INSTITUTO UNIVERSITARIO AERONÁUTICO
Ingeniería Web II
Javascript - Usando el DOM - Manipulando el HTML para crear páginas dinámicas
Docente: Ing. Carlos Simes

Una de las características más interesantes y poderosas del Lenguaje Javascript es su capacidad de acceder a los elementos que conforman una página web.
Con javascript se puede acceder a los elementos HTML fundamentalmente para:

- Leer sus propiedades
- Modificar sus propiedades
- Leer sus estilos
- Modificar sus estilos
- Crear nuevos elementos dentro de otros
- Eliminar elementos ya existentes
- Responder mediante acciones (funciones) a los eventos que el usuario efectúa sobre los elementos.

Todo esto permite el procesamiento y modificación dinámica de una página, lo que aprovechando las posibilidades de programación del lenguaje serán útiles cuando queramos dotar a nuestro sitio de funcionalidades que no necesariamente sean implementadas en el servidor.

De esta manera, nuestra app se aproximará a una aplicación de escritorio embebida dentro de un navegador, mejorando la performance, el tiempo de respuesta al usuario y potenciando las posibilidades de procesamiento al no recargar al servidor web con funciones y acciones que puede resolver directamente el cliente de manera local.

Pensemos por ejemplo si quisiéramos que el usuario de nuestra aplicación se registrase en un formulario donde debe, por ejemplo, colocar su correo electrónico el cual se guardará en una base de datos.
Si no contásemos con javascript y fuese necesario asegurarse que el correo a registrar sea válido, al menos desde el punto de vista del formato, cada vez que el usuario introduzca su correo y oprima el botón Guardar, deberíamos enviar ese correo al servidor web, analizar la estructura del valor introducido para verificar que cuenta al menos con un símbolo @ y con el menos un punto.

Si el valor es erróneo debería el servidor enviar esa información al usuario pidiéndole que corrija lo que está mal. Imaginemos un sitio con miles de usuarios y pensemos en que además haya que controlar datos como la edad, que no puede ser negativa, el DNI que debe tener al menos 7 caracteres separados por . cada 3 cifras y así con cada valor por cada usuario. Si todas esas funciones las debe hacer el servidor nuestra app se volvería sumamente lenta.
Si además quisiéramos que cada vez que se introduce un valor erróneo, la casilla con el valor incorrecto cambiase de color indicando el error, el servidor debería modificar los estilos y reenviar la página con el nuevo estilo para cada error que se

produzca. Esto multiplicado por cada usuario. Para todo ello y muchas otras cosas más es que se utiliza Javascript.

A lo largo de su evolución este lenguaje de script, ha incorporado nuevas y poderosas funciones, al punto tal de volverse un lenguaje con un nivel de complejidad que llevó al desarrollo de varios frameworks de trabajo basados en este lenguaje como Angular, React, Vue, Ember, Polymer, etc.

Accediendo a los elementos de la página. EL DOM

Para acceder a los distintos elementos HTML, debemos conocer como está estructurada la misma. Toda página compuesta por distintos elementos html ubica estos dentro de una estructura jerárquica en función de como hemos ido agregando estos mientras diseñábamos la página.

Esta estructura jerarquica de elementos HTML se denomina DOM que es el acrónimo de Document Objects Model o Modelo de Objetos de Documentos.

En principio, cuando tenemos una página en blanco, el documento base está "vacío" y solo contiene el **objeto document.**

document es la raíz de la estructura del DOM

Representa básicamente el área visible de nuestra página en la que incorporaremos luego otros elementos. Algo por el estilo ya habíamos usado antes al hacer **document.write** Esta función nos permite escribir el cuerpo de nuestra página con texto plano y/o con etiquetas html, reemplazando el contenido existente por lo que coloquemos dentro de write.

Por ejemplo si tenemos en nuestra página un Hola a Todos y queremos reemplazar ese párrafo por Bienvenido a nuestro sitio
en una función de javascript deberíamos colocar

document.write(" Bienvenido a nuestro sitio </P>").

Esto elimina el contenido de la página reemplazándolo por el nuevo contenido.

Esto nos será útil para cuestiones muy simples, pero que pasaría si en la página tengo un conjunto de elementos que no quiero que se pierdan, o solo quiero modificar el contenido de algunos elementos en particular sin alterar el resto de la página ?

Podríamos imaginar un mecanismo en el que memorizamos todo el contenido de la página, luego lo procesamos para encontrar el elemento que queremos cambiar, modificamos al valor deseado y reescribimos la página usando document.write

Si bien desde el punto de vista técnico la solución es viable imaginen una página compleja con cientos de controles, imágenes, tablas, etc en donde debiéramos

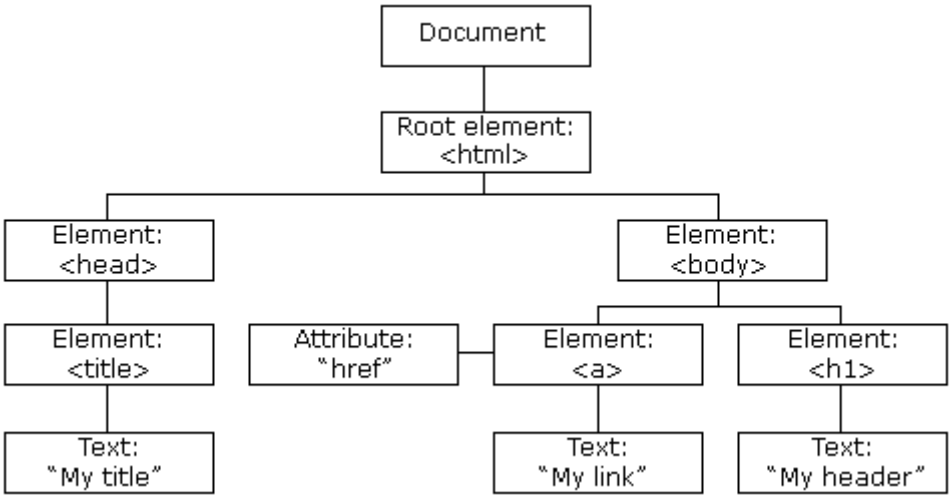
procesar todo el contenido solo para encontrar un párrafo en particular, modificarlo y luego reconstruir la página con decenas de **document.write** !

Algo por el estilo ya hicieron en las actividades anteriores y seguro renegaron bastante. Bueno por suerte para todos está el DOM y las funciones de manipulación del mismo.

Resumiendo lo que hemos explicado hasta aquí tendremos:

En DOM podemos navegar por los nodos gracias a JavaScript.
Con él podemos **crear nuevos Nodos, eliminarlos o modificarlos**, para ello debemos tener claro la relación entre nodos:

- Cada elemento del Html es un nodo
- Los textos existentes en los elementos Html son nodos texto
- Cada atributo que existe en Html es un nodo atributo
- Todos los comentarios son nodos comentarios



Los nodos en los árboles de nodos tienen una relación jerárquica entre sí. Esta relación es entre padre, hijos o hermanos:

- En nuestro árbol de nodos siempre tenemos un nodo padre, que también es llamado nodo raíz.
- Todos los nodos tienen un padre, a excepción del nodo raíz.
- Un nodo puede tener una cantidad de hijos
- Son nodos hermanos aquellos nodos que tienen el mismo padre

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 desearamos encontrar de manera rápida y fácil un elemento usaremos la función **getElementById**.

Esta nos permite un acceso inmediato a cualquier elemento de la página tan sólo conociendo el valor de su **atributo id**.
Como el atributo id es único para cada elemento de la página y lo define el diseñador de la misma, podemos asignarle a cada elemento un id y luego usarlo para acceder al mismo.

Ejemplo:

```
<p>
<a id="contacto" href="https://www.yahoo.com.ar">Navegar a Yahoo</a>
</p>
```

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

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

Ahora el valor de la variable elementoAncla es una referencia al elemento [a] y cualquier operación sobre la misma afectará el hiperenlace.
Por ejemplo podríamos querer cambiar el valor del atributo href para modificar la navegación. Entonces haríamos:

```
function cambiarNavegador()
{
var elementoAncla = document.getElementById("contacto");
elementoAncla.href="https://www.google.com.ar";
}
```

En las actividades más adelante hay ejemplos completos de esta función.

