Introducción a JavaScript

Francisco Javier Arce Anguiano

1. Sintaxis de JavaScript

Objetivos: Al finalizar la unidad el participante conocerá los principios del lenguaje JavaScript.

JavaScript es un lenguaje de programación creado por Netscape con el objetivo de integrarse en HTML y facilitar la creación de páginas interactivas sin necesidad de utilizar scripts de CGI o Java.

No hay que confundir Java con JavaScript. Java es un lenguaje completo que permite crear aplicaciones independientes, mientras que JavaScript es un lenguaje que funciona como extensión del HTML, que está orientado a objetos, diseñado para el desarrollo de aplicaciones cliente-servidor a través de Internet.

1.1. Historia de JavaScript

El código de programa de JavaScript, llamado script, se introduce directamente en el documento HTML y no necesita ser compilado. El navegador se encarga de traducir dicho código. JavaScript fue creado por ***Brendan Eich*** en 1995 para Netscape bajo el nombre de "***Mocha***".

Posteriormente cambió de nombre a "***LiveScript***" y más tarde a JavaScript.
En 1997 adoptó el estándar ***ECMAScript 1*** (ES1).

Versión de EcmaScript	Año
1	Junio 1997
2	Junio 1998
3	Diciembre 1999
4	Abandonada

5	Diciembre 2009
6	Junio 2015
7	En proceso

La implementación de la versión ***ES6*** está bastante avanzada en los principales navegadores.

1.2. El primer programa con JavaScript: Hola Mundo

El primer paso es diferenciar dentro de un documento lo que es JavaScript del resto. Para ello HTML dispone de una etiqueta que define el incio y el final de un código JavaScript. Para definir el inicio de un programa se debe utilizar la etiqueta <SCRIPT LANGUAGE=”JavaScript”> y determinar el final con </script>.

El atributo LANGUAGE indica al navegador el lenguaje script utilizado. En el caso que nos ocupa el nombre a poner debe ser JavaScript. Si se omite la especificación del lenguaje, el navegador asume que el script está programado en JavaScript. Para HTML 5 ya no es necesario el atributo ***LANGUAGE***.

```
<!DOCTYPE html>
<html>
<head>
	<title>Hola Mundo</title>
	<meta charset="utf-8">
	<script>
		//los objetos de JS son navigator, document, window y history
		document.write("Hola Mundo");
		alert("Hola Mundo 2");
		console.log("Hola Mundo 3");
	 </script>
```

```
 </head>
<body>
</body>
</html>
```

Listado 2.1.

El texto anterior muestra en pantalla una línea con el texto Hola Mundo. JavaScript es un lenguaje de programación *Case Sensitive*, es decir, que distingue las mayúsculas de las minúsculas, por lo que tendremos que prestar atención a la utilización de variables y comandos.

1.3. JavaScript en un archivo js

Lo correcto es escribir en archivos diferentes el código HTML, que generalmente es el contenido de la página o aplicación, y el código de JavaScript, que es la lógica de nuestra aplicación. Desde la etiqueta <script> debemos utilizar el atributo src (abreviación de source o fuente):

```
<!DOCTYPE html>
<html>
<head>
        <title>Llamar a un archivo externo de JS</title>
        <meta charset="utf-8">
        <script src="js/saludo.js"> </script>
 </head>
<body>
</body>
</html>
```

En un archivo aparte escribimos es código de JavaScript:

```
alert("Hola desde JS externo");
```

1.4. Comentarios al código en JS

Para introducir comentarios en JavaScript podemos utilizar dos técnicas diferentes:

- Los comentarios en una sola línea irán precedidos de //;
- Los comentarios de varias líneas irán encerrados en /* y */.

Ejemplo de un fragmento de código en el que se utilizan los sistemas vistos anteriormente:

```
<!DOCTYPE html>
<html>
<head>
	<title>Comentarios</title>
	<meta charset="utf-8">
	<script>
		//Este es un comentario de línea
		/****Variables******/

		/****Funciones*******/

		/****Inicio ******/

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

1.5. Las llaves(*)

Las llaves se utilizan para definir fragmentos de código de manera que éstos no se junten con el resto del programa.

Todo el código que ejecuta una función debe ir bien diferenciado del resto mediante corchetes.

1.6. El punto y coma (*)

El punto y coma sirve para separar sentencias que se encuentran en una misma línea. También puede indicar el final de una sentencia que ocupa varias líneas.

```
var texto="FAQ(Frequently Asked Questions, Documentos de Preguntas Frecuentes)
Documentos recopilatorios para usuarios principiantes, que contienen las preguntas
más frecuentes sobre un tema determinado.  Son las mejor forma de comenzar a
aprender cosas sobre Internet.";
```

2. Variables y tipos de datos

Objetivos: Al finalizar la unidad el participante conocerá el manejo de variables en JavaScript.

Todos los lenguajes de programación necesitan en algún momento cargar en memoria los datos que se van a procesar. Las variables son fundamentales.

2.1. Introducción a las variables en JavaScript

JavaScript admite prácticamente cualquier tipo de nombre para definir una variable, no obstante, hay una serie de consideraciones que se deben tener presentes:

El primer carácter debe ser siempre una letra o el guión subrayado (_). Los restantes caracteres pueden ser letras, números o el guión subrayado, teniendo como precaución no dejar espacios entre ellos.

El nombre de la variable no debe coincidir con las palabras reservadas de JavaScript. JavaScript diferencia entre mayúsculas y minúsculas. Para declarar variables se utiliza la palabra clave ***var*** seguida del nombre de la variable. Las siguientes variables serán reconocidas como tales por JavaScript.

```
var nombre;
var dirección;
var entrada_valor_concreto;
var variable_numero_12;
```

Ahora se muestran otras variables que no serán reconocidas por JavaScript al no cumplir algunas de las reglas de definición vistas anteriormente.

```
var 1dato;
var entrada datos;
```

```
var while;
var new;
```

Se recomienda utilizar siempre la misma pauta para definir los nombres de las variables. Se puede escribir en minúsculas, o bien la primera mayúscula y las demás minúsculas. Aunque las siguientes variables parezcan iguales, JavaScript las interpretará como diferentes.

```
var resultadosuma
var Resultadosuma
var resultadoSuma
var RESULTADOSUMA
var resultado suma
var resultadosumA
```

2.2. Tipos de datos en JavaScript

JavaScript puede manejar tres tipos de datos distintos decidiendo por nosotros el tipo de variable que deberá emplear durante la ejecución del *script*.

Los tres tipos de variables son:

- Variables de cadena
- Variables numéricas
- Variables booleanas

Un cuarto tipo podrían ser los datos Nulos (***null***). Estos se utilizan para comprobar si a una variable se le ha asignado un valor o no. ***Null*** representa un valor nulo para cualquier tipo de variable; por el contrario, una variable que no ha sido iniciada tiene un valor *undefined*.

2.3. Cadenas en JavