Programación Front End (nivel inicial)

Ing. Leonardo Torres, Lic. Gustavo Lafuente

FACUTLAD DE INGENIERÍA
Universidad Nacional de la Pampa

Argentina programa 4.0

Lenguaje **Javascript**

Programación Orientada a la Web

Lenguaje Javascript

Javascript

- Un lenguaje compacto de *"scripting"* basado en objetos para desarrollar aplicaciones Internet Cliente – Servidor

- Es embebido en un documento HTML
  - ❖ Se coloca el *head* como en el *body*

- Puede reconocer y responder a eventos de usuario sin ninguna conexión a la red

  Ej. Chequeos de validación

Lenguaje Javascript

Fue desarrollado originalmente por Brendan Eich para utilizarse en Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a LiveScript, para finalmente quedar como **JavaScript**.

En el transcurso de los años tuvo muchos versionados, desde el **Javascript 1.0** hasta el actual de junio de 2018, el estándar **ECMAScript 2021** (1 junio de 2021)

Características

- Es interpretado por un intérprete incorporado en el navegador que posibilita la ejecución de código generado dinámicamente.
- La sintaxis es muy similar a Java o C++, permitiendo una fácil adaptación. (como diferencia este puede o no terminar sus sentencias con ;)
- Es un lenguaje basado en objetos que no implementa ni el concepto de clase ni el mecanismo de herencia.
- Utiliza una jerarquía de objetos encabezada por el propio navegador (DOM)
- Proporciona un conjunto de objetos predefinidos de mucha utilidad
- Permite la captura y tratamiento de una serie de eventos provocados tanto por el usuario como por el navegador
- Sirve para incorporar otros elementos tecnológicos como ActiveX, XML, DHTML (es la base de AJAX)

Ejecución Javascript

Del lado del Cliente	Del lado del Servidor
A través de los clientes Web del usuario, es compatible con la mayoría de los navegadores del mercado. Se tornó mas atractivo su uso en los dispositivos móviles.	En su inicios solo era ejecutado por el servidor Netscape Fiveware pero desde la liberación del código en el año 2000 han surgido otros servidores como **Node.Js**, **Jaxer, EJScript**, **RingoJS**, **AppengineJS** entre otros.

JS – del Lado del Cliente

- Incluido en el doc. HTML y descarga con este:
  - Funciones, variables con los tags <script>...</script>
  - Atributos para manejar eventos para los tags comunes del HTML
- Interpretado por la mayoría de los Navegadores.
  - funciones definidas cuando la página es cargada
  - funciones invocadas sobre eventos accionados(ej., click mouse)
- Acceso a objetos y operaciones
  - windows (abrir una nueva ventana y cargar un URL) frames (cargar un frame desde otro)
  - elementos de un form (invocar a una función cuando se cambia el valor o cuando se invoca a submit)
  - links (cambios hrefs)
- Puede también crear HTML Dinámico

JS – del Lado del Servidor

- Incluido en el documento HTML "original“
  - funciones, variables con los tags <script>...</script>
  - Código para HTML Dinámico con el HTML estático
- Compilado y Ejecutado por Netscape LiveWire , NodeJS entre otros
- Acceso a objetos y operaciones
  - Estado del Servidor
    - ✓ *Estado por usuario*
  - Requerimientos de HTTP concurrente (Chequeos de usuarios y nombres de dominio)
  - Accesos a Bases de Datos (invocando a consultas en SQL)
- Puede ser usado con cualquier navegador

JS –En el Cliente embebido


```
<HTML>
 <HEAD>
        <SCRIPT language="JavaScript">
            <!--Código oculto para los navegadores
            alert("Hola JavaScript");
            -->
        </SCRIPT>
 </HEAD>
 <BODY>
    <P>Esto es un simple alerta JavaScript
 </BODY>
</HTML>
```

Nota: el código <!-- ..... --> es interpretado por Internet Explorer.

Ref. js00.html

JS –En el Cliente Enlazado


```
<HTML>
 <HEAD>
        <SCRIPT Language="JavaScript"    SRC="codigo.js">
        </SCRIPT>
 </HEAD>
 <BODY>

 </BODY>
</HTML>
```

JS –En el HEAD o en BODY


```
<HTML>
        <HEAD>
                <SCRIPT
Language="JavaScript"
        SRC="codigo.js">
                </SCRIPT>
        </HEAD>
        <BODY>



        </BODY>
</HTML>
```

```
<HTML>
        <HEAD>
        </HEAD>
        <BODY>



                <SCRIPT
Language="JavaScript"
        SRC="codigo.js">
                </SCRIPT>
        </BODY>
</HTML>
```

Colocar los Scripts en la parte inferior del elemento <body> mejora la velocidad de visualización, porque la interpretación de los Scripts relentiza la visualización

Javascript vs Java

JAVA	JAVASCRIPT
Requiere Compilación	No requiere compilación, es interpretado por los navegadores del lado del cliente.
Es un lenguaje que se puede considerar pesado, potente y robusto en el sentido de que permite hacer de todo con un gran control.	Es un lenguaje que se puede considerar ligero, ágil y poco robusto en el sentido de que no permite hacer todo lo que permiten otros lenguajes.
Es un lenguaje bajo el paradigma de orientación a objetos completamente.	Es un lenguaje no clasificable bajo un paradigma concreto y admite algunas formas de programación no admitidas por Java.
Requiere de un kit de desarrollo y máquina virtual Java para poder programar en él.	No requiere nada específico para poder programar en él (únicamente un navegador web para ver los resultados y un editor de texto para escribir el código).
Es Fuertemente tipado	Es débilmente tipado y las variables pueden cambiar de tipo en la ejecución

Javascript vs Java (cont)

JAVA	JAVASCRIPT
Clases y las instancias son entidades distintas	Todos los objetos son instancias
Define una clase con una definición de clase; inicializa la clase con un método constructor	Define y crea un conjunto de objetos con funciones constructoras
Crea un simple objeto con el operador new	igual
Construye una jerarquía de objetos usando definiciones de clases para definir subclases de clases existentes	Construye una jerarquía de objetos asignando un objeto como el prototipo asociado con una función constructor
Hereda propiedades por la cadena de clases	Hereda propiedades de la cadena de prototipos html
La definición de la clase especifica todas las propiedades de todas las instancias de una clase. No permite agregar propiedades dinámicamente en tiempo de ejecución	El constructor de función o prototipo especifica un conjunto inicial de propiedades. Puede agregar o remover propiedades dinámicamente para objetos individuales o para el conjunto entero de objetos

JS –Contenido del Script

- Comentarios
  - Similares a C++

    //Esta es una línea de comentario

    */*Este es un comentario multilinea*/*
- Código
  - Serie de sentencias separadas por punto y coma (;). Es opcional el punto y coma pero clarifica la separación las instrucciones

JS –Contenido - Impresión


```
<SCRIPT>
    document.write("Hola Mundo"+ "<BR>");
</SCRIPT>
```

- El texto puede ir con comillas simples o dobles
- Incorporación de Tags embebidos en el texto, interpretados por el Navegador

Ref. js01.html

JS –Impresión

- *Alert*
  - Permite mostrar una ventana de mensajes sobre la pantalla

```
<SCRIPT>
        alert("Hola JavaScript");
</SCRIPT>
```

- *Document.write*
  - El método write permite escribir sobre la pantalla

```
<SCRIPT>
        document.write("Hola JavaScript");
</SCRIPT>
```

Nota: **document** se refiere al documento objeto, y **write** es un método que puede ser usado con el objeto document.

JS –Escribiendo código html

- **Document.write** es analizado por el navegador, éste puede analizar tags html.

```
<SCRIPT>
    document.write("<B>Hola JavaScript</B>");
</SCRIPT>
```

- Caracteres de escape (&*;)
  - Para que aparezcan los tags sobre la pantalla

```
<SCRIPT>
    document.write("&lt;B&gt;Hola JavaScript&lt;/B&gt;");
</SCRIPT>
```

JS –Navigator Objetos (DOM)

- Ej. Jerarquía de objetos (Netscape Navigator). El modelo de objeto de documento puede ser accedido y manipulado con javascript

window
--parent, frames, self, top, location, history
--document
--forms
--elements
(text fields, textarea, checkbox, radio,password, select, button, submit, reset)
--links, anchors

JS –Valores, Variables, Constantes

Hay 4 formas de definir Variables

- Como una variable (**var**)
- Como una Constante con valor fijo predefinido (**const**)
- como una constante de valor variable (**let**)
- No definiendo la variable y solo usándola asignando cualquier valor.

Importante: **let** y **const** fueron agregados en 2015 y no pueden funcionar el algunos navegadores que no contemplen el estándar.

JS –Valores, Variables, cont…

- Existen tres tipos de datos
  - Cadena de caracteres (strings)
    - *“Hola JavaScript”, ‘Hola JavaScript’*
  - Números (Numbers)
    - *1,5,3,14,65000*
  - Booleanos (boolean)
    - *True, False*
- Creación y asignación de variables
  - var pepe
  - pepe=“Ser o no ser”
- Caracteres Especiales
  - \b, \f, \n, \r, \t, \"

JS –Valores, Variables, cont...

- JS interpreta variables en mayúscula y minúscula
  - PEPE, Pepe, pepe, son todas variables distintas
- Las variables deben comenzar con una letra o un Subrayado (_), luego con una combinación de letras números o subrayados.
- JS no es tipado. Las variables no contienen tipos de datos, la conversión entre los tipos de datos la hace automáticamente.

JS –Valores, Variables, cont…

- Para clarificar el código, conviene asignar un tipo de dato a una variable cuando es declarada

```
var primerNombre = “”;
var año = 0;
var vive = true;
```

- La asignación de valores es siempre del lado derecho
- Las variables pueden ser reasignadas

```
Var dato=0;  dato=“hola mundo”; //es válido
```

JS –Valores, Variables, cont...

Let y **const** fueron introducidos en 2015

Const permite definir constantes con valores definidos:

```
const PI = 3.141592653589793;
```

La constante no puede ser redefinida

```
PI=3.14;  //no es valido porque Pi esta definida
PI=PI * 3;//no es valido porque Pi esta definida
```

JS –Valores, Variables, cont…

LET permite definir constantes con valores
Variables:

```
let x=5;
let y =3;
let z=x+y;  //el valor se obtiene de una suma variable
```

Let como const, no puede ser redefinida

```
Let x=4;
Let x=54;
```

Let ya tiene un valor 4. el contenido de x al colocar 54 se anula y no se muestra en pantalla.

Ref. let.html

JS –Variables en bloques

- El ámbito de una variable definida por var o sin usar asignación es global en el ámbito del documento o del a función que es definida.

- Las **let** y **const** pueden ser definidas para un bloque especifico encerrado por llaves **{}**

```
let x=5;
{
        let x =5;
        ……
}
//al finalizar el bloque x vale 5;
//si no se colocan las llaves seria un error de reasignación.
```

Ref.bloques.html

JS –Operadores

- Operadores String
  - Concatenación
    - *String1 + String2 + String3*
- Operadores Numéricos
  - + (suma)
  - - (resta)
  - * (Multiplicación)
  - / (División)
  - % (Módulo) Resto de la división

JS –Operadores Cont…

- Operadores Incrementales
  - ++ (Incremento)
    - *Incrementa el valor de una variable numérica en 1*
    - *y=x++; o y=x+1; ++x;*
- Operadores Decrementales
  - -- (decremento)
    - *Decrementa el valor de una variable numérica en 1*
    - *y=x--; o y=x-1; --x;*

JS – Operadores Cont...

- Operadores de Igualdad
  - = = es igual a
  - != no es igual
  - > es mayor que
  - >= es mayor o igual a
  - < es menor que
  - <= es menor o igual a
- Operadores Lógicos
  - && AND lógico
  - | | OR lógico
  - ! NOT

JS – Respetando operadores

- Si bien JS es no tipado y las variables pueden tomar diferentes tipos de valores, debemos respetar el contenido de las variables
- Respetando el tipo de variable

```
var r=2;
r=r+5;
//valor de r= 7
//suma
```

```
var r="2";
r=r+5;
//valor de r= "25"
//concatenación de in string y un entero
```

- Operaciones sin respetar tipos de variables que no deberían ocurrir

```
var r="2";
r=r*5;
//valor de r= 10
//multiplica
```

```
var r="2";
r=r-5+r;
//valor de r= -32
//resta y luego concatena
```

JS – Palabras Reservadas

- break
  - Termina (corta) el bucle actual y transfiere el control del programa a la siguiente sentencia
- comment
  - Anotaciones hechas por el autor para explicar que es lo que hace el script.
  - Los comentarios son ignorados por el interprete.
- continue
  - Termina la ejecución del bloque de sentencias en un bucle while o for y continúa la ejecución del bucle con la próxima iteración

JS – Control de Flujo

- Sentencias condicionales
  - + If ..... Else ....
  - + If ..... Else if ..... Else ....

```
If (condición a evaluar)
{
    /*Bloque de código que será ejecutado si la condición es verdadera*/
}
Else
{
/*Bloque de código que será ejecutado si la condición es falsa*/
}
```

JS – Control de Flujo


```
<SCRIPT>
    var color="azul";
    if (color=="rojo")
      {
                document.write("el color es rojo");
                document.write("<HR>");
      }
    else if (color=="azul")
      {
                document.write("el color es azul");
      }
    else
      {
                document.write("el color no es ni rojo ni
                        azul");
      }
</SCRIPT>
```

Ref. js02.html

JS – Control de Flujo

- Bucles
  - **For**
    - Crea un bucle que consiste de tres expresiones opcionales encerradas en paréntesis y separadas por (;) y seguidas por un bloque de sentencias que ejecutadas en el bucle
  - **for...in**
    - Itera una variable específica sobre todas las propiedades de un objeto. Para cada propiedad distinta, JavaScript ejecuta las sentencias especificadas
  - **While**
    - Ejecuta un bloque de código mientras la evaluación de la condición sea verdadera

JS – Control de Flujo cont.

- For

```
For (var i=0;i=10; i++)
{
   /*Bloque de código a se ejecutado*/
}
```

- For..in

```
For ( variable in object) {
   sentencias
}
```

- While

```
While (condición)
{
   /*Bloque de código a se ejecutado*/
}
```

JS – Control de Flujo cont.

- For..in

```
<html>
  <body>

  <script type="text/javascript">
  var x;
  var misautos= new Array();
  misautos[0] = "Ferrari";
  misautos[1] = "Volvo";
  misautos[2] = "BMW";

  for (x in misautos)
    {
    document.write(misautos[x] + "<br />");
    }
  </script>

  </body>
</html>
```

Ref. for_in.html

JS – Control de Flujo


```
<SCRIPT>
        var i;
        For (i=0;1<=1000;i=i+100)
           {
                    document.write(i+ “<BR>”);
           }
</SCRIPT>
```

```
<SCRIPT>
        var i;
        while (i<=1000)
           {
                    document.write(i+ “<BR>”);
                    i=i+100;
          }
</SCRIPT>
```

Ref. while.html

JS – Input / Output

- **Alert**
  - Muestra un simple mensaje. Se cierra haciendo clic en el boton “OK” o “Aceptar”
- **Confirm**
  - Da al usuario la opción de elegir “OK” o “Cancel”.
    - *OK=true, Cancel=False*
- **Prompt**
  - prompt(str1,str2)
  - Str1= la instrucción sobre el prompt
  - Str2=texto por defecto (suele ser “”)
- **Console.log**
  - console.log(5 + 6); //Propósitos de debug

JS – Input / Output


```
<SCRIPT>
<!--
if(confirm("Ud. Quiere imprimir texto grande en rojo?")==true)
    {
      document.write("<DIV STYLE='color:red;font-size:36pt;'>texto
    grande en rojo<\/DIV>");
    }
else
{
document.write("texto pequeño en negro");
}
    -->
</SCRIPT>
```

Ref. js03.html

JS – Input / Output cont.


```
<SCRIPT>
<!--
var suNombre=prompt("Por favor escriba su nombre",
   "(sobreescriba este texto con su nombre)");

document.write("Su nombre es:- "+suNombre);
   //-->
</SCRIPT>
```

Ref. js04.html

JS – Arreglos

- Son útiles para crear, ordenar y manipular lista usando bucles
- Creación de un arreglo
  - nombreArray= new Array(); //arreglo vacío
  - nombreArray= new Array(10); //arreglo con 10 items para almacenar
- Propiedad length
  - Da la longitud de un arreglo
  - nombreArray.length();
- Extensión de Arreglo
  - Simplemente si necesitamos extender el tamaño de un arreglo, a través de una asignación le reasignamos el tamaño.

JS – Arreglos Cont.