El método **getElementById** es adecuado para operar sobre un elemento en específico, sin embargo, en ocasiones se necesita trabajar sobre un grupo de elementos.
En este caso puede utilizarse la función **getElementsByTagName**.
Esta retorna **un array** de 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>
```


Pueden obtenerse todos los hipervínculos de la siguiente manera:

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

El valor de la variable **hipervinculos** es una matriz de elementos [a], pudiéndose acceder a cada elemento a través de la ya conocida notación con corchetes.

Los elementos devueltos por **getElementsByTagName** serán ordenados según el orden que aparezcan en la página. 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"

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 directamente a través de **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.**

Por ejemplo:

Supongamos una página que tiene un formulario que contiene dos input de tipo texto en el que introduciremos dos números. Además habrá un botón al cual le haremos click para calcular la suma de ambos números.

Nuestra página se vería más o menos así:

```
<!DOCTYPE html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Suma de dos números</title>
</head>

<script>
      function Sumar()
      {
            var n1=parseInt(formulario.num1.value);
            var n2=parseInt(formulario.num2.value);
            var resultado=n1 + n2;
            alert("El resultado de la Suma es: " + resultado);
      }
```

```
</script>

<body>
<!-- Formulario -->
<form name="formulario">
Primer  Número   <input type="number" name="num1"/><br>
Segundo Número   <input type="number" name="num2"/><br>

<input type="button" value="Sumar" onclick="Sumar()"/><br>
</form>
<!-- /Formulario -->

</body>
</html>
```

En este ejemplo como nuestro formulario tiene un nombre y los elementos contenidos en el también, podemos acceder a ellos usando la notación de **.** de elementos jerárquicos.
formulario.num1.value permite acceder al valor del elemento input que se llama **num1** que está dentro del elemento **formulario.**
En una página simple, es sencillo usar esta forma. En una página compleja el anidamiento puede ser muy grande y el código volverse confuso.

Ahora veamos este mismo ejemplo usando **getElementById**
Nos quedaría:

Ej002.html

```
<!DOCTYPE html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Suma de dos números</title>
</head>

<script>
function Sumar(){
var n1=parseInt(document.getElementById("num1").value);
var n2=parseInt(document.getElementById("num2").value);
var resultado=n1 + n2;
alert("El resultado de la Suma es: " + resultado);
}
</script>

<body>
<!-- Formulario -->
<form name="formulario">
Primer  Número   <input type="number" id="num1" /><br>
Segundo Número   <input type="number" id="num2" /><br>
```

```
<input type="button" value="Sumar" onclick="Sumar()"/><br>
</form>
<!-- /Formulario -->

</body>
</html>
```

Actualmente la mayoría de los navegadores soportan estos métodos, aún así es recomendable el uso de los métodos:

getElementById y getElementsByTagName, pues hacen más claro el código e independientemente del nivel de anidamiento proveen acceso directo a los elementos de DOM.

Usando innerHTML

En aplicaciones complejas donde es necesario crear varios elementos a la vez, el código JavaScript generado puede ser extenso y dificultoso de mantener. Cuando ocurre esto podemos recurrir 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**:
Esta propiedad permite leer o escribir el contenido de un elemento html que puede obrar como contenedor de otros, por ejemplo una etiqueta

Supongamos que queremos resolver usando esta propiedad y lo aprendido hasta el momento el ejercicio de la tabla de multiplicar. Nuestra página html nos quedaría así:

Ej003.html

```
<!DOCTYPE html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

       <title>Suma de dos números</title>
</head>

<script>
function GenerarTabla(){
var n1=parseInt(document.getElementById("num1").value);
var salida="<table border=1 width='60%' align='center'>";
for(i=0;i<13;i++){
salida=salida +  "<tr><td>" + n1 + "</td><td> x </td><td>"  + i + "</td><td>
= </td><td>" + i*n1 + "</td><tr>";
```

```
}
salida=salida + "</table>";
document.getElementById("tabla").innerHTML=salida ;
}
</script>

<body>
<!-- Formulario -->
<h1> Tabla de Multiplicar </h1>
<p>Ingrese el número cuya tabla de multiplicar desea obtener</p>
Número   <input type="number" id="num1" />
<br>
<br>

