Conferencia #3 JavaScript y DOM

Objetivos:

- Conocer utilidad de las tecnologías de programación Web del lado del cliente.
- Conocer la filosofía de trabajo con el DOM.
- Conocer elementos básicos de JavaScript que permitan acceder y manipular los objetos del DOM.

Contenido:

- Introducción.
- Document Object Model (DOM).
- Lenguaje JavaScript.
- Referencia a los objetos del DOM con JavaScript.
- Manipulación de eventos.
- Conclusiones.
- Motivación para la próxima clase.

Bibliografía:

1. Deitel & Deitel. “e-Business & e-Commerce. How to program.” Capítulos 20, 21. p.621-664.
2. Hernán Ruiz, Marcelo. “Programación Web Avanzada”. Capítulo 3. Colección: Manuales USERS. Editorial: MP Ediciones. Buenos Aires, Argentina. ISBN: 987-526-115-7.

Introducción

En la primera clase se comentó del surgimiento de las tecnologías del lado del cliente y del lado del servidor para proporcionarle interactividad a las páginas HTML.

Las tecnologías del lado del cliente permiten realizar aplicaciones de propósito general sobre la WWW. No están diseñadas para el desarrollo de grandes aplicaciones. Una aplicación JavaScript puede ser incrustada en un documento HTML y proporcionar un mecanismo para detectar y manipular eventos, así como validar las entradas realizadas.

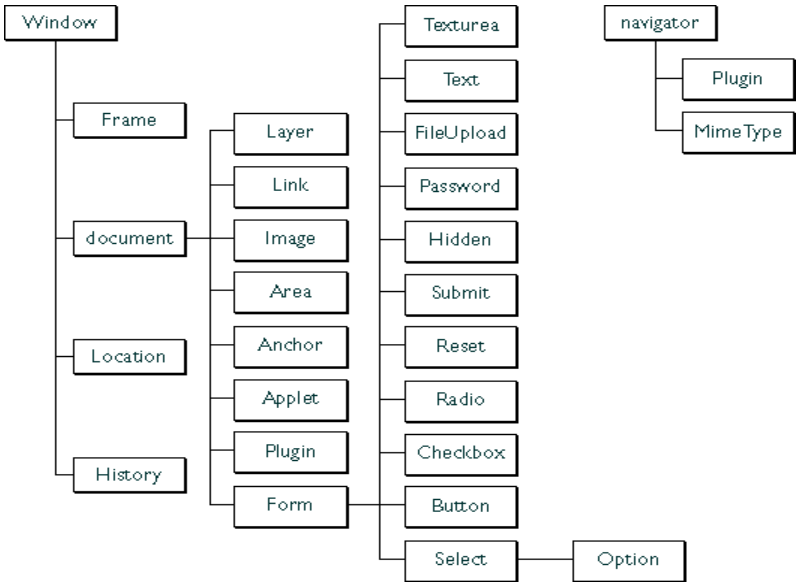
Document Object Model (DOM)

El Modelo de Objetos del Documento es una jerarquía de objetos asociados al navegador. Proporciona el control sobre la presentación de las páginas, pues permite el acceso a todos los elementos de la misma.

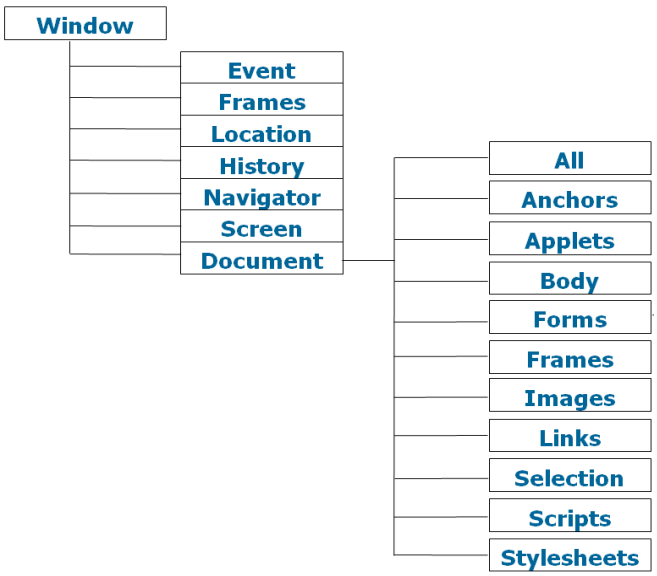
Una página Web (todos sus elementos: formularios, tablas, etc.) están representados en una jerarquía de objetos.

