instrucciones específicas en un determinado documento HTML

Si se requiere modificar el código, debemos modificar todas las páginas que incluyen ese bloque de código

JavaScript – Inclusión en documentos

Ejemplo

```
<!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>
```

JavaScript – Inclusión en documentos

- Definir en un archivo externo

El código JavaScript se incluye en un archivo externo, que los documentos HTML referencian mediante la etiqueta <script>

Se pueden enlazar tantos archivos como sean necesarios

Se debe configurar el atributo “type” y “src” con la URL del archivo JavaScript que se quiere enlazar

Los archivos JavaScript son archivos con extensión “.js”

La ventaja es que simplifica el código HTML, se puede reutilizar el código en múltiples páginas del sitio web, y los cambios se aplican automáticamente a todos los documentos que lo enlazan

JavaScript – Inclusión en documentos

Ejemplo

Archivo `codigo.js`

```
  alert("Un mensaje de prueba");
```

Documento HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
D/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" src="/js/codigo.js"></script>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

JAVASCRIPT COMPONENTES

JavaScript – Conceptos generales

- Script:

  Cada uno de los programas, aplicaciones o bloques de código creados con el lenguaje de programación JavaScript

- Sentencia:

  Cada una de las instrucciones que forman un script

JavaScript - Sintaxis

Normas básicas:

- No se tienen en cuenta los espacios en blanco y saltos de línea
- Es case sensitive
- No se define el tipo de variable en su creación
- Variables dinámicas
- No es necesario terminar cada sentencia con punto y coma
- Dos tipos de comentarios:

```
// a continuación se muestra un mensaje
alert("mensaje de prueba");
```

```
/* Los comentarios de varias líneas son muy útiles
cuando se necesita incluir bastante información
en los comentarios */
alert("mensaje de prueba");
```

JavaScript - Variables

Se declaran mediante la palabra “var”

```
var unValor = 50;
```

No es necesario declarar las variables para usarlas

El nombre debe comenzar con un carácter o los símbolos “_” o “$”, y puede estar formado por números

Cuando no se lo inicializa, el valor por defecto es “Undefined”

El tipo de variable puede cambiar en ejecución

```
unValor = "Ahora es un string";
```

JavaScript – Tipos de datos

Los valores que puede tomar una variable son:

- Numéricos: permite almacenar números enteros o decimales

  Enteros
  - Decimales: del 0 al 9
  - Octales: del 0 al 7
  - Hexadecimales: del 0 al 9 y de A a F. Comienzan con “0x”

  Flotantes
  - La parte entera se separa de la parte decimal mediante un punto (“.”)

- Lógicos: toma valores “True” o “False”

JavaScript – Tipos de datos

- Strings:

  Permite almacenar caracteres, palabras o frases

  Se encuentran encerradas entre “” o ‘’

Caracteres especiales:

Si se quiere incluir...	Se debe incluir...
Una nueva línea	`\n`
Un tabulador	`\t`
Una comilla simple	`\'`
Una comilla doble	`\"`
Una barra inclinada	`\\`

JavaScript – Tipos de datos

- Objetos:

  Es una entidad que tiene propiedades que definen su estado, y métodos o funciones para su comportamiento

Conjunto de pares nombre:valor

```
avion={marca:"Boeing",modelo:"747",npasajeros:"45
  0"}
```

Para invocar una propiedad del objeto

`avion.modelo` → Retorna “747”

JavaScript – Tipos de datos

- Arrays:

  Es una colección de variables, que pueden ser del mismo o distintos tipos

  Es un conjunto de elementos encerradas entre corchetes y separados por comas

  Cada elemento se accede indicando su posición dentro del array, el primer elemento ocupa la posición 0

```
coches=["BMW","Mercedes","Audi","Volvo"]
```

```
ciudades=["Madrid",,"Valladolid"]
ciudades=[,,"Pamplona"]
```

- Null

- Undefined

JavaScript – Ámbito de las variables

Corresponde a la zona del programa en que se define

- Globales:

  Son declaradas en cualquier parte del programa

  Si se declaran fuera de cualquier función automáticamente se transforman en una variable global

  Pueden ser declaradas incluso dentro de funciones

  Si en el interior de una función no se declaran las variables utilizando la palabra “var”, se transforman automáticamente en variables globales

JavaScript – Ámbito de las variables

- Locales:

  Son declaradas dentro de una función específica

  Sólo se puede utilizar dentro de la función donde fue declarada

  No son reconocidas desde otras funciones o fuera de la función donde se la declara