- Declaraciones

```
gatos=new Array(4);
gatos[0]=Tom;
gatos[1]=Lucas;
gatos[2]=Adam;
gatos[3]=Luis;
.....
gatos=new Array("Tom", "Lucas", "Adam", "Luis");
```

Existen metodos para manejo de arreglos como *sort(), Reverse(), pop(), toSrting(), join(), push(),*

JS – Funciones

- Funciones hacen cualquier cosa, por ej. imprimir mensaje de alerta.
- Pueden tomar un dato, manipularlo y retornar un valor.
- Declaración

```
function nombreFunción()
{
    //Código de la función
}
```

- Llamado a una función

```
var x=nombreFunción();
```

JS – Funciones cont.


```
<SCRIPT>
function warning()
    {
      document.write("<DIV STYLE='color:red;text-
    align:center;font-size:32pt;'>Warning!</DIV>");
      document.write("<BR><DIV STYLE='color:green;text-
    align:center;font-size:24pt;'>Ud. ha sido
    avisado!</DIV>");
    }
var x=warning();
</SCRIPT>
```

Ref. js05.html

JS – Funciones cont.


```
<SCRIPT>
        function multiplicar(a,b)
        {
          var producto=0;
          producto=a*b;
          return producto;
        }
        var x=multiplicar(3,5);
        document.write(x)
</SCRIPT>
```

Ref. js06.html

JS – Objetos

- JS permite definir objetos personalizados

```
var nombreObjeto = new Object
```

```
<SCRIPT>
    var remera=new Object
    remera.color="roja"
    remera.size=8
    remera.type="algodon"
    document.write(remera.color + "<BR>")
    document.write(remera.size + "<BR>")
    document.write(remera.type + "<BR>")
</SCRIPT>
```

Ref. js07.html

JS – Objetos

- JS Contiene un conjunto de métodos preconstruidos

  nombreObjeto.metodo()

```
<SCRIPT>
    var miDia=new Date();
    document.write(miDia + ”<BR>”)
    var x=miDia.toGMTString();
    document.write(x + ”<BR>”)
</SCRIPT>
```

Ref. js08.html

JS – Funciones cont.

- Objetos Matemáticos

E
LN10
LN2
LOG10E
LOG2E
PI
SQRT1_2
SQRT2
abs
acos
atan
atan2
ceil
cos
exp
floor
log
max
min
pow
random
round
sin
sqrt
tan

JS – Modelo de Eventos

- Los eventos suceden a tres niveles:
  - + A nivel del documento HTML
  - + A nivel de un formulario individual
  - + A nivel de un elemento de un formulario
- El evento es gestionado por una sección de código en JavaScript (Gestor de Eventos)
- Declaración de Gestores de Eventos: similar a los atributos en HTML

```
<BODY onLoad="cargarfuncion()" onUnload="descargarfuncion()">

<FORM name="nombre_del_formulario" ...
onSubmit="función_o_sentencia">

<INPUT type="button" name="mycheck" value="HA!" onClick= "alert('Te he dicho que no me aprietes')">
```

JS – Eventos


```
<HTML>
    <HEAD>
        <SCRIPT>
            function funcionClick()
            {
              alert("Ud ha cliqueado")
            }
        </SCRIPT>
    </HEAD>
    <BODY>
        <H2 onclick="funcionClick()">Cliqueame</H2>
    </BODY>
</HTML>
```

Ref. js09.html

JS – Eventos

- El resultado de evento puede cambiar el comportamiento de un Tag html. Por ejemplo para validar el click sobre un determinado link usamos el evento **onclick**.

 ir a mi pagina

Ó bien usando una función que retorne verdadero o falso

 ir a mi pagina

Nótese que antes de la función lleva **return**, esto es para devolverle al comportamiento de la etiqueta el resultado true o false.

Ref.confirm.html

JS – Eventos

- Validar un campo de un formulario antes de ser enviado

```
<html>
<head>
<script>
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
</script>
</head>
```

```
<body>

<form name="myForm"
action="/action_page.php" onsubmit="return
validateForm()" method="post">
  Name: <input type="text" name="fname">
  <input type="submit" value="Submit">
</form>

</body>
</html>
```

Hoy la mayoría de las validaciones se realizan con HTML5 en este caso el atributo required Evita la código anterior

Ref. valida_formulario.html

JS – Gestores de Eventos

Evento	Manejador de Evento	Descripción
Abort	onAbort	Ejecuta código JS cuando el usuario aborta la carga de una imágen.
Blur	onBlur	Ejecuta código JS cuando el elemento del formulario pierde el foco o cuando una ventana o frame pierde el foco (focus).
Change	onChange	Ejecuta código JS cuando Select Text o TextArea pierde el foco cuando el valor del mismo fue modificado.

JS – Gestores de Eventos

Evento	Manejador de Evento	Descripción
Click	onClick	Ejecuta código JS cuando un objeto es cliqueado
DbClick	onDBClick	Ejecuta código JS cuando un usuario hace doble click sobre un elemento o un enlace
Mouse Over Execute	onMouseOver	Ejecuta código JS cada vez que el puntero del mouse se mueve sobre un objeto o un área de un formulario

JS – Gestores de Eventos

onDragDrop
onError
onFocus
onKeyDown
onKeyPress
onKeyUp
onLoad
onMouseDown
onMouseMove
onMouseOut
onMouseUp
onMove
onReset
onResize
onSelect
onSubmit
onUnload

JS – Eventos: Gestores

- Los gestores (manejadores) de eventos son las instrucciones que se ejecutan cuando ocurre un evento
  - + <MARCA ATRIBUTOS manejador="Programa JavaScript">
  - + Colocando funciones:
    - × *<input type="text" onChange="checaCampo(this)">*
  - + Colcando varias instrucciones
    - × *<input type="text" onChange="*
      if(parseInt(this.value) <= 5) {
      alert('Ponga un número mayor que 5.');
      }
      ">

JS – Eventos: palabra clave “this”

- Hace referencia al objeto que produjo el evento
  - Permite acceder a los valores de los atributos de la marca que uno hace referencia, por ejemplo el valor (**value**)