A través de los lenguajes scripts, es posible manipular los objetos del modelo.

En la siguiente figura se muestra el DOM de Netscape:



El DOM de Internet Explorer es el siguiente:



Principales objetos del lado cliente

- window
  - Es el objeto padre de todos los que contiene la página
  - Contiene las propiedades de la ventana
  - También se crea un objeto de este tipo por cada *frame* creado
- document
  - contiene las propiedades del documento actualmente desplegado, como su título, color de fondo, formularios, imágenes, etc.
- location
  - contiene las propiedades de la URL actual
- history
  - contiene las propiedades que representan a las URL que el usuario ha visitado anteriormente

Lenguaje JavaScript

Basado en Java fue desarrollado por Netscape para Navigator 2.0 con el nombre LiveScript. Es un lenguaje de scripts basado en objetos (pero no orientado a objetos) interpretado en el cliente. Fue diseñado para el desarrollo de aplicaciones cliente-servidor a través de Internet. MS Internet Explorer lo incorpora en su versión 3.0. Se ejecuta sobre 16 plataformas diferentes.

JavaScript es un lenguaje sensible a mayúsculas y minúsculas (case-sensitive).

El lenguaje JavaScript está pensado para agregar interactividad a las páginas HTML con el usuario o para controlar ciertos aspectos de los formularios Web, imágenes y demás elementos del navegador.

JavaScript se utiliza para ejecutar secuencias de comandos en el mismo navegador del usuario. Permite hacer cálculos rápidos y complejos, así como controlar la mayor parte de los elementos del navegador. Posibilita realizar acciones como abrir ventanas nuevas, verificar formularios antes de enviarlos, entre otras.

El código JavaScript se incorpora a un documento HTML de dos maneras:

- Como funciones y sentencias usando el elemento <SCRIPT>
- Como manipuladores de eventos usando atributos manipuladores en HTML

Podrá colocarse el script dentro del elemento <HEAD> de un fichero HTML (su contenido es procesado antes de ejecutar el código del cuerpo del documento HTML) o dentro del elemento <BODY> de un fichero HTML (su contenido es procesado en el orden en que aparece en el cuerpo del documento HTML). Se recomienda colocar el código JavaScript en el encabezamiento de la página para garantizar que este se haya descargado en el momento en que se invoque.

Variables en JavaScript

A diferencia de Java, JavaScript no exige la definición explícita de las variables. Es posible declarar la variable con sólo utilizarla por primera vez. También se pueden declarar utilizando la palabra reservada var.

Ejemplo:
```
var mi_variable;
var edad = 25;
```

Observar que no se declara tipo de dato. JavaScript asigna automáticamente el tipo, según el valor que se le asigne a la variable.

Las cadenas de caracteres se colocan entre comillas dobles (”) o simples (‘).

Operadores aritméticos

Descripción	Símbolo
Multiplicación	*

División	/
Resto de la división entera	%
Suma	+
Resta	-
Incremento	++
Decremento	--

Operadores de comparación

Descripción	Símbolo
Igualdad	==
Desigualdad	!=
Menor que	<
Mayor que	>
Menor o igual que	<=
Mayor o igual que	>=

Operadores lógicos

Descripción	Símbolo
Negación	!
Y	&&
O	||

Definición de funciones

Las funciones se definen de la siguiente manera:

```
<script language="JavaScript">
  function MiFuncion(){
        //sentencia1; ...sentenciaN;
        return valueX;
  }
</script>
```

Donde function es una palabra reservada que se escribe en minúsculas. Los paréntesis, tanto en la declaración como en la llamada a una función son obligatorios. Las variables definidas dentro del cuerpo de una función son locales a dicha función. Dos variables con el mismo nombre pueden existir en el espacio global y local de un código JavaScript, aunque esto no es aconsejable.

Ejemplo de funciones:

```
<SCRIPT language="JavaScript">
        function calcular()
        {
                var a = 3;
                var b = 7;
                var c = suma (a,b); //invocar a la función suma.
                alert(c); //mostrar resultado en un mensaje.
```

```
        }
        function suma(x,y) {    //declarar la función.
                z = x + y;
                return z;
        }
</SCRIPT>
```

La declaración de la función suma se realizó después de utilizarla, pero pudo haber sido antes.