<input type="button" value="Generar Tabla"
onclick="GenerarTabla()"/><br>

<!-- Div para mostrar la tabla de resultados -->

<div id="tabla" width="600px">
</div>

</body>
</html>
```

En este ejemplo hemos usado innerHTML para reemplazar el contenido del elemento con la salida que queremos mostrar, que en nuestro caso es la tabla de multiplicar generada por código.

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 ser 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.
Usando estos métodos podemos leer y escribir las propiedades.

Ej004.html

```
<!DOCTYPE html>
<head>
   <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

```
        <title>Cambiar Atributos</title>
</head>


<script>
function CambiarColor(){

objeto=document.getElementById("parrafo1").getAttribute("id");
var style= document.getElementById("parrafo1").getAttribute("style") ;

alert("El style del objeto cuyo id es: " + objeto + " es " + style);
nuevocolor="background-color: powderblue"; // cambiamos el color
document.getElementById("parrafo1").setAttribute("style",nuevocolor);
//document.getElementById("parrafo1").color="red";

}
</script>
<body>
<h1> Ejemplo de getAttribute y setAttribute  que usaremos para cambiar un estilo
</h1>

<p id="parrafo1" style="background-color: red"> A este párrafo le cambiaremos su
color </p>
<br>
<br>

<input type="button" value="Cambiar Color"
onclick="CambiarColor()"/><br>

</body>
</html>
```

Manipulando los estilos de los elementos

Hay una forma más sencilla para modificar los estilos de los elementos html. 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:
Supongamos un elemento llamado ancla
ancla.style.fontFamily = 'sans-serif'; // cambiamos la fuente del enlace

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';

Ahora veremos algunos ejemplos más mostrando lo explicado hasta ahora y luego les propongo que realicen unas actividades al final de este documento.

Ejemplo Nº1:

Características básicas de los nodos DOM. Acceso usando **getElementById**

```
<html>
<head>
<title>Identificando elementos </title>
</head>
<body>
<h3 id="intro" class="titulo">Introducción</h3><br>
<input type="text" class="miclase" value="Nombre" id="texto"
name="Nombre"><br>
<input type="button" id="Boton" value="Mostrar" onclick="Mostrar();">
<script>
var nodo=document.getElementById( "intro" )
alert( "PROPIEDADES del nodo intro:\n\n"+
      "Tipo: "+nodo.nodeType+"\n"+
      "Nombre: "+nodo.nodeName.toLowerCase()+"\n"+
      "Clase: "+nodo.className+"\n"+
      "Valor: "+nodo.nodeValue )
var nodo=document.getElementById( "texto" )
alert( "PROPIEDADES del nodo intro:\n\n"+
      "Tipo: "+nodo.nodeType+"\n"+
      "Nombre: "+nodo.nodeName.toLowerCase()+"\n"+
      "Clase: "+nodo.className+"\n"+
      "Valor: "+nodo.nodeValue )
```

```
function Mostrar()
{
alert(document.getElementById( "texto" ).value)
}
</script>
</body>
</html>
```

Ejemplo Nº2: Capturando el Contenido de un componente

```
<html>
<head><title>Ejemplo de uso del getElementById</title></head>
<body>
<form action="">
  <textarea rows="20" cols="40" id="areadetexto">Hola a Todos como están ? Yo soy el Contenido del TextArea !!! </textarea>
  <input type="button" value="Contenido del TextArea" onClick="
alert(document.getElementById('areadetexto').value);">
 </form>
</body>
</html>
```

Ejemplo Nº3: Usando InnerHTML y setTimeout

Nota: La función **setTimeout** al igual que **setInterval** permiten programar eventos automáticos en base a intervalos en milisegundos predefinidos. Puede encontrar más información en la web.

```
<html>
<head>
<title>Accediendo a la propiedad innerHTML</title>
</head>
<body>
<div id="parrafillo">
Este es un párrafo que contiene muchas cosas:
<ul>
  <li>Una</li>
  <li>lista</li>
</ul>
<table border=1><tr><td>Y una</td><td>tabla!</td></tr></table>
</div>
<script>
function cambiar() {
  var el=document.getElementById( "parrafillo");
  el.innerHTML="Y lo cambio completamente"+
    "<ol><li>aunque</li><li>también incluyo</li><li>una lista</li></ol>";
}
setTimeout( "cambiar()", 5000 );
// después de 5 segundos se ejecuta la función cambiar.
```

```
</script>
</body>
</html>
```