  Si dentro de una función se define una variable con el mismo nombre que una global, las variables locales prevalecen sobre las globales dentro de la función

  La utilización de la palabra “var” es obligatoria en la declaración

JavaScript – Ámbito de las variables

¿Qué sucede en estos casos?

```
function creaMensaje() {
  mensaje = "Mensaje de prueba";
}

creaMensaje();
alert(mensaje);
```

```
function creaMensaje() {
  var mensaje = “Mensaje de prueba”;
}
creaMensaje();
alert(mensaje):
```

```
var mensaje = "gana la de fuera";

function muestraMensaje() {
  var mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

```
var mensaje = "gana la de fuera";
function muestraMensaje() {
  mensaje = "gana la de dentro";
  alert(mensaje);
}

alert(mensaje);
muestraMensaje();
alert(mensaje);
```

JavaScript - Operadores

Permiten manipular el valor de las variables, realizar operaciones matemáticas y comparar diferentes variables

- Asignación

```
var numero1 = 5;
numero1 += 3;  // numero1 = numero1 + 3 = 8
numero1 -= 1;  // numero1 = numero1 - 1 = 4
numero1 *= 2;   // numero1 = numero1 * 2 = 10
numero1 /= 5;   // numero1 = numero1 / 5 = 1
numero1 %= 4;   // numero1 = numero1 % 4 = 1
```

- Aritméticos

+	-	*	/	% (Resto)

JavaScript - Operadores

- Lógicos:

NOT	→	!
AND	→	&&
OR	→	||

- Relacionales

<	>	<=	>=	==	!=	===	!==

== Devuelve true si son iguales. Fuerza conversiones de tipo

!= Devuelve true si son distintos. Fuerza conversiones de tipo

=== Devuelve true si son iguales y del mismo tipo

!== Devuelve true si son distintos y de distinto tipo

JavaScript – Estructuras de control

- Sentencias condicionales

1. Estructura IF

```
if(condicion) {
   ...
}
else {
   ...
}
```

Ejemplo

```
var edad = 18;

if(edad >= 18) {
  alert("Eres mayor de edad");
}
else {
  alert("Todavía eres menor de edad");
}
```

2. Switch

```
switch(variable) {
   case valor_1:
      ...
      break;
   case valor_2:
      ...
      break;
   ...
   case valor_n:
      ...
      break;
   default:
      ...
      break;
}
```

Para hacer comprobaciones múltiples, para evitar el uso de muchos if anidados

JavaScript – Estructuras de control

- Sentencias iterativas

1. For

```
for(inicializacion; condicion; actualizacion) {

  ...

}
```

Ejemplo

```
var mensaje = "Hola, estoy dentro de un bucle";


for(var i = 0; i < 5; i++) {
  alert(mensaje);
}
```

JavaScript – Estructuras de control

- Sentencias iterativas

  1. For .. In

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

Ejemplo

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


for(i in dias) {
  alert(dias[i]);
}
```

JavaScript – Estructuras de control

- Sentencias iterativas

3. While

```
while(condicion) {
   ...
}
```

Ejemplo

```
var resultado = 0;
var numero = 100;
var i = 0;

while(i <= numero) {
   resultado += i;
   i++;
}

alert(resultado);
```

4. Do - While

```
do {
   ...
} while(condicion);
```

Ejemplo

```
var resultado = 1;

var numero = 5;


do {

  resultado *= numero;  // resultado = resultado * numero

  numero--;

} while(numero > 0);


alert(resultado);
```

JavaScript – Funciones

Es un bloque de código diseñado para realizar una tarea particular

- Declaración

Para definir una función se debe usar la palabra “function” seguida de su nombre y los parámetros entre paréntesis

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

El nombre tiene las mismas condiciones que la de las variables

Mediante las llaves se encierran las instrucciones pertenecientes a la función

JavaScript - Funciones

- Ejecución

El código dentro de la función es ejecutado cuando sucede alguna de las siguientes acciones

- Cuando ocurre un evento

- Cuando la función es invocada desde cualquier punto del código

- Automáticamente al cargar la página

Para invocar a la función, lo hacemos colocando el nombre seguida de los parámetros entre paréntesis

```
nombre_funcion(dato1,dato2)
```

JavaScript - Funciones

- Parámetros

Parámetros Vs Argumentos

- Los parámetros son los nombres listados en la definición de la función
- Los argumentos son los valores reales recibidos cuando la función es invocada. Se comportan como variables locales a la función

Existen métodos útiles para procesar los parámetros:

- arguments.length → Devuelve el número de argumentos
- nombre_funcion.arguments[i] → Accede a los argumentos

JavaScript - Funciones

- Return

Las funciones además de procesar información pueden devolver un resultado

Para ello, se utiliza la sentencia “Return” que permite retornar un solo elemento de cualquier tipo

Esta sentencia es la última que se ejecuta dentro de la función

JavaScript – Funciones predefinidas

Funciones matemáticas

`abs()` - Valor absoluto

`max(v1,..,vn)` - Valor máximo

`min(v1,..,vn)` - Valor mínimo

`round()` - Redondear

`exp()` - Exponencial

`log()` - Logaritmo

`pow(base,exponente)` – Potencia (exp puede ser neg.)

`sqrt()` - Raíz cuadrada

`sin()` - Seno

`cos()` - Coseno

`tan()` - Tangente

`asin()` - Arcoseno

`acos()` - Arcocoseno

`atan()` - Arcotangente

JavaScript – Funciones predefinidas

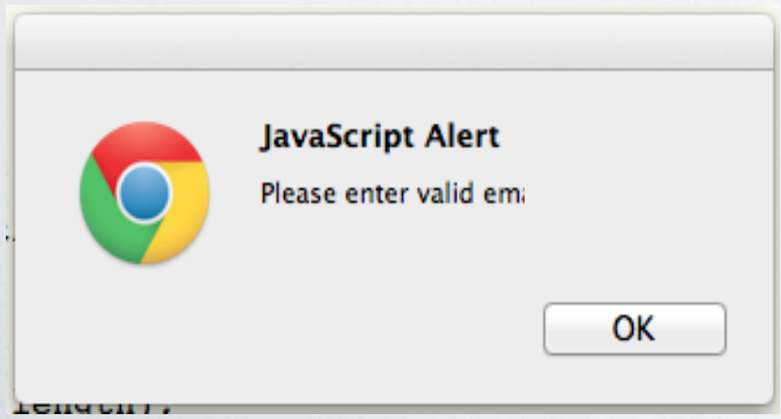
Funciones del tiempo

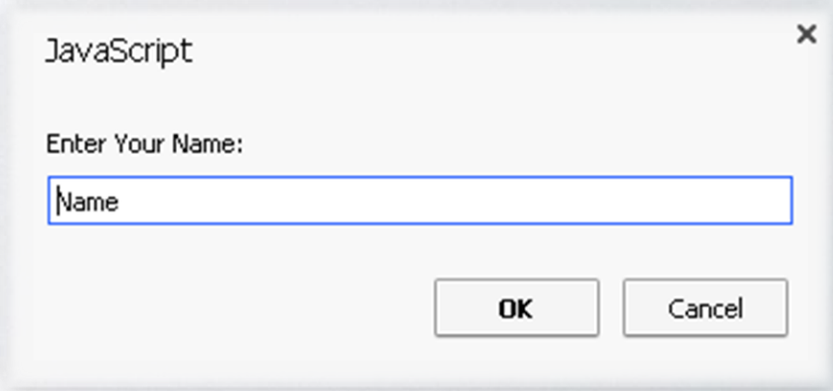
`getDate()` - Día del mes
`getDay()` - Día de la semana
`getMonth()` - Día del mes
`getYear()` - Año (con dos dígitos)
`getTime()` - Milisegundos transcurridos desde 1/1/1970
`getHours()` - Hora entre 0 y 23
`getMinutes()` - Minutos entre 0 y 59
`getSeconds()` - Segundos entre 0 y 59

JavaScript - Ventanas

Para mostrar información útil al usuario disponemos de los siguientes elementos:

- Alert: Muestra el contenido de lo que recibe por parámetros, y despliega un botón de aceptar
- Confirm: Igual al Alert, pero muestra dos botones (aceptar y cancelar). Si se evalúa el resultado, al pulsar aceptar devuelve “true”, y con cancelar “false”
- Prompt: Muestra una ventana donde se puede escribir un valor, que es devuelto por la función





JavaScript - Objetos

Es una entidad con propiedades (variables) que definen su estado, y métodos (funciones) que actúan sobre esas propiedades

Se utilizan para organizar el código fuente de manera más clara y para encapsular atributos y métodos comunes

Los objetos están formados por variables que contienen valores

Los valores son escritos como pares “nombre:valor” separadas por punto y coma (“,”)

JavaScript - Objetos

- Creación de un objeto

- Función constructora

  Una técnica común es la de definirlo a partir de un “constructor”

  En JavaScript no existe este concepto, pero se puede simular el funcionamiento mediante el uso de funciones

```
function Factura(idFactura, idCliente) {
  this.idFactura = idFactura;
  this.idCliente = idCliente;
}
```

- Otra forma de crear el mismo objeto

```
nombreObjeto = {idFactura:valor, idCliente:valor}
```

JavaScript - Objetos

- Alocación e instanciación

Una vez definida la función constructora, es posible instanciar un objeto de tipo “factura”

```
var laFactura = new Factura(3, 7);
```

- Referenciación de atributos

La forma de acceder a los atributos del objeto es la siguiente

```
alert("cliente = " + laFactura.idCliente + ", factura = " + laFactura.idFactura);
```

JavaScript - Objetos

- Subobjetos

Es posible crear objetos complejos, que contengan otros objetos

```
miFactura = { idFactura:"001234", idCliente:"46345",
              sucursal: {idLocalidad:"0001",idSucursal:"001"} }
```

```
miFactura.sucursal.idSucursal = “008”
```

- Desalocación

Para liberar memoria utilizamos la sentencia “delete”

```
delete miFactura;
```

JavaScript - Objetos

- Métodos

Un método es una función asociada a un objeto

Se puede definir mediante la notación de puntos

Los métodos se declaran sobre las propiedades del objeto

Dos maneras de definir una función

```
elObjeto.muestraId = function() {
  alert("El ID del objeto es " + this.id);
}
```

Se asigna una función anónima a una propiedad

```
function obtieneId() {
  return this.id;
}

elObjeto.obtieneId = obtieneId();
```

Se asigna una función externa ya definida a una propiedad

JavaScript - Objetos

“this”

Un concepto importante es la utilización de la palabra “this” dentro de los métodos del objeto

Hace referencia al objeto que está llamando al método, independientemente del nombre del objeto

Se pueden manipular sus propiedades y métodos

La invocación de un método se realiza del mismo modo que las propiedades, pero agregando paréntesis “()” al final

```
var id = elObjeto.obtieneId()
```

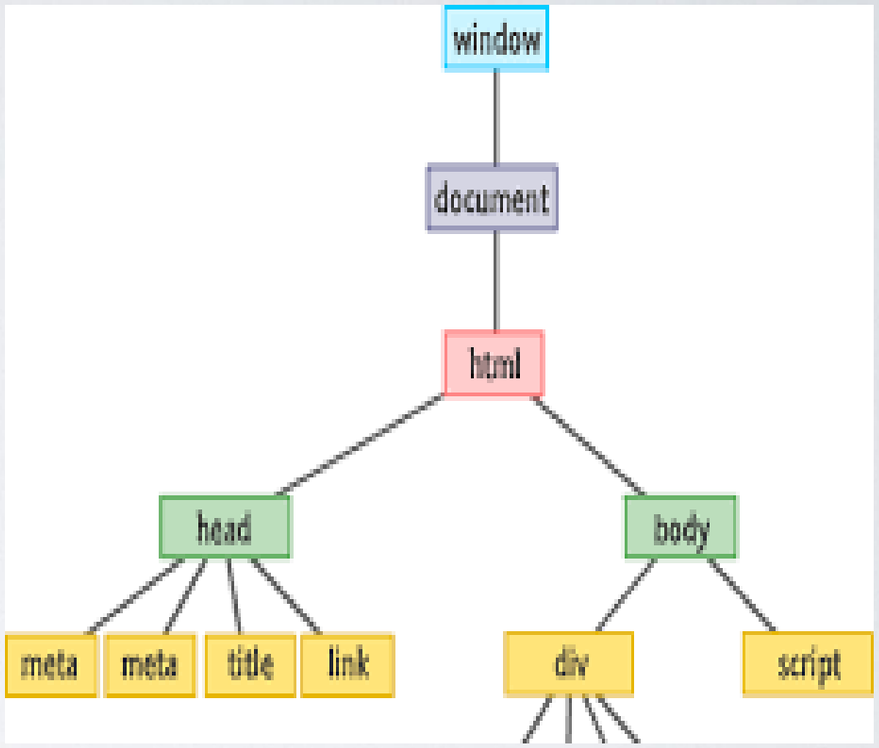
Si se accede a la propiedad “obtieneId” sin paréntesis, retorna la definición de la función y no se ejecuta