Estructuras condicionales (If-else)

```
if (expresión)
{
     //acciones a realizar si la condición es verdadera.
 }
else
{
     //acciones a realizar si no se cumplió la condición.
}
```

Estructuras condicionales (switch)

```
switch (mi_var)
 {
     case 1:
         //acciones a realizar si mi_var es 1.
     break;

     case 2:
         //acciones a realizar si mi_var es 2.
     break;

     default:
         //acción por predefinición.
 }
```

Estructuras de iteración (for)

```
for (inicio; condición; incremento)
 {
        //código.

 }
```

En JavaScript, al igual que en la mayoría de los lenguajes de programación, hay diferentes formas de crear Estructuras de iteración: **for**, **while** y **do ... while**. La elección entre uno u otro tipo de estructura puede afectar al rendimiento de nuestro código.

Objetos predefinidos de JavaScript

A continuación se muestran algunos de los objetos predefinidos de JavaScript:

- Math: Para trabajar con funciones y constantes matemáticas.
- Array: Para el trabajo con arreglos.
- String: Para el trabajo con cadenas de caracteres.
- Date: Para obtener fechas y horas del sistema.

Notar que todos los objetos comienzan con letra mayúscula. Cada objeto tiene un conjunto de métodos definidos.

La primera posición del arreglo es 0. Una forma de crear un arreglo sería:
varArr = new Array (cantElem);

Para acceder a la cantidad de elementos: cantElem = varArr.length;
Algunos métodos del Objeto Array son:

concat(new elements)	Concatena los elementos dados al arreglo. Tambien se puede aplicar pasando como argumento otro arreglo en cuyo caso todos los elementos del mismo formaran parte del arreglo que invoca al método.
join()	Une los elementos de un arreglo utilizando un separador para formar una cadena como resultado.
pop()	Elimina y retorna el último valor del arreglo
push(new elements)	Añade los elementos al final del arreglo. myArray.push("red", "green", "yellow");
reverse()	Invierte el orden de los elementos en el arreglo.
shift()	Elimina y devuelve el primer elemento del arreglo.
slice(first element, last element).	Copia los elementos de un arreglo hacia un nuevo arreglo, puede recibir como argumentos la posición del primer elemento que se requiere copiar y la posición del último elemento +1, considerando 0 la posición del primer elemento dentro del arreglo, es decir si se requiere copiar del elemento 2 hasta el elemento 4 de un arreglo X, para el Arreglo Y la sintaxis seria la siguiente: var x = new Array(“a”, “b”, “c”, “d”, “e”); var y = x.slice(1,3); Entonces el arreglo **y** contendría los valores: b, c, d. Si se omiten los argumentos el método copia todos los

	elementos del arreglo. Si se omite el segundo argumento el método copia todos los elementos a partir del índice indicado.
sort()	Ordena un arreglo alfabéticamente, o numéricamente.
splice(index position, number of elements to remove, replacement elements**)**	Elimina un número especificado de elementos a partir de una posición dada y permite el reemplazo de estos elementos por otros nuevos. Ejemplo: Var x = new Array(“a”, “b”, “c”, “d”, “e”); x **.splice(**1,2, “maria”,”pepe”**);** El arreglo x ahora esta compuesto por los siguientes elementos: a, maria, pepe, d, e.
unshift(new elements)	Añade nuevos elementos al inicio del arreglo.

Algunos de los métodos del objeto String son:

CharAt(n)	devuelve el carácter que está ubicado en la posición n, considerando 0 la posición del primer carácter.
length:	devuelve la longitud de la cadena que la invoca.
concat(cadena2)	Agrega el contenido de cadena2 al final de la cadena invocante. No forma parte de JavaScript estándar. La forma correcta de concatenar cadenas en JavaScript es mediante el operador +.
substr(indice,n)	Devuelve una parte de la cadena contando n caracteres a partir de la posición índice.
substring(indice1, indice2)	Devuelve la subcadena comprendida entre las posiciones indice1, indice2.
toLowerCase():	Transforma la cadena a minúsculas.
toUpperCase():	Transforma la cadena a mayúsculas.

Para determinar la fecha del día de hoy sería: fecha_hoy = new Date();

Algunos de los métodos del objeto Date son:

getYear();
getMonth();
getDay();
getHours();

```
getMinutes();
getSeconds();
```

Referencia a los objetos del DOM con *JavaScript*

La manera más simple de referenciar un objeto es a través de su atributo ID. El elemento o etiqueta es representado como un objeto, y sus atributos se convierten en propiedades que pueden ser manipuladas por código script.