Ejemplo Nº4: Selección de Múltiples elementos simultáneamente

```
<html>
<head>
<title>Trabajando con formularios </title>
</head>
<body>
Modificamos checkbox automáticamente
<form name="n1" id="i1" action="http://www.google.com.ar/search">
  <input type="checkbox" name="seleccionar">Seleccionar todos<br/><br/>
  <input type="checkbox" name="linea">Línea 1<br/>
  <input type="checkbox" name="linea">Línea 2<br/>
  <input type="checkbox" name="linea">Línea 3<br/>
  <input type="checkbox" name="linea">Línea 4<br/>
  <input type="checkbox" name="linea">Línea 5<br/>
</form>
<script>
el=document.getElementById( "i1" )
el.seleccionar.onclick=function cambia() {
  for( var i=0; i<el.linea.length; ++i ) {
    el.linea[i].checked=el.seleccionar.checked;
  }
}
</script>
</body>
</html>
```

Ejemplo Nº5: Usando Propiedades de String

```
<html>
<head>
<title>Trabajando con formularios</title>
</head>
<body>
Pasamos texto a mayúsculas
<form name="n1" id="B1" action="http://www.google.com.ar/search">
  Nombre: <input type="text" name="nombre" size="30"><br/>
  <input type="submit" value="Enviar">
</form>

<script>
el=document.getElementById( "B1" )
el.nombre.onblur=function cambia() {
  el.nombre.value=el.nombre.value.toUpperCase();
```

```
}
</script>
</html>
```

Ejemplo Nº6: Propiedades y Nodos – Navegación

```
<html>
<head>
<title>Navegando por los elementos del documento </title>
</head>
<body>
<ul>
  <li><a href="uno.html">El primer enlace</a></li>
  <li class="miClase"><a href="dos.html" target="new">El segundo
enlace</a></li>
  <li><a href="tres.html">El tercer enlace</a></li>
</ul>
<script>
var nodo=document.getElementsByTagName( "li" )[1]
document.write( "El propio nodo: ", nodo.nodeName, "<br/>" )
document.write( "Su hermano anterior: ", nodo.previousSibling.nodeName,
"<br/>" )
document.write( "El hermano anterior a ese: ",
nodo.previousSibling.previousSibling.nodeName, "<br/>" )
document.write( "Su hermano posterior: ", nodo.nextSibling.nodeName, "<br/>" )
document.write( "El padre del nodo: ", nodo.parentNode.nodeName, "<br/>" )
document.write( "¿Tiene nodos hijo?: ", nodo.hasChildNodes() , "<br/>" )
document.write( "Número de hijos: ", nodo.childNodes.length, "<br/>" )
document.write( "Primer hijo: ", nodo.firstChild.nodeName, "<br/>" )
document.write( "Último hijo: ", nodo.lastChild.nodeName, "<br/>" )
</script>
</body>
</html>
```

Ejemplo Nº7: JavaScript y Estilos

```
<html>
<head>
<title>Modificando todos los enlaces de un documento </title>
</head>
<body>
<ul>
  <li><a href="uno.html">El primer enlace</a></li>
  <li class="miClase"><a href="dos.html" target="new">El segundo
enlace</a></li>
  <li><a href="tres.html">El tercer enlace</a></li>
</ul>
<script>
var nodos=document.getElementsByTagName( "a" )
```

```
for ( var i=0;  i<nodos.length; ++i ) {
  nodos[i].onmouseover=function() {
    this.style.backgroundColor='yellow'
  }
  nodos[i].onmouseout=function() {
    this.style.backgroundColor=''
  }
}
</script>
</body>
</html>
```

Ejemplo Nº8: Para esta actividad será necesario contar con 3 imágenes llamadas Notebook1.jpg, Notebook2.jpg y Notebook3.jpg

