UTN.BA FACULTAD REGIONAL BUENOS AIRES
SECRETARÍA DE EXTENSIÓN UNIVERSITARIA FRBA UTN

Centro de e-Learning

Javascript, JQuery y JSON avanzado.

www.sceu.frba.utn.edu.ar/e-learning

Módulo 1

INTRODUCCIÓN A JQUERY

Introducción al lenguaje Javascript.

Introducción al DOM.

Presentación de la Unidad:

En esta unidad aprenderán que es el Modelo de objetos del documento, DOM (Document Object Model).

Aprenderán como manipular los elementos HTML por el tipo de etiqueta o por su id, como modificar con Javascript atributos de CSS o como agregar dinámicamente elementos HTML.

Ya que JQuery es un framework Javascript esta unidad les servirá de base para entender ciertos conceptos que veremos luego en JQuery.

Objetivos:

- Aprender a seleccionar desde Javascript elementos HTML por su etiqueta o por su id.
- Aprender a modificar atributos CSS desde Javascript.
- Aprender a agregar dinámicamente elementos HTML en una página.

Temario:

CONSIGNAS PARA EL APRENDIZAJE COLABORATIVO

En esta Unidad los participantes se encontrarán con diferentes tipos de consignas que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

1. Los foros asociados a cada una de las unidades.
2. La Web 2.0.
3. Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen las actividades sugeridas y compartan en los foros los resultados obtenidos.

Introducción a la manipulación del DOM

Una página web es un documento HTML que es interpretado por los navegadores en forma gráfica, pero también permiten el acceso al código.

El Modelo de Objetos del Documento (DOM) permite ver el mismo documento de otra manera, describiendo el contenido del documento como un conjunto de objetos que un programa Javascript puede actuar sobre ellos.

¿Qué es el Modelo de Objetos del Documento?

Acorde al W3C el Modelo de Objetos del Documento es una interfaz de programación de aplicaciones (API) para documentos validos HTML y bien construidos XML. Define la estructura lógica de los documentos y el modo en que se accede y manipula.

El DOM permite un acceso a la estructura de una página HTML mediante el mapeo de los elementos de esta página en un árbol de nodos. Cada elemento se convierte en un nodo y cada porción de texto en un nodo de texto. Para comprender más fácilmente véase el siguiente ejemplo:

```
<body>
<p>Esto es un párrafo que contiene <a href="#">un enlace</a> en el
medio. </p>
<ul>
<li>Primer punto en la lista</li>
<li>Otro punto en la lista</li>
</ul>
</body>
```

Como puede verse un elemento [a] se encuentra localizado dentro de un elemento [p] del HTML, convirtiéndose en un nodo hijo, o simplemente hijo del nodo [p], de manera similar [p] es el nodo padre. Los dos nodos li son hijos del mismo padre, llamándose nodos hermanos o simplemente hermanos.

Es importante comprender la diferencia entre elementos y nodos de textos. Los elementos comúnmente son asociados a las etiquetas. En HTML todas las etiquetas son elementos, tales como <p>, <img> y por lo que tienen atributos y contienen nodos hijos. Sin embargo, los nodos de textos no poseen atributos e hijos.