DOCUMENT OBJECT MODEL

DOM

Fue una de las herramientas que más influyó en el desarrollo de páginas web dinámicas y aplicaciones web

Es una api que provee funciones que se pueden utilizar para manipular las páginas HTML de forma rápida y sencilla



Es utilizada por el navegador para armar una nueva estructura

El navegador transforma internamente el archivo HTML original en una estructura jerárquica más fácil de manejar formada por una jerarquía de nodos, es decir a partir del código original se genera una nueva estructura donde cada elemento HTML se convierte en un nodo, y cada uno de ellos se interconecta con otros nodos formando un árbol

DOM

El árbol generado no solo representa cada uno de los elementos que forman el archivo original (nodos del árbol), sino también las relaciones entre ellos (los conectores)

El origen del DOM surgió por la necesidad de procesar fácilmente archivos XML, pero la api DOM se puede utilizar independientemente al lenguaje de programación

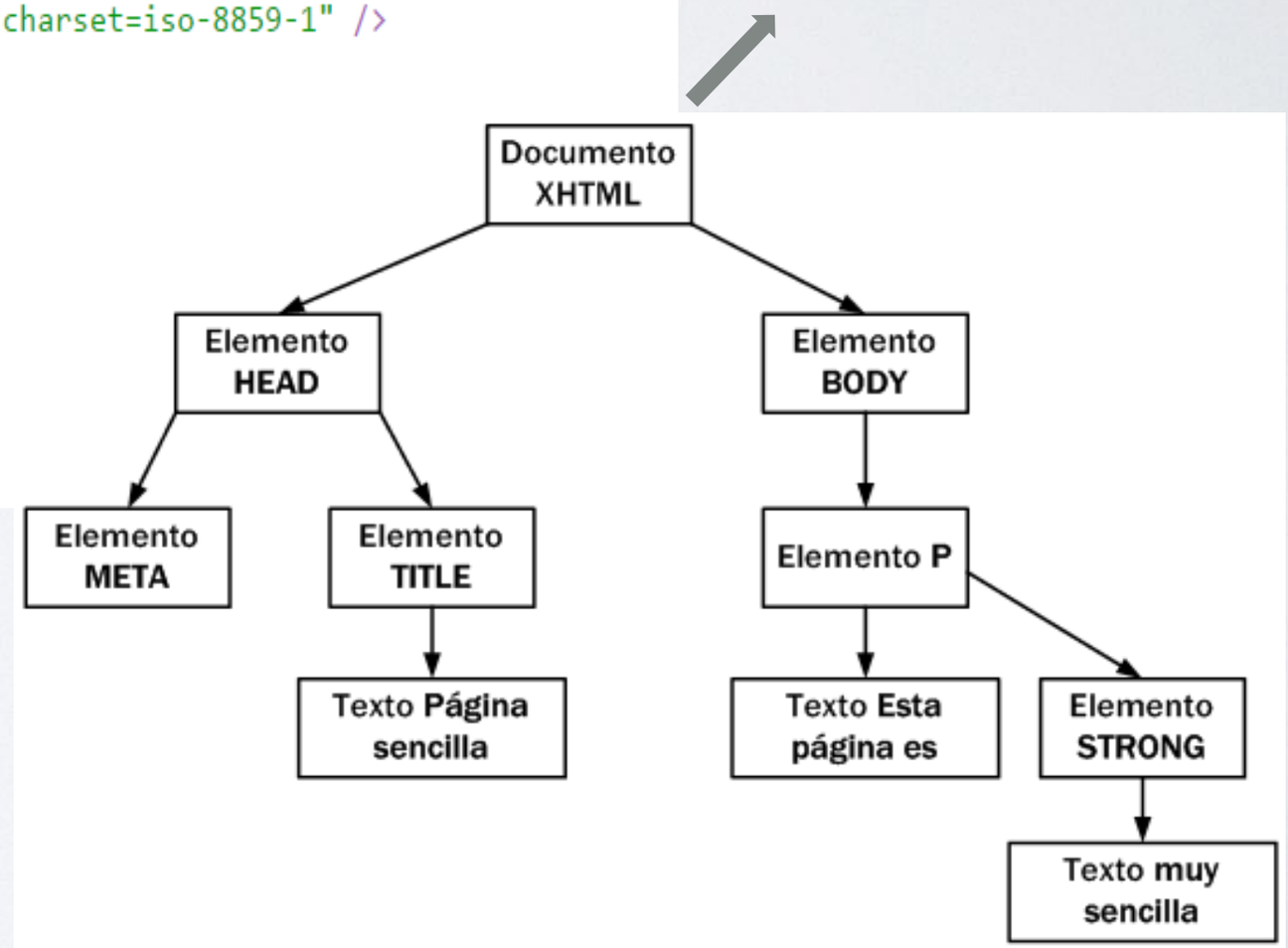
DOM

Ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
D/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>Página sencilla</title>
  </head>