JS – onLoad y onUnLoad


```
<html>
  <head>
  <title>Ejemplo</title>
</head>
  <body
    onLoad="alert('Bienvenido!');"
    onUnload="alert('Adios!');">
  <h1>Página efusiva</h1>
  </body>
</html>
```

JS – onLoad abrir una ventana

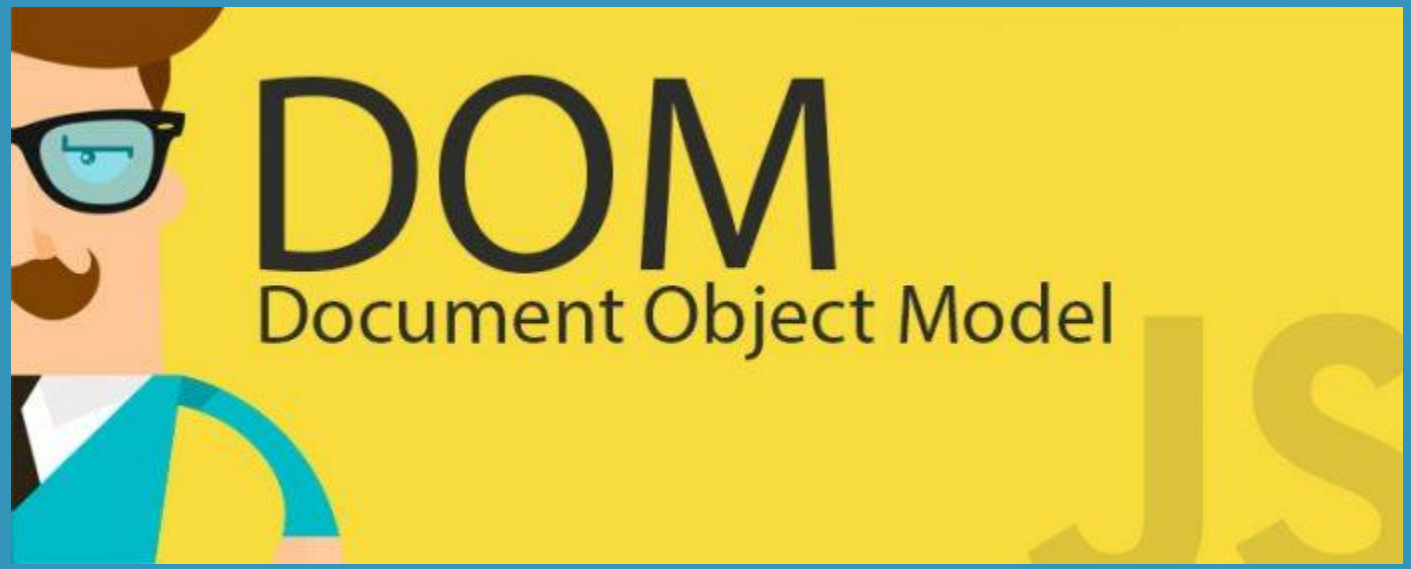
- onUnload="
  window.open(
  ' publicidad.html');“
- Abre una ventana nueva del navegador, en algunos casos puede solicitar permiso para abrirse dependiendo la configuración de confianza que está seteada en el cliente del usuario.

JS – Eventos onMouseOver


```
<a href="#"
    onMouseOver="document.the_image.src='stuff/js_on.gif';"
    onMouseOut="document.the_image.src='stuff/js_off.gif';">
<img src="stuff/js_des.gif" name="the_image"
border="0"></a>
```

La imagen cambia al pasar por encima de la misma. Al salir el evento onMouseOut vuelve a modificar la imagen.