Se muestra como generar un catálogo móvil para mostrar varias imágenes en el mismo lugar.

```
<html><head><title>Trabajando con Imágenes y GetElementById </title> <meta
charset="utf-8">
<style type="text/css">
body {text-align: center; font-family: sans-serif;}
div {margin:20px;}
#contenedor {width:405px;margin:auto;}
#adelante, #atras {padding:15px; width: 130px; float: left;
color: white; border-radius: 40px; background: rgb(202, 60, 60);}
#adelante:hover, #atras:hover {background: rgb(66, 184, 221);}
</style>
<script type="text/javascript">
var numeroImagenActual =1;
function moverImagen(movimiento) {
if (numeroImagenActual == 1 && movimiento == 'atras' || numeroImagenActual
== 3 && movimiento == 'adelante') {
alert ('No es posible hacer ese movimiento');
}
if (numeroImagenActual == 3 && movimiento == 'atras' || numeroImagenActual
== 1 && movimiento =='adelante') {
valorNuevoNumeroImagen = 2;
document.getElementById('imgCarrusel').src = 'Notebook2.jpg';
document.getElementById('imgCarrusel').alt = 'Notebook2';
document.getElementById('imgCarrusel').title = 'Notebook2';
}
if (numeroImagenActual == 2 && movimiento == 'atras') {
valorNuevoNumeroImagen = 1;
document.getElementById('imgCarrusel').src = 'Notebook1.jpg';
document.getElementById('imgCarrusel').alt = 'Notebook1';
document.getElementById('imgCarrusel').title = 'Notebook1';
}
if (numeroImagenActual == 2 && movimiento == 'adelante') {
```

```
valorNuevoNumeroImagen = 3;
document.getElementById('imgCarrusel').src = 'Notebook3.jpg';
document.getElementById('imgCarrusel').alt = 'Notebook3';
document.getElementById('imgCarrusel').title = 'Notebook3';
}
numeroImagenActual = valorNuevoNumeroImagen;
document.getElementById('numeracion').firstChild.nodeValue = 'Notebook ' +
numeroImagenActual;
}
</script>
</head>
<body>
<div >
<p>Pulsa adelante o atras</p>
<h1 id="numeracion">Notebook 1</h1>
<img id="imgCarrusel" src="Notebook1.jpg" alt="Notebook1" title="Notebook1">
<div id="contenedor">
<div id ="atras" onclick="moverImagen('atras')"> <<< Atras </div>
<div id="adelante" onclick="moverImagen('adelante')" >Adelante >>></div>
</div>
</div></body></html>
```

<u>Actividades a desarrollar por el alumno:</u>

Actividad Nº 1:

Desarrolle un Banner o Marquesina que permita mostrar tres párrafos distintos que deberán mostrarse automáticamente cada 5 segundos uno a continuación del otro. Aplique estilos diferentes para cada uno de los párrafos.
Párrafos:
1) Bienvenidos a Nuestra Web – Los mejores productos de Electrónica
2) ABC – Notebooks – Las Mejores Marcas y el Mejor Precio
3) Servicio Técnico Especializado y Garantizado

Actividad Nº 2:

Desarrolle una página web donde el usuario ingresará un número y al oprimir Mostrar, se generará la tabla de multiplicar del 1 a 10 del número ingresado.

Actividad Nº 3:

Desarrolle una página Web donde el usuario ingresará en un textarea un párrafo y un botón Corregir. Al oprimirlo, se mostrarán en rojo las palabras que tienen errores de ortografía. Use como comparación un diccionario base de palabras al que puedan ir agregándosele palabras.

Actividad Nº 4:

Modifique la actividad que le permitía la carga de los alumnos y sus promedios. Agregue un botón buscar y una casilla de texto donde incluir el texto a buscar. a medida que se vaya incluyendo el texto, se deberán ir mostrando los registros que contengan el texto introducido en alguno de sus campos.

Actividad Nº 5:

Desarrolle un Formulario de Registro para la aplicación web de venta de notebook. Genere una función javascript que analice los datos ingresados y valide los mismos. Los campos faltantes o incorrectos deberán colorearse para indicar al usuario su ingreso erróneo.

El Formulario deberá contener al menos los siguientes campos:
Apellido, Nombres, DNI, Cuit, Sexo, Calle, Número, Barrio, Ciudad, Provincia, Teléfono, Correo Electrónico, siendo todos obligatorios.