Para hacer una referencia a una propiedad o método de un objeto se debe incluir el objeto y todos sus ancestros separados por punto (.) Ejemplo document.formDatos.txtNombre.value, donde formDatos es el nombre del formulario, txtNombre es el nombre de un campo de entrada y value es la propiedad que permite acceder al valor del campo.

Otras formas de llegar hasta el formulario es:

- Utilizando el nombre como literal dentro de corchetes
  - document.forms['form1']
- Utilizando el número de orden dentro de corchetes
  - document.forms[0]
- Utilizando el nombre
  - document.form1

En el ejemplo anterior se utilizó la tercera variante.

Incluido en el DOM están las colecciones, las cuales son básicamente arreglos de objetos relacionados dentro de una página. Existen numerosas colecciones en el modelo de objetos, una de ellas es la colección ***all***, la cual incluye a todos los elementos HTML de un documento, en el orden en que estos aparecen. Esto proporciona una vía fácil para referirse a un elemento específico, especialmente si no tiene un ID.
La colección ***all*** es una propiedad del objeto documento. La propiedad ***length*** de esta colección (y de otras colecciones) especifica el número de elementos que forman parte de ella. Dos propiedades importantes para trabajar con las etiquetas HTML son las siguientes:

innerText: Permite acceder al texto dentro de una etiqueta.
Ej: <P id = “pText”> Welcome to our web page</p>
valorTexto = pText.innerText ; // valorTexto tomaría el valor Welcome to our web page

tagName: Permite acceder al nombre de una etiqueta.
Ej: document.all[0].tagname // devuelve el nombre de la primera etiqueta html del documento.

Cuando hacemos uso de la colección ***all***, nos referimos a todos las etiquetas HTML del documento. Sin embargo, todos las etiquetas tiene su propia colección ***all***, que consiste en todas aquellas contenidas adentro. Veamos este fragmento de código:

```
<BODY>
<P ID = “pText”> Elements on this web page:</P>
```

```
</BODY>
```

En este caso, la colección ***all*** para la etiqueta BODY contiene al elemento P.

Otra colección similar a ***all*** es ***children***, en la cual se tienen todos los elementos que son hijos directos de otro. Por ejemplo, el elemento HTML sólo tiene dos hijos: HEAD y BODY.

Durante la ejecución del código de un script existen ciertas sentencias que consumen más tiempo que sentencias equivalentes. El ejemplo más claro es el acceso a elementos de un objeto de la página HTML. Por ejemplo, se tiene que reasignar el índice del elemento seleccionado de una lista desplegable:

```
document.miFormulario.miLista.selectedIndex = 0
```

A priori, no parece que esta sentencia vaya a requerir mucho tiempo de ejecución. Sin embargo, si esta misma sentencia está en el interior de un ciclo que se repite... 1000 veces:

```
for (n = 0; n < 1000; n++) {
 ...
 document.miFormulario.miLista.selectedIndex = n
 ...
}
```

Ahora, el esfuerzo se ha multiplicado por 1000. La codificación anterior no es eficiente, ya que el acceso al objeto document.miFormulario.miLista se reptie 1000 veces.

Las llamadas repetidas a objetos de la página pueden ralentizar la ejecución del script, consumiendo excesiva cantidad de memoria. La solución: cachear el objeto.

Cachear el objeto es almacenarlo dentro de una variable. Si se hace uso de esta variable en lugar de referencias directas al objeto, se obtiene un ahorro substancial de tiempo de ejecución:

```
var lista = document.miFormulario.miLista
for (n = 0; n < 1000; n++) {
 ...
 lista.selectedIndex = n;
 ...
}
```

Es conveniente cachear objetos cuando se accede a elementos de objetos cuya estructura jerárquica es muy profunda, como el **document.all**

Objetos del navegador

El objeto *window* tiene los siguientes métodos:

- *open*(): permite abrir nuevas ventanas con una URL, un nombre y una serie de características dadas.
    - Ejemplo: window.open(“http://nombre_sitio”, “título”, “características”);
- *close*(): cierra la ventana actual.