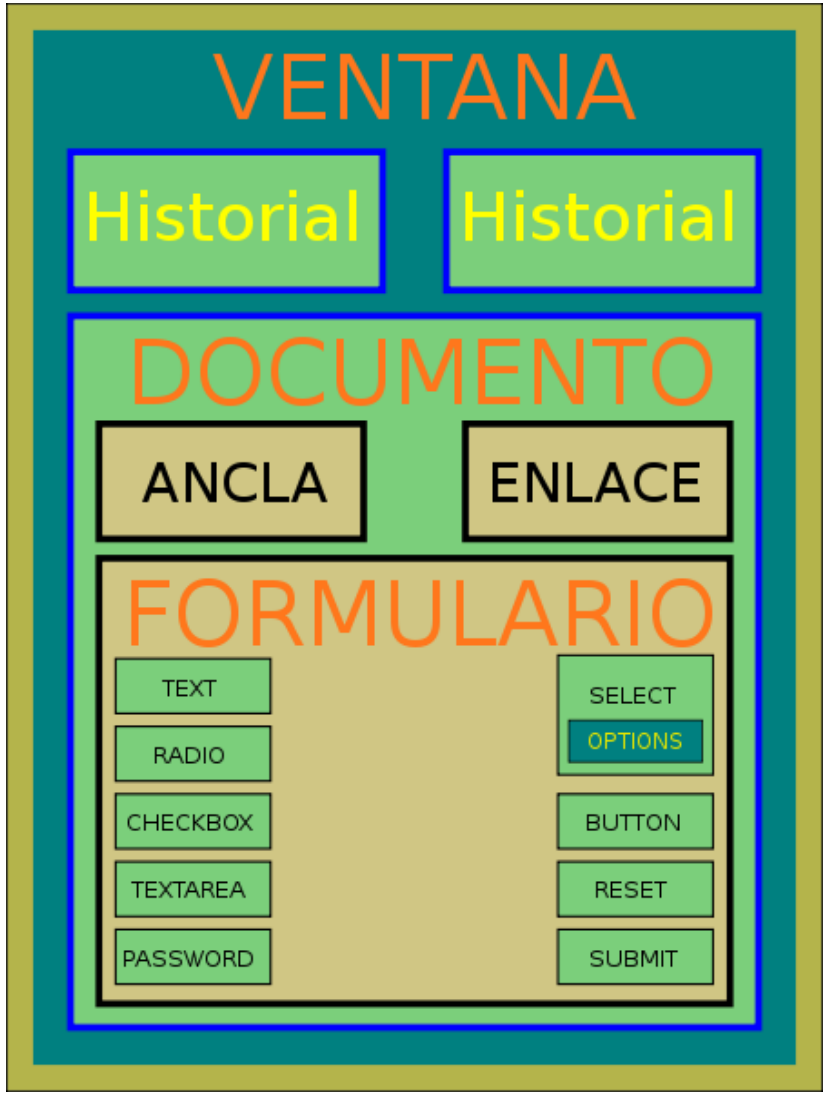
Ref.cambia_imagenes.html


DOM
Document Object Model
JS

DOM – Modelo de Objeto de Documento

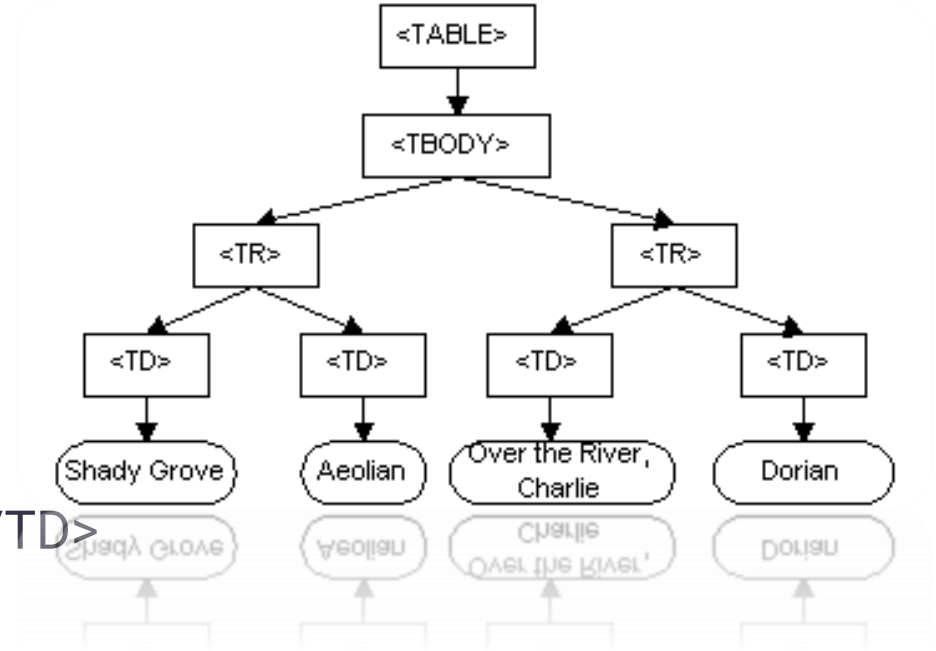
- DOM
  - Modelo de Objetos del Documento' o 'Modelo en Objetos para la Representación de Documentos')
  - Es una (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.
  - A través del DOM, los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.
  - El responsable del DOM es el World Wide Web Consortium (W3C).
  - El DOM es una interfaz de programación de aplicaciones para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos con lenguajes como ECMAScript (JavaScript).

DOM – Modelo de Objeto de Documento



DOM – Instanciación

```
<TABLE>
    <TBODY>
          <TR>
            <TD>Shady Grove</TD>
          <TD>Aeolian</TD>
          </TR>
          <TR>
            <TD>Over the River, Charlie</TD>
        <TD>Dorian</TD>
    </TR>
    </TBODY>
</TABLE>
```



DOM – Instanciación


```
        <HTML>
          <HEAD>
     <TITLE>Ejemplo sencillo de página HTML</TITLE>
   </HEAD>
   <BODY>
      <A name="principio">Este es el principio de la página</A> // ancla
      <HR>
      <FORM method="POST">
         <P> Introduzca su nombre:<INPUT type="text" name="me" size="70">
</P>
         <INPUT type="reset" value="Borrar Datos">
         <INPUT type="submit" value="OK">
      </FORM>
      <HR>
      Clica aquí para ir al
      <A href="#principio">principio</A> de la página // link
   </BODY>
 </HTML>
```

document.title

document.anchors[0].name

document.forms[0].method

document.forms[0].elements[1].value

document.links[0].href

DOM – Objetos del Navegador

- Objetos del navegador
  - Un número de objetos que corresponden a la página, a los contenidos y a otra información relacionada
  - Cada objeto tiene propiedades y métodos
  - Tiene una jerarquía
    - *Refleja la estructura jerárquica de la página HTML.*
    - *Los descendientes de un objeto son propiedades de un objeto*

  Ej. document.title = “un simple documento“;

  document.myform.length = 5;

  document.forms[0].buttons[0].value = “Aceptar" ;

DOM – objetos

Jerarquía de objetos del Navegador

```
window
  + history
  + location
  + document   <BODY> ... </BODY>
    - anchor  <A NAME="..."> ... </A>
    - applet  <APPLET> ... </APPLET>
    - area    <MAP> ... </MAP>
    - form    <FORM> ... </FORM>
      + button     <INPUT TYPE="button">
      + checkbox   <INPUT TYPE="checkbox">
      + fileUpload <INPUT TYPE="file">
      + hidden     <INPUT TYPE="hidden">
      + password   <INPUT TYPE="password">
      + radio      <INPUT TYPE="radio">
      + reset      <INPUT TYPE="reset">
      + select     <SELECT> ... </SELECT>
        - options <INPUT TYPE="option">
      + submit     <INPUT TYPE="submit">
      + text       <INPUT TYPE="text">
      + textarea   <TEXTAREA> ... </TEXTAREA>
    - image   <IMG SRC="...">
    - link    <A HREF="..."> ... </A>
    - plugin  <EMBED SRC="...">
  + frame     <FRAME>
```