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

La raíz del árbol es el nodo Document



Nodos formados por tipo y contenido

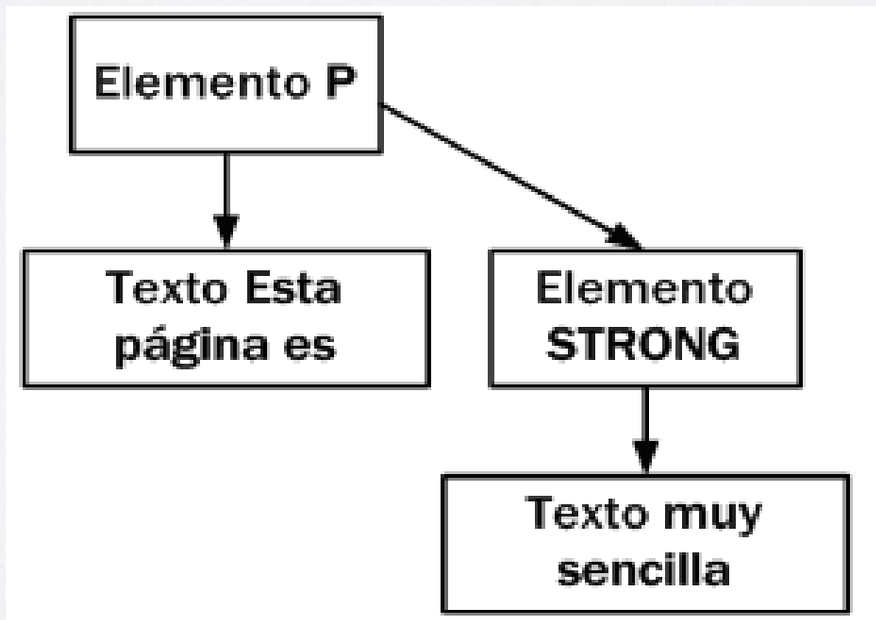
DOM

Se distinguen dos tipos de nodos:

- Elementos – Represen las etiquetas
- Textos – Contiene el texto encerrado entre esas etiquetas

Ejemplo

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



DOM

La transformación automática de la página web en un árbol de nodos se realiza siguiendo las reglas:

- Cada etiqueta HTML se transforma en dos nodos:
  - La propia etiqueta
  - El contenido dentro del la etiqueta se convierte en un nodo hijo
- Si una etiqueta HTML se encuentra dentro de otra, siguiendo la primer regla, el primer nodo generado será hijo de la etiqueta padre
- Si una etiqueta HTML tiene contenido textual, el nodo que representa ese contenido será una hoja del árbol general

DOM

Antes de poder
utilizar la api del DOM, es necesario que se construya automáticamente el árbol para ejecutar eficientemente las funciones

La página web debe cargarse completamente

DOM

- Acceso a los nodos

Una vez construido el árbol de nodos completo se pueden utilizar las funciones de la api

Acceder a un nodo equivale a acceder a un elemento de la página web

Manipulación de elementos

Existen dos alternativas:

- A través de sus nodos padres: se accede al nodo raíz de la página, después a sus nodos hijos y a los nodos hijos de esos hijos, y así sucesivamente..
- Acceso directo al nodo

DOM

- Acceso directo al nodo

La api provee las siguientes funciones:

- ✓ getElementsByTagName(nombre_etiqueta)

  Obtiene todos los elementos de la página web cuya etiqueta es igual que el parámetro que recibe

- ✓ getElementsByName(nombre_elemento)

  Selecciona los elementos cuyo atributo “name” es igual al recibido por parámetro

- ✓ getElementsById(id_elemento)

  Selecciona un elemento específico a partir de su “id”

DOM

Ejemplos:

getElementsByTagName()

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

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

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

getElementsByName()

getElementsById ()

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

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

DOM

- Acceso a los atributos HTML

Los atributos de los elementos HTML se transforman automáticamente en propiedades de los nodos

Para acceder al atributo lo hacemos mediante la notación de puntos

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

alert(enlace.href); // muestra http://www...com


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

DOM

- Acceso a las propiedades CSS

El acceso y modificación de las propiedades CSS se realiza mediante el atributo Style del elemento