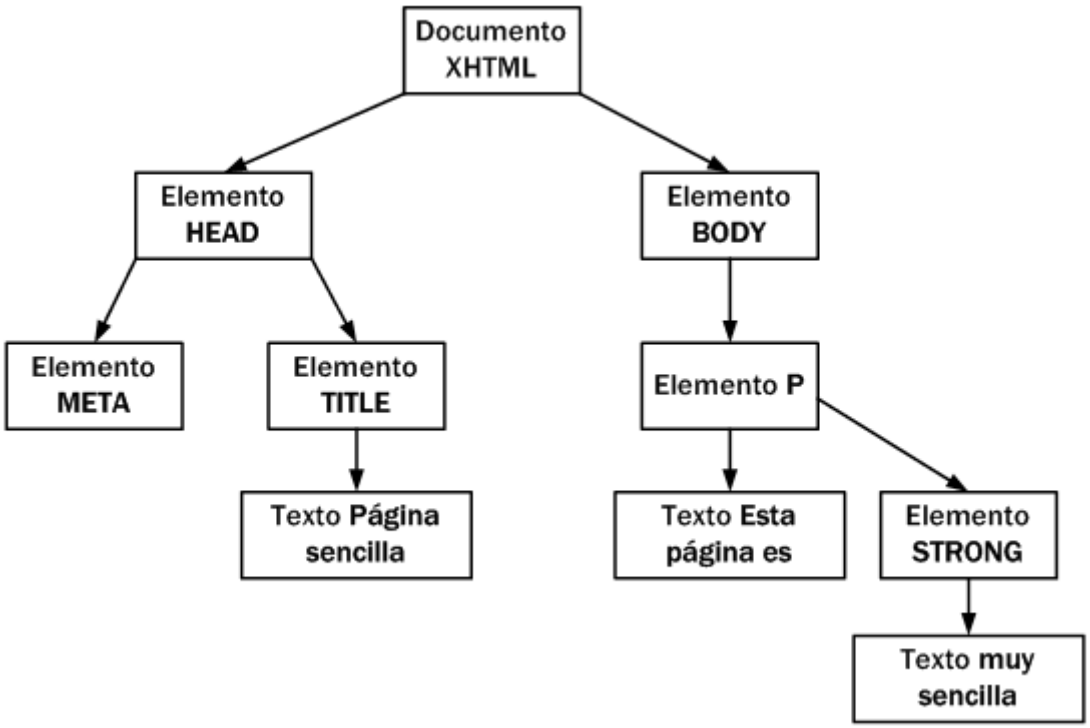
Los navegadores web transforman automáticamente todas las páginas web en una estructura más eficiente de manipular. Esta transformación la realizan todos los navegadores de forma automática y nos permite utilizar las herramientas de DOM de forma muy sencilla. El motivo por el que se muestra el funcionamiento de esta transformación interna es que condiciona el comportamiento de DOM y por tanto, la forma en la que se manipulan las páginas.

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"*.

La siguiente página XHTML sencilla:

```
<!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>Página sencilla</title>
</head>
<body>
<p>Esta página es <strong>muy sencilla</strong></p>
</body>
</html>
```

Se transforma en el siguiente árbol de nodos:



Árbol de nodos generado automáticamente por DOM a partir del código XHTML de la página.

En el esquema anterior, cada rectángulo representa un nodo DOM y las flechas indican las relaciones entre nodos. Dentro de cada nodo, se ha incluido su tipo y su contenido. 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 de las etiquetas XHTML habituales genera dos nodos: el primero es el nodo de tipo *"Elemento"* (correspondiente a la propia etiqueta XHTML) y el segundo es un nodo de tipo *"Texto"* que contiene el texto encerrado por esa etiqueta XHTML.

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`.

Siempre use el DOCTYPE correcto

El DOCTYPE (abreviado del inglés “document type declaration”, declaración del tipo de documento) informa cual versión de (X)HTML se usará para validar el documento; existen varios tipos a seleccionar. El DOCTYPE, debe aparecer siempre en la parte superior de cada página HTML y siendo un componente clave de las páginas web “obedientes” a los estándares.

En caso de usarse un DOCTYPE incompleto, no actualizado o simplemente no usarlo llevará al navegador a entrar en modo raro o extraño, donde el navegador asume que se ha programado fuera de los estándares.

Todavía todos los navegadores actuales no son capaces de procesar correctamente todos los tipos de documentos, sin embargo, muchos de ellos funcionan correctamente en los navegadores más utilizados actualmente, tales como:

HTML 4.01 Strict y Transitional, XHML 1.0 Strict y Transitional los se comportan del

modo correcto en Internet Explorer (versión 6, 7 Beta), Mozilla y Opera 7. De ahora en adelante se adoptará para cada ejemplo HTML 4.01 Strict :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

Resultando una única línea de código, o dos líneas con un salto de línea después de EN".

La importancia de validar el HTML

Si los elementos son anidados de manera inadecuada pueden generarse problemas, véase la siguiente línea:

```
<p>Estos elementos han sido <strong>incorrectamente </p>anidados
</strong>
```

El árbol que resulta de esto se encuentra incorrectamente anidado del todo, por tanto generará errores inesperados en los navegadores. Manteniendo su HTML válido se pueden evitar tales problemas.

Accediendo a los elementos

Afortunadamente, Javascript permite acceder a cada uno de los elementos de una página utilizando tan sólo algunos métodos y propiedades.

Si desea encontrar de manera rápida y fácil un elemento se tiene a la mano el método getElementById. El mismo permite un acceso inmediato a cualquier elemento tan sólo conociendo el valor de su atributo id. Véase el siguiente ejemplo:

```
<p>
<a id="contacto" href="contactos.html">Contáctenos</a>
</p>
```

Puede usarse el atributo id del elemento a para acceder al mismo:

```
var elementoContacto = document.getElementById("contacto");
```

Ahora el valor de la variable elementoContacto está referida al elemento [a] y cualquier operación sobre la misma afectará el hiperenlace.

El método getElementById es adecuado para operar sobre un elemento en específico, sin embargo, en ocasiones se necesita trabajar sobre un grupo de elementos por lo que en este caso puede utilizarse el método getElementsByTagName. Este retorna todos los elementos de un mismo tipo. Asumiendo la siguiente lista desordenada:

```
<ul>
<li><a href="editorial.html">Editorial</a></li>
<li><a href="semblanza.html">Autores</a></li>
```

```
<li><a href="noticias.html">Noticias</a></li>
<li><a href="contactos.html">Contátenos</a></li>
</ul>
```

Puede obtenerse todos los hipervínculos de la siguiente manera:

```
var hipervinculos= document.getElementsByTagName("a");
```

El valor de la variable hipervinculos es una colección de elementos [a]. Las colecciones son arreglos pudiéndose acceder a cada elemento a través de la ya conocida notación con corchetes.

Los elementos devueltos por getElementsByTagName serán ordenado según el orden que aparezcan en el código fuente. Por tanto para el caso anterior quedaría así:

- hipervinculos[0] el elemento [a] para “Editorial”
- hipervinculos[1] el elemento [a] para “Autores”
- hipervinculos[2] el elemento [a] para “Noticias”
- hipervinculos[3] el elemento [a] para “Contáctenos”

Otra maneras de acceder a un elemento usando su id es document.all["id"] la cual fue introducida en Internet Explorer 4 y document.layers["id"] introducida por Netscape 5 por que el W3C todavía no había estandarizado la manera de acceder a los elementos mediante su id. Sin embargo, no se recomienda su uso porque al estar fuera de los estándares actuales hay navegadores que no soportan estos métodos.

Por otro lado existen varios elementos en un documento HTML que pueden ser accedidos de otras maneras. El elemento body de un documento puede accederse a través de la forma document.body, mientras que el conjunto de todos los formularios en un documento puede encontrase en document.forms, así mismo el conjunto de todas las imágenes sería mediante document.images.

Actualmente la mayoría de los navegadores soportan esto métodos aún así es recomendable el uso del método getElementsByTagName, véase el siguiente ejemplo para acceder al elemento body:

```
var body = document.getElementsByTagName("body")[0];
```

Creando elementos y textos

La creación de nodos es posible mediante el uso de dos métodos disponibles en el objeto document. Dichos métodos son:

- createElement(Tipo cadena): Crea un nuevo elemento del tipo especificado y devuelve un referencia a dicho elemento.
- createTextNode(Cadena de texto): Crea un nuevo nodo de texto con el contenido especificado en la cadena de texto.

El siguiente ejemplo muestra cómo se crea un nuevo elemento de párrafo vacío:

```
var nuevoEnlace = document.createElement("a");
```

La variable nuevoEnlace ahora referencia un nuevo enlace listo para ser insertado en el documento. El texto que va dentro del elemento [a] es un nodo de texto hijo, por lo que debe ser creado por separado.

```
var nodoTexto = document.createTextNode("Semblanza");
```

Luego si desea modificar el nodo de texto ya existente, puede utilizarse la propiedad nodeValue, esta permite coger y poner el nodo de texto:

```
var textoViejo = nodoTexto.nodeValue;

nodoTexto.nodeValue = "Novedades";
```

El valor de la variable textoViejo es ahora "Semblanza" y el nuevo texto "Novedades". Se puede insertar un elemento o texto (nodo) como último hijo de un nodo ya existente usando el método appendChild. Este método coloca el nuevo nodo después de todos los hijos del nodo.

```
NuevoEnlace.appendChild(nodoTexto);
```

Ahora todo lo que se necesita es insertar el enlace en el cuerpo del documento. Para hacer esto, se necesita una referencia al elemento body del documento, teniendo como guía los estándares siguientes:

```
var cuerpoRef = document.getElementsByTagName("body")[0];
cuerpoRef.appendChild(nuevoEnlace);
```

Otra manera sería utilizando el método getElementById. Para ello se asume que la etiqueta <body> tiene asignado un valor para el atributo id.

```
<body id=”cuerpo”>
var cuerpoRef = document.getElementById("cuerpo");
cuerpoRef.appendChild(nuevoEnlace);
```

Existen básicamente tres maneras mediante las cuales un nuevo elemento o nodo de texto puede ser insertado en una página Web. Todo ello depende del punto en el cual se desee insertar el nuevo nodo: como último hijo de un elemento, antes de otro nodo o reemplazo para un nodo.

El caso de apertura de un nuevo hijo ya fue visto en el ejemplo anterior, luego para insertar el nodo antes de otro nodo se realiza utilizando el método insertBefore de su elemento padre, mientras que el reemplazo de nodo se utiliza el método replaceChild de su elemento padre.

Al usar insertBefore, se necesita tener referencias al nodo que va ser insertado y donde va a ser insertado, considérese el siguiente código HTML:

```
<p id="mwEnlaces">
<a id="editor" href="editorial.html">Editorial</a>
</p>
```

Luego el nuevo enlace será insertado antes de enlace ya existente llamando el método insertBefore desde el nodo padre (párrafo):

```
var anclaTexto = document.createTextNode("Actualidad");
var nuevoAncla = document.createElement("a");
nuevoAncla.appendChild(anclaTexto);
var anclaExistente = document.getElementById("editor");
var padre = anclaExistente.parentNode;
var nuevoHijo = padre.insertBefore(nuevoAncla, anclaExistente);
```

Si se hiciera una traducción del DOM hacia HTML después de esta operación el resultado sería el siguiente:

```
<p id="mwEnlaces">
<a> Actualidad </a><a id="editor" href="editorial.html">Editorial</a>
</p>
```

En el caso de reemplazar el enlace usando replaceChild:

```
var nuevoHijo = padre.replaceChild(nuevoAncla, anclaExistente);
```

El DOM lucirá así:

```
<p id="mwEnlaces">
<a> Actualidad </a>
</p>
```

Usando innerHTML

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso recurriéndose a la propiedad innerHTML. Dicha propiedad fue introducida por Microsoft permitiendo leer y escribir el contenido HTML de un elemento.

Por ejemplo, puede crearse fácilmente una tabla con múltiples celdas e insertarla luego en la página con innerHTML:

```
var tabla = '<table border="0">';
tabla += '<tr><td>Celda 1</td><td>Celda 2</td><td> Celda 3</td></tr>';
tabla += '</table>';
document.getElementById("datos").innerHTML = tabla;
```

Eliminando un elemento o nodo de texto

Se pueden eliminar nodos existentes y nuevos. El método removeChild permite eliminar nodos hijos a cualquier nodo con tan sólo pasarle las referencias del nodo hijo [a] eliminar y su correspondiente padre. Para mejor compresión retómese el ejemplo

anterior:

```
<p id="mwEnlaces">
<a id="editor" href="editorial.html">Editorial</a>
</p>
```

El método removeChild será usado para eliminar el hipervínculo del elemento padre párrafo:

```
var ancla = document.getElementById("editor");
var padre = ancla.parentNode;
var hijoRemovido = padre.removeChild(ancla);
```

La variable hijoRemovido todavía hace referencia al elemento, de manera que fue removido pero no destruido, no pudiéndose localizar en ninguna parte del DOM. Este se encuentra disponible en memoria como si fuera creado usando el método createElement. Esto permite posicionarlo en cualquier otra parte de la página.

Lectura y escritura de los atributos de un elemento

Las partes más frecuentemente usadas de un elemento HTML son sus atributos, tales como: id, class, href., title, estilos CSS, entre muchas otras piezas de información que pueden se incluidas en una etiqueta HTML.

Los atributos de una etiqueta son traducidos por el navegador en propiedades de un objeto. Dos métodos existen para leer y escribir los atributos de un elemento, getAttribute permite leer el valor de un atributo mientras que setAttribute permite su escritura.

En ocasiones se hace necesario ver las propiedades y métodos de un determinado elemento, esto puede realizarse mediante la siguiente función utilitaria:

```
function inspector(el) {
var str =””;
for (var i in el){
str+=I + “: ” + el.getAttribute(i) + “\n”;
}
alert(str);
}
```

Para usar la función inspector() tan sólo debe pasarle la referencia al elemento, continuando con el ejemplo anterior resulta:

```
var ancla = document.getElementById("editor");

inspector(ancla);
```

Para modificar el atributo title del hipervínculo, elemento referenciado por la variable ancla, se usará el setAttribute, pasándole el nombre del atributo y el valor:

```
var ancla = document.getElementById("editor");
ancla.setAttribute("title", "Artículos de programación");
```

```
var nuevoTitulo = ancla.getAttribute("title");
//El valor de la variable nuevoTitulo es ahora “Artículos de
programación”.
```

Manipulando los estilos de los elementos

Como se ha visto, los atributos que le son asignados a las etiquetas HTML están disponibles como propiedades de sus correspondientes nodos en el DOM. Las propiedades de estilo pueden ser aplicadas a través del DOM.

Cada atributo CSS posee una propiedad del DOM equivalente, formándose con el mismo nombre del atributo CSS pero sin los guiones y llevando la primera letra de las palabras a mayúsculas. Véase el siguiente ejemplo para mayor entendimiento donde se utiliza un atributo CSS modelo:

```
algun-atributo-css
```

Tendrá como equivalente la siguiente propiedad o método en Javascript:

```
algunAtributoCss
```

Por tanto, para cambiar el atributo CSS font-family de un elemento, podría realizarse de lo siguiente:

```
ancla.style.fontFamily = 'sans-serif';
```

Los valores CSS en Javascript serán en su mayoría del tipo cadena; por ejemplo: font-size, pues posee dimensiones tales como “px”, “%”. Sólo los atributos completamente numéricos, tales como z-index serán del tipo entero.

En muchos casos es necesario aparecer y desaparecer un determinado elemento, para ellos se utiliza el atributo CSS display, por ejemplo, para desaparecer:

```
ancla.style.display = 'none';
```

Luego para volverlo a mostrar se le asigna otro valor:

```
ancla.style.display = 'inline';
```

Ejemplo 1: Adjuntar múltiples ficheros a la vez

(Se adjunta con los ejemplos con el nombre de ejemplo1.php)

Este es el primer ejemplo completo donde se propone utilizar la manipulación del DOM mediante javascript con el objetivo de adicionar tantos elementos input del tipo file como tantos ficheros se deseen subir al servidor.

Se muestra una versión simplificada del problema limitada tan sólo al lado del cliente. Para ello es necesario imaginarse un sistema en línea donde se suben ficheros al servidor, ejemplo de ello podría ser una aplicación de correo electrónico.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
    <title>Ejemplo para adjuntar múltiples ficheros</title>
    <script language="javascript" type="text/javascript">
    function nuevoFichero() {
    var input = document.getElementsByTagName("input")[0];
    var nuevoInput = input.cloneNode(true);
    input.parentNode.appendChild(nuevoInput);
    }
    </script>
</head>
<body>
   <form action="upload.php" method="post" enctype="multipart/form-
   data">
   <fieldset><legend>Adjuntar múltiples ficheros</legend>
   <input name="ficheros[]" type="file" size="60" >
   </fieldset>
   <a href="javascript: nuevoFichero();">Adjuntar otro fichero</a>
   <input name="Subir" type="submit" value="Adjuntar" >
   </form>
</body>
</html>
```

El enlace para adjuntar otro fichero hace una llamada a la función nuevoFichero() realizándose las siguientes tareas en la misma:

Se accede al primer elemento input encontrado en el documento:

```
var input = document.getElementsByTagName("input")[0];
```

- Se crea un nuevo elemento input referenciado por la variable nuevoInput utilizando el método cloneNode resultando un elemento idéntico al primero:

```
var nuevoInput = input.cloneNode(true);
```

- Aquí se accede al nodo padre del elemento input mediante el método parentNode y la vez se inserta un nuevo elemento hijo copia del primero por medio del método appendChild:

```
input.parentNode.appendChild(nuevoInput);
```

Es de notar que en este ejemplo podría haberse usado el evento clic para la llamada de la función, si embargo se dejó reservado para cuando el tema de la manipulación de eventos del DOM sea abordado.

Ejemplo 2: DOM y validación

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
<head>
	<title>Ejemplo 2: DOM y validación</title>

	<script type="text/javascript">

	function validar_claves() {

		var c1 = document.getElementById('clave1').value;
		var c2 = document.getElementById('clave2').value;

		if (c1 == "" || c2 == "")
		{
			alert("Debe ingresar la contraseña y repetirla");
			document.getElementById('clave1').focus();
			return false;
		}

		if (c1 != c2)
		{
			alert("Las contraseñas no coinciden");
			document.getElementById('clave1').focus();
			return false;
		}

		if (c1.length < 6)
		{
			alert("La contraseña debe tener al menos 6
caracteres");
			document.getElementById('clave1').focus();
			return false;
		}

		if (document.getElementById('menor18').checked &&
document.getElementById('mayor18').checked){
			alert("Debe seleccionar solo una opcion");
			return false;
		}
			else if
(!(document.getElementById('menor18').checked ||
document.getElementById('mayor18').checked)){
				alert("Debe seleccionar al menos una opcion");
				return false;
		}

		return true;
	}

	</script>

</head>
<body>

<h1>Registro de usuario</h1>

	<form action="noimporta" method="get" onSubmit="return
validar_claves()">

		Ingrese su email:
		<input type="text" name="email" /> <br />

		Ingrese su clave:
		<input type="password" name="clave" id="clave1" /> <br />

		Repita su clave:
		<input type="password" id="clave2" /> <br />

		Edad: <br/>
	  Menor de 18 <input id="menor18" type="checkbox">
	  Mayor de 18 <input id="mayor18" type="checkbox"> <br/><br/>
```

```
            <input type="submit" value="Registrar" />
      </form>
</body>
</html>
```

El formulario tiene asociado el evento onSubmit ante el cual se ejecuta la función de Javascript validar_claves(),

```
<form action="noimporta" method="get" onSubmit="return
validar_claves()">
```

el resultado de la función de validación hará que se produzca el envío o no del formulario, es decir si la función devuelve true, se hará el envío del formulario correctamente, caso contrario, si la función devuelve false, el envío no se hará y se habrán mostrado desde Javascript los popups con los errores correspondientes a través de los alert().

```
alert("La contraseña debe tener al menos 6 caracteres");
```

La línea

```
document.getElementById('clave1').focus();
```

hace que el cursor sea posicionado en el cuadro de texto cuyo id es clave1.

Ejemplo 3: Fechas en Javascript

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
      <title>Ejemplo 3: Fechas en javascript</title>

      <link rel="stylesheet" type="text/css" href="calendar-win2k-
cold-1.css" /> <!-- estilo -->
      <script type="text/javascript" src="calendar.js"></script> <!--
principal -->
      <script type="text/javascript" src="calendar-es.js"></script>
<!-- idioma -->
      <script type="text/javascript" src="calendar-setup.js"></script>
<!-- config -->

      <script type="text/javascript">

      function armar_calendario() {
            Calendar.setup({
            inputField     :    "nacim",    // id del input para fecha
            ifFormat       :    "%d/%m/%Y", // formato
            button          :     "abrircal", // id del botón que abre
calendario
            });
      }
```