DOM – Objetos del Navegador

- **El objeto WINDOW**
  - Es el objeto padre para todos los objetos del navegador.
  - Propiedades
    - ***closed***.
      - Es un booleano que nos dice si la ventana está cerrada (closed = true) o no (closed = false).
    - ***defaultStatus***.
      - Cadena que contiene el texto por defecto que aparece en la barra de estado (status bar) del navegador. (hoy en día deshabilitada por la mayoría de los navegadores)
    - ***frames***.
      - Es un array: cada elemento de este array (frames[0], frames[1], ...) es uno de los frames que contiene la ventana. Su orden se asigna según se definen en el documento HTML.
    - ***history***.
      - Se trata de un array que representa las URLS visitadas por la ventana (están almacenadas en su historial).

DOM – Objetos del Navegador

- **El objeto WINDOW (Propiedades)**
  - ***length***.
    - Variable que nos indica cuántos frames tiene la ventana actual.
  - ***location***.
    - Cadena con la URL de la barra de dirección.
  - ***name***.
    - Contiene el nombre de la ventana, o del frame actual.
  - ***opener***.
    - Es una referencia al objeto window que lo abrió, si la ventana fue abierta usando el método *open*.
  - ***parent***.
    - Referencia al objeto window que contiene el frameset.
  - ***self***.
    - Es un nombre alternativo de la ventana actual.
  - ***status***.
    - String con el mensaje que tiene la barra de estado

DOM – Objetos del Navegador

- **El objeto WINDOW (Propiedades)**
  - ***top***.
    - Nombre alternativo de la ventana del nivel superior.
  - ***window***.
    - Igual que *self*: nombre alternativo del objeto *window* actual.

DOM – Objetos del Navegador

- El objeto WINDOW (Métodos)
  - ***alert(mensaje)***
    - Muestra el mensaje 'mensaje' en un cuadro de diálogo
  - ***blur()***
    - Elimina el foco del objeto *window* actual.
  - ***clearInterval(id)***
    - Elimina el intervalo referenciado por 'id' (ver el método *setInterval()*, también del objeto window).
  - ***clearTimeout(nombre)***
    - Cancela el intervalo referenciado por 'nombre' (ver el método *setTimeout()*, también del objeto *window*).
  - ***close()***
    - Cierra el objeto *window* actual.
  - ***confirm(mensaje)***
    - Muestra un cuadro de diálogo con el mensaje 'mensaje' y dos botones, uno de aceptar y otro de cancelar. Devuelve *true* si se pulsa aceptar y devuelve *false* si se pulsa cancelar.

DOM – Objetos del Navegador

- El objeto WINDOW (Métodos)
  - ***focus()***
    - Captura el foco del mouse sobre el objeto *window* actual.
  - ***moveBy(x,y)***
    - Mueve el objeto *window* actual el número de pixels especificados por (x,y).
  - ***moveTo(x,y)***
    - Mueve el objeto *window* actual a las coordenadas (x,y). A partir de NS 4.

DOM – Objetos del Navegador

- El objeto WINDOW (Métodos)
  - ***open(URL,nombre,caracteristicas)***. *Abre la URL que le pasemos como primer parámetro en una ventana de nombre 'nombre'.*

toolbar = [yes|no|1|0]. Nos dice si la ventana tendrá barra de herramientas (yes,1) o no la tendrá (no,0).

location = [yes|no|1|0]. Nos dice si la ventana tendrá campo de localización o no.

directories = [yes|no|1|0]. Nos dice si la nueva ventana tendrá botones de dirección o no.

status = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de estado o no.

menubar = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barra de menús o no.

scrollbars = [yes|no|1|0]. Nos dice si la nueva ventana tendrá barras de desplazamiento o no.

resizable = [yes|no|1|0]. Nos dice si la nueva ventana podrá ser cambiada de tamaño (con el ratón) o no.

width = px. Nos dice el ancho de la ventana en pixels.

height = px. Nos dice el alto de la ventana en pixels.

outerWidth = px. Nos dice el ancho *total* de la ventana en pixels.

outerHeight = px. Nos dice el alto *total* de la ventana el pixels.

left = px. Nos dice la distancia en pixels desde el lado izquierdo de la pantalla a la que se debe colocar la ventana.

top = px. Nos dice la distancia en pixels desde el lado superior de la pantalla a la que se debe colocar la ventana.

Ref. abrir.html

DOM – Objetos del Navegador

- El objeto WINDOW (Métodos)
  - ***prompt(mensaje,respuesta_por_defecto)***
    - Muestra un cuadro de diálogo que contiene una caja de texto en la cual podremos escribir una respuesta a lo que nos pregunte en 'mensaje'.
  - ***scroll(x,y)***
    - Desplaza el objeto *window* actual a las coordenadas especificadas por (x,y). A partir de NS3, IE4.
  - ***scrollBy(x,y)***
    - Desplaza el objeto *window* actual el número de pixels especificado por (x,y). A partir de NS4.
  - ***scrollTo(x,y)***
    - Desplaza el objeto *window* actual a las coordenadas especificadas por (x,y). A partir de NS4.
  - ***setInterval(expresion,tiempo), setTimeout(expresion,tiempo)***
    - Evalúa la expresión especificada después de que hayan pasado el número de milisegundos especificados en tiempo. Devuelve un valor que puede ser usado como identificativo por clearTimeout(). A partir de NS4, IE4.