- *alert*(): muestra una ventana de alerta con el mensaje pasado por parámetro.
    - Ejemplo alert("Debe entrar el nombre del cliente"); muestra una ventana como la siguiente:

- *confirm*(): muestra una ventana de confirmación
- *prompt*(): muestra una ventana donde se pregunta por un texto
- *blur*() and *focus*(): Pierde y toma el foco
- location: permite conocer o especificar la dirección URL.
    - Ejemplo: Redirigir el navegador a la página de google:
    - window.location.href = "http://www.google.com";

Objeto *document*

- El navegador crea un objeto *document* por cada página HTML
- Métodos
    - write(): Escribe una o más expresiones HTML en la ventana (*window*) especificada
- Propiedades
    - *forms*: Arreglo que contiene una entrada por cada formulario del documento

Objeto *form*

- El navegador crea un objeto de este tipo para cada formulario del documento
- Métodos
    - *reset*: Simula el clic sobre el botón restablecer
    - *submit*: Envía el formulario
- Propiedades
    - *elements*: Arreglo de los elementos del formulario
    - *length*: Cantidad de elementos

Manipulación de eventos

El HTML dinámico con la manipulación de eventos, permite que el código script que se programe responda a determinadas acciones del usuario. Propician que las aplicaciones sean más amistosas con los usuarios y reducen la carga del servidor.

Con el modelo de eventos, los scripts pueden responder a las acciones del usuario, como pueden ser el movimiento del mouse, hacer scroll, o presionar determinada tecla. Esto permite que el contenido se vuelva más dinámico mientras las interfaces se vuelven más intuitivas.

Los eventos son acciones que ocurren usualmente como resultado de algo que hace el usuario. Por ejemplo, cuando el usuario hace clic sobre un objeto se genera el evento onClick. Las aplicaciones JavaScript son manipuladas por los eventos que ofrece el navegador.

Los eventos más importantes son:

- OnClick: Se dispara cuando el usuario hace click con el mouse en un elemento.
- OnLoad: Se dispara cada vez que un elemento se termina de cargar exitosamente.
- OnError: Se dispara cuando ocurre un error en la página.
- OnMouseMove: Se dispara mientras el mouse está en movimiento.
- OnMouseOver\OnMouseOut: Cuando el cursor del mouse se mueve sobre un elemento\abandona un elemento.
- OnFocus\OnBlur: Cuando un elemento gana\pierde el foco.
- OnSubmit\OnReset: Cuando se le hace submit\reset a un formulario.

Sintaxis general del manipulador

<*Elemento ManipuladorDeEvento* = "*Código_JavaScript*">
Donde:
Elemento: es un elemento HTML (Form)
ManipuladorDeEvento: es el nombre del evento a manipular
Código_JavaScript: Código *script* que será ejecutado

Ejemplos de manipulación de eventos:

Supongamos que creamos una función en JavaScript llamada calcular(). Podemos hacer que el navegador ejecute esta función cuando el usuario presione un botón. Esto se hace asignando la función al evento onClick del botón:

<INPUT TYPE="button" VALUE="Calcular" onClick="calcular()">

Otros ejemplos de manipuladores de eventos son:

<INPUT TYPE = "button" VALUE = "Click Me!" ONCLICK = "alert ('Hi again')">

<INPUT TYPE = "button" VALUE = "Click Me!" ONCLICK = "ValidateForm()">

Es posible utilizar cualquier sentencia JavaScript para manipular el evento. Para incluir más de una sentencia, se deben separar con punto y coma (;). Es una buena costumbre definir funciones para manejar eventos, ya que hace más modular el código y más legible.

Validación de Formularios

Llegado a este punto en la clase, sería muy fácil realizar la validación de un formulario. A continuación se muestra un ejemplo en el que se pretende validar que el campo nombre no quede vacío:

Para acceder al valor de un campo text (txt_nombre) del formulario se hace de la siguiente manera

document.forms.frm_ejemplo.elements.txt_nombre.value

o simplemente:

document.frm_ejemplo.txt_nombre.value

Por tanto, la función quedaría:

```
function ValidarEntradaNombre() {
	if (document.frm_ejemplo.txt_nombre.value == "") {
		alert('Debe entrar el nombre');
		document.frm_ejemplo.txt_nombre.focus();
		return;
	}
}
```

Conclusiones

En la clase hoy se estudió el lenguaje JavaScript, que permite agregar interactividad con el usuario a las páginas HTML. Se vio la forma de referenciar a los objetos del DOM a través del uso de JavaScript, así como la forma de manipular los eventos.

Se estudió también el Modelo de Objetos del Documento (DOM), que es quien proporciona el control sobre la presentación de las páginas, pues permite el acceso a todos los elementos de la misma.

Motivación para la próxima clase

En la próxima clase varemos ejercicios de JavaScript donde manipularemos el DOM. Podremos ver un caso específico de validación de formulario.