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

alert(imagen.style.margin);
```

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

Si el nombre de la propiedad CSS es compuesto, para acceder a su valor se debe transformar el nombre

`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`

EVENTOS

JavaScript - Eventos

JavaScript permite que los scripts se queden esperando la ocurrencia de algún evento

Un evento es una acción que el usuario realiza como presionar una tecla, mover el mouse, cerrar la ventana del navegador, etc..

El script relacionado al evento, responderá a la acción del usuario procesando información y generando un resultado

Existen numerosos eventos que se provocan con la interacción del usuario, y a cada uno de ellos se le puede asignar una función JavaScript que se ejecuta cuando dichos eventos ocurren. Esta función se denomina “manejador de eventos”

JavaScript - Eventos

Tipos de eventos

Cada elemento tiene su propia lista de eventos que se le pueden asignar

Un mismo tipo de evento puede estar definido para varios elementos diferentes, y un mismo elemento puede tener asociados varios eventos distintos

Los nombres de los eventos llevan el prefijo “on” seguido del nombre de la acción asociada al evento

JavaScript - Eventos

Evento	Descripción	Elementos para los que está definido
`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>`

JavaScript - Eventos

Evento	Descripción	Elementos para los que está definido
`onload`	La página se ha cargado completamente	`<body>`
`onmousedown`	Pulsar (sin soltar) un botón del ratón	Todos los elementos
`onmousemove`	Mover el ratón	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>`

JavaScript – Manejadores de eventos

Existen distintas formas de definirlos:

- ✓ Manejadores como atributos de los elementos HTML
- ✓ Manejadores como funciones externas
- ✓ Manejadores semánticos

JavaScript – Manejadores de eventos

✓ Manejadores como atributos de los elementos HTML

Es el más sencillo y menos profesional

Consiste en definir atributos sobre los elementos HTML con el mismo nombre del evento que se quiere gestionar

```
<input type="button" value="Pinchame y verás" onclick="alert('Gracias por pinchar');" />
```

```
<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>
```

Desventaja: Complejo cuando se deben ejecutar un gran número de instrucciones

JavaScript – Manejadores de eventos

✓ Manejadores como funciones externas

Consiste en agrupar el código en una función externa

Se debe colocar el nombre de la función en el atributo del elemento correspondiente al evento

```
function muestraMensaje() {
  alert('Gracias por pinchar');
}


<input type="button" value="Pinchame y verás" onclick="muestraMensaje()" />
```

Si varios elementos invocan la misma función, la forma de identificar cuál fue el elemento que provocó el evento, es enviar por parámetro la variable “this” que hace referencia a ese elemento

JavaScript – Manejadores de eventos

```
function resalta(elemento) {
  switch(elemento.style.borderColor) {
    case 'silver':
    case 'silver silver silver silver':
    case '#c0c0c0':
      elemento.style.borderColor = 'black';
      break;
    case 'black':
    case 'black black black black':
    case '#000000':
      elemento.style.borderColor = 'silver';
      break;
  }
}

<div style="width:150px; height:60px; border:thin solid silver" onmouseover="resalta(this)" onm
ouseout="resalta(this)">
  Sección de contenidos...
</div>
```

JavaScript – Manejadores de eventos

Los métodos vistos hasta ahora ensucian el código HTML

Si lo que se busca es la separación de los contenidos de su aspecto, también se recomienda separar los contenidos de su comportamiento o programación JavaScript

- ✓ Manejadores semánticos

Consiste en utilizar las propiedades DOM de los elementos para asignar todas las funciones externas que actúan como manejadores de eventos

El único inconveniente es que la página se debe cargar completamente antes de que se pueda utilizar las funciones DOM, para asegurarse de eso se puede usar el evento “onload”

JavaScript – Manejadores de eventos

Ejemplo

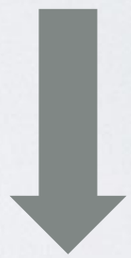
```
// Función externa
function muestraMensaje() {
  alert('Gracias por pinchar');
}

// Asignar la función externa al elemento
document.getElementById("pinchable").onclick = muestraMensaje;