DOM – Acceso directo a los nodos

- **getElementsByTagName()**
  - *obtiene todos los elementos de la página XHTML cuya etiqueta sea igual que el parámetro que se le pasa a la función.*

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

muestra cómo obtener todos los párrafos de una página XHTML

Obtener el primer párrafo de la página de la siguiente manera:

```
var primerParrafo = parrafos[0];
```

Recorrer todos los párrafos de la página con el siguiente código:

```
for(var i=0; i<parrafos.length; i++) {
  var parrafo = parrafos[i];
}
```

DOM – Acceso directo a los nodos

- **getElementsByTagName()**

  Otros ejemplos…

```
var parrafos = document.getElementsByTagName("p");
var primerParrafo = parrafos[0];
var enlaces = primerParrafo.getElementsByTagName("a");
```

DOM – Acceso directo a los nodos

- **getElementsByName()**

  busca los elementos cuyo atributo name sea igual al parámetro proporcionado.

  var parrafoEspecial = document.getElementsByName(“teoria");

  *<p name=“Evaluación">... *
  *<p name=“teoria">... *
  * ... *

  Nota: Normalmente el atributo name es único para los elementos HTML que lo definen, por lo que es un método muy práctico para acceder directamente al nodo deseado. En el caso de los elementos HTML radiobutton, el atributo name es común a todos los radiobutton que están relacionados, por lo que la función devuelve una colección de elementos. (no funciona bien con IE6)

DOM – Acceso directo a los nodos

- **getElementById()**
  - *Devuelve el elemento XHTML cuyo atributo id coincide con el parámetro indicado en la función. Como el atributo id debe ser único para cada elemento de una misma página, la función devuelve únicamente el nodo deseado.*

```
var cabecera = document.getElementById(“encabezado");
```

```
<div id=“encabezado">
  <a href="/" id="logo">...</a>
</div>
```

Nota: (no funciona bien con IE6)

DOM – Creación de elementos simples

- **createElement(etiqueta)**

```
// Crear nodo de tipo Element
var parrafo = document.createElement("p");
```

- **createTextNode(contenido)**

```
// Crear nodo de tipo Text
var contenido = document.createTextNode("Hola Mundo!");
```

- **nodoPadre.appendChild(nodoHijo)**

```
// Añadir el nodo Text como hijo del nodo Element
parrafo.appendChild(contenido);

// Añadir el nodo Element como hijo de la pagina
document.body.appendChild(parrafo);
```

DOM – Eliminación de Nodos

- **removeChild()**
  - *Requiere como parámetro el nodo que se va a eliminar. Además, esta función debe ser invocada desde el elemento padre de ese nodo que se quiere eliminar. La forma más segura y rápida de acceder al nodo padre de un elemento es mediante la propiedad nodoHijo.parentNode*

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

<p id=“sacame">...</p>
```

DOM – Acceder a un valor de atributo

- **getAttribute()**
  - *Obtiene el valor de un atributo asociado a un elemento nodo*

```
<img src="stratocaster.jpg" alt="Guitarra Electrica" id="fender">

var ImagenGrande = document.getElementById("fender");
alert( ImagenGrande.getAttribute("src") ); // Alerts "stratocaster.jpg".
```

DOM – Acceder a un valor de atributo

- **setAttribute()**
  - *setea el valor a un atributo*

```
Var ImagenGrande = document.getElementById(“fender");
ImagenGrande.setAttribute("src", “GuittarraElectrica.jpg");
```

DOM – Acceder a un valor de atributo

- **InnerHTML**
  - *Es una de las diferentes formas que tiene JavaScript para mostrar información. Se utiliza en las librerías JS disponibles en la Web como JQuery.*
  - *Se usa para acceder y cambiar el texto html dentro de un elemento.*

(document.write muestra el contenido de salida del script en la página actual borrando el contenido de la misma, mientras que inerHTML muestra el contenido dentro de un elemento de la página)

```
var introduccionDiv = document.getElementsByClassName("intro");
introduccionDiv.innerHTML = "<p>Introducción</p>";
```

DOM – Acceder a un valor de atributo

La propiedad innerHTML mostrando datos en un documento HTML .

```
<!DOCTYPE html>
<html>
	<body>

	<h1>Mi Pagina</h1>
	<p>Esto es un parrafo</p>

	<p id="demo"></p>

	<script>
		document.getElementById("demo").innerHTML = 5 + 6;
	</script>

</body>
</html>
```

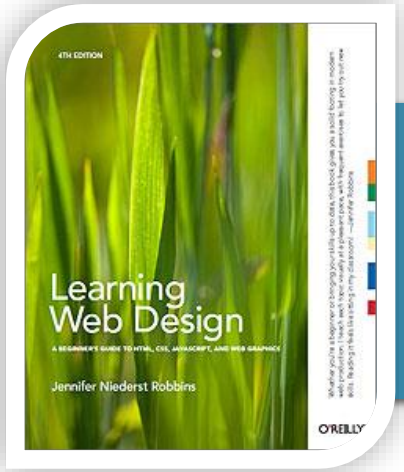
DOM – Acceder a un valor de atributo

- **style**
  - *Para acceder, cambiar y remover CSS dentro de un elemento*

```
document.getElementById("intro").style.color = "#fff";
document.getElementById("intro").style.backgroundColor =
"#f58220";
//Naranja
```

```
var brandColor =
document.getElementById("intro").style.backgroundColor;
```

Bibliografía



Learning Web Design
A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics (Fourth Edition)
Autor Jennifer Niederst Robbins ISBN: 978-1-449-31927-4

Bibliografía/Estándares/Tutoriales

ECMA-6 JavaScript Language Specification
http://www.ecma-international.org

JavaScript WebEstilo
http://www.webestilo.com/javascript/

JavaScript Source –
http://javascript.internet.com /

The Definitive JavaScript Resource JavaScript.com
http://www.javascript.com

Recursos educativos

Campus Virtual Facultad de Ingeniería

http://campusvirtual.ing.unlpam.edu.ar/8010/

Programación Front End 2023