```
function validar_fecha() {

      var f=document.getElementById('nacim').value;

      if (f == "") // que la fecha esté
      {
            alert("Por favor ingrese su fecha de nacimiento");
            return false;
      }

      var elementos=f.split("/");
      if (elementos.length != 3) //que tenga dos barras o tres
subgrupos de nros
      {
            alert("La fecha ingresada no es válida.");
            return false;
      }

      if     (elementos[0]==""     ||     elementos[1]==""     ||
elementos[2]=="" ) //que los tres tengan algo
      {
            alert("La fecha ingresada no es válida..");
            return false;
      }

      if    (isNaN(elementos[0])    ||    isNaN(elementos[1])    ||
isNaN(elementos[2]) ) //que sean tres números
      {
            alert("La fecha ingresada no es válida...");
            return false;
      }

      //que sea una fecha de verdad
      var    prueba=new    Date(elementos[2],    elementos[1]-1,
elementos[0]);
      var anio=prueba.getFullYear();
      var mes=prueba.getMonth();
      var dia=prueba.getDate();

      if (anio != elementos[2] || mes != elementos[1]-1 || dia
!= elementos[0])
      {
            alert("La fecha ingresada no es válida....");
            return false;
      }

      //que no sea menor de 18 años
      var fechahoy=new Date();
      var aniohoy=fechahoy.getFullYear();
      var meshoy=fechahoy.getMonth();
      var diahoy=fechahoy.getDate();

      if (prueba > fechahoy){
            alert ("La fecha de nacimiento debe ser menor a la
fecha actual");
            return false;
      }

      if(mes > meshoy){
            var edad = aniohoy - anio - 1;
      }
      else{
            if(mes == meshoy && dia>diahoy){
                  var edad = aniohoy - anio - 1;
            }
            else{
                  var edad = aniohoy - anio;
            }
      }
```

```
            if (edad < 18){
                  alert ("Debe ser mayor de 18 años");
                  return false;
            }

            return true;
      }

      </script>

</head>
<body onLoad="armar_calendario()">

<h1>Fechas en javascript</h1>

      <form    action="noimporta"    method="get"    onSubmit="return
validar_fecha()">

            Ingrese su nombre: <input type="text" name="nombre" /> <br
/>
            Ingrese  su  nacimiento:  <input  type="text"  name="nacim"
id="nacim" />
                  <input  type="button"  id="abrircal"  value="..."  />
<br />

            <input type="submit" />

      </form>

</body>
</html>
```

Este ejemplo trabaja de la misma manera que el anterior en cuanto a la validación de los datos en la etiqueta form se le agrega el evento onSubmit al cual se le asocia la función de Javascript validar_fecha().

Se utiliza el método split() que divide un string en un array de substrings y retorna dicho array es utilizado para separar el día, mes y año

```
var elementos=f.split("/");
```

La función isNaN(*value*) (is Not a Number) devuelve true si el parámetro pasado no es un número y false si lo es.

```
(isNaN(elementos[0]) || isNaN(elementos[1]) || isNaN(elementos[2]) )
//que sean tres números
```

El objeto Date calcula la fecha del día si no recibe parámetros:

```
var fechahoy=new Date();
```

Luego se utilizan los métodos getFullYear(), getMonth() y getDate() para obtener el año en 4 dígitos, mes y día

```
var aniohoy=fechahoy.getFullYear();
var meshoy=fechahoy.getMonth();
var diahoy=fechahoy.getDate();
```

El objeto Date siempre devuelve una fecha a partir de los parámetros recibidos aunque no sean válidos, por ejemplo si le pasáramos Date(2013,10,39) no serían datos válidos

ya que no existe el día 39, pero Date trata de convertir esos parámetros en una fecha válida y devuelve esa fecha, es por eso que para validar si la fecha es válida se hace a través del if:

```
if (anio != elementos[2] || mes != elementos[1]-1 || dia != elementos[0])
```

Una característica del objeto Date es que trabaja con los meses de 0 a 11 es por eso que al llamarla con parámetros se le resta 1 al mes:

```
var prueba=new Date(elementos[2], elementos[1]-1, elementos[0]);
```

Lo mismo se hace en la comparación:

```
if (anio != elementos[2] || mes != elementos[1]-1 || dia != elementos[0])
```

Ejemplo 4: Manipulación del DOM

El objetivo del siguiente ejemplo es ver como podemos agregar elementos HTML dinámicamente creando elementos y sus nodos de texto y como podemos modificar o agregar atributos de CSS desde Javascript .

En el cuadro de texto ingresamos los nombre de una lista de invitados, los cuales se van mostrando a continuación, a través del agregado de un div por cada invitado ingresado, dentro del div id=”lista” y se agrega un campo de input type=”hidden” dentro del formulario que será el encargado de enviar la lista de invitados al archivo correspondiente que los procese.

Dentro del div id=”lista” se agregará un div con un elemento [a] (vínculo para poder eliminar ese invitado del alista) por cada invitado agregado.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>

<title>Ejemplo 3: Manipulación del DOM</title>

      <script type="text/javascript">

            var contador=0;
            var totalinvitados=0;
            function agregar() {
                  var inv=document.getElementById('ingreso'); //referencia al input

                  if (inv.value=="") return;

                  var nuevo = document.createElement("div"); //fila para agregar
                  nuevo.setAttribute("id", "invdiv"+contador); //id para borrar --- LUEGO
                  nuevo.style.fontFamily='sans-serif';
                  nuevo.style.fontSize=22;
```

```
                nuevo.style.backgroundColor='yellow';
//              nuevo.style.border='dotted';
//              nuevo.style.display='none';
//              nuevo.style.display='inline';


                var txtnuevo= document.createTextNode( inv.value + '
'); //texto para la fila
                nuevo.appendChild(txtnuevo); //meto texto

                // ---------------- PARA ELIMINAR
                var   borrarnuevo   =   document.createElement("a");
//link para borrar
                borrarnuevo.setAttribute("href",   "#");   //id   para
borrar LUEGO
                borrarnuevo.setAttribute("onClick",        "borrar("+
contador +")"); //id para borrar LUEGO
                borrarnuevo.style.fontFamily='sans-serif';
                borrarnuevo.style.fontSize=15;
                borrarnuevo.style.fontStyle='italic';


                var     txtborrarnuevo=     document.createTextNode(
"borrar" ); //texto para el link borrar
                borrarnuevo.appendChild(txtborrarnuevo);

                nuevo.appendChild(borrarnuevo);   //meto   link   para
borrar
                // ----------------

                var           caja=document.getElementById('lista');
//referencia a la div
                caja.appendChild(nuevo); //meto fila

                var       nuevoinput=document.createElement("input");
//para el form oculto
                nuevoinput.setAttribute("type","hidden");   //ponerle
HIDDEN, TEXT es para verlos
                nuevoinput.setAttribute("name","inv[]");
                nuevoinput.setAttribute("value",inv.value);
                nuevoinput.setAttribute("id","invinput"+contador);

                var
formoculto=document.getElementById('formoculto'); //referencia al form
                formoculto.appendChild(nuevoinput); //meto fila

                //listo! agregado
                contador++;
                totalinvitados++;

                inv.value=""; //reseteo input
                inv.focus();
          }

          function borrar(id) {

                //borro de la lista, la fila
                var                    divborrar                    =
document.getElementById('invdiv'+id);
                var padre = divborrar.parentNode;
                padre.removeChild(divborrar);

                //borro del form, el input
                var                  inputborrar                    =
document.getElementById('invinput'+id);
                var padre = inputborrar.parentNode;
                padre.removeChild(inputborrar);

                totalinvitados--;
          }
```

```
            function validar(){
                  if (totalinvitados == 0){
                        alert("Debe haber alguna persona invitada");
                        return false;
                  }
                  return true;
            }

      </script>

</head>
<body>

      <h1>Lista de invitados</h1>

      Ingrese nombre del invitado: <br />
      <input   type="text"   id="ingreso"   />   <input   type="button"
value="Agregar a la lista" onClick="agregar()" />

            <hr />

      <div id="lista">
      <h3> Lista de invitados </h3>
      </div>

            <hr />

      <form     action="noimporta"     method="get"     id="formoculto"
onSubmit="return validar()">

      <input type="submit" value="Enviar lista" />
      </form>

</body>
</html>
```

Conclusión

El presente material tan sólo ha sido una ligera aproximación a la manipulación del Modelo de Objetos del Documento. Explorando el DOM se puede encontrar, cambiar, adicionar y eliminar elementos de un documento. Es sin duda alguna una poderosa técnica para generar aplicaciones Web dinámicas en el lado del cliente.