// Elemento XHTML
<input id="pinchable" type="button" value="Pinchame y verás" />
```

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

No asegura la carga completa del DOM



Se coloca la función dentro del evento “onload”

JavaScript – Manejadores de eventos

Los manejadores de eventos se pueden usar para manejar y verificar:

- Entradas de usuario

  Contenido que se debe verificar cuando el usuario ingresa datos

- Acciones de usuario

  Instrucciones que se deben ejecutar cuando el usuario interactúa con la página

- Acciones del navegador

  Cosas que debería hacer cada vez que carga la página, o cuando la página se cierra

BROWSER OBJECT MODEL

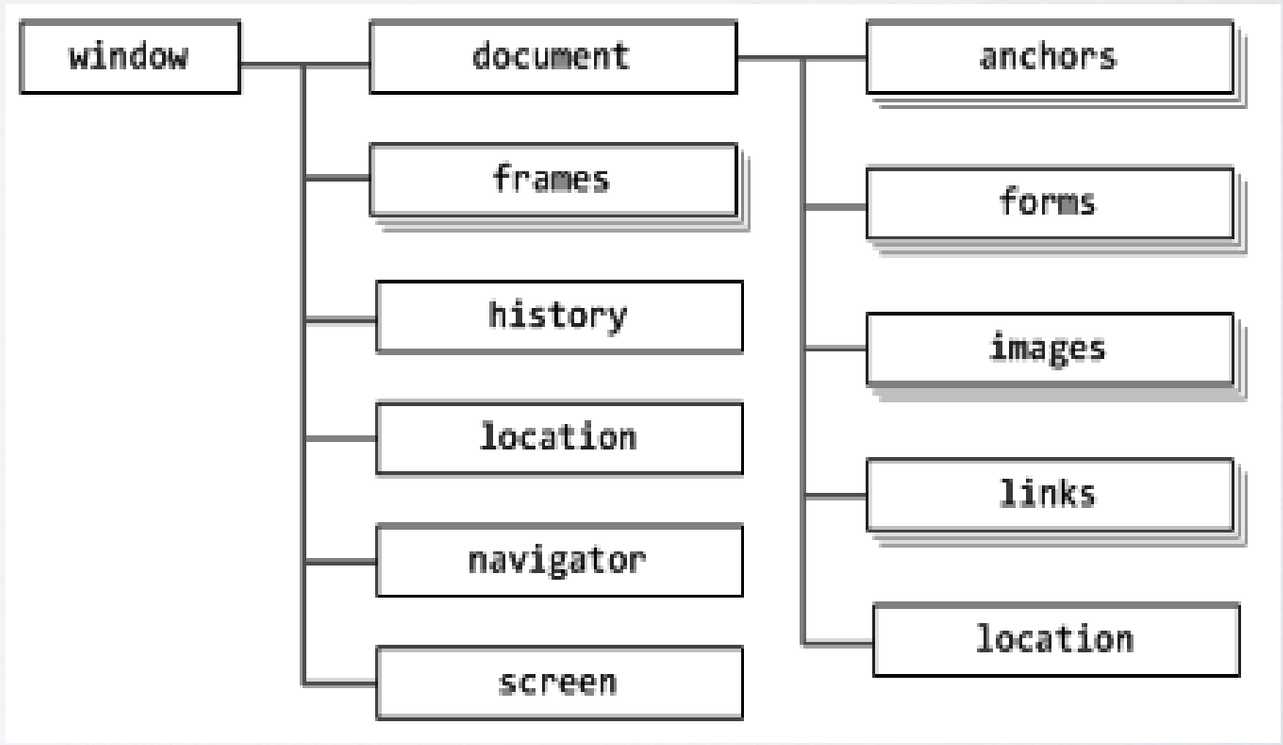
BOM

Es la herramienta que permite acceder y modificar las propiedades de la ventana del navegador

Se puede redimensionar, mover la ventana, modificar el texto de la barra de estados, etc..

Está compuesto por varios objetos relacionados entre sí

En el siguiente esquema se muestra la jerarquía de objetos que lo forman



BOM

Los objetos más importantes son:

- Window

  Representa la ventana completa del navegador. Permite redimensionar, mover y manipular la ventana actual del navegador

- Document

  Es el único elemento que pertenece tanto al BOM como al DOM, y proporciona información sobre la propia página
  Las propiedades más importantes son:

`lastModified`	La fecha de la última modificación de la página
`referrer`	La URL desde la que se accedió a la página (es decir, la página anterior en el array `history`)
`title`	El texto de la etiqueta `<title>`
`URL`	La URL de la página actual del navegador

BOM

- Navigator

  Permite obtener información útil del propio navegador

- Location

  Representa la URL de la página web que se muestra en la ventana del navegador, y brinda propiedades útiles para el manejo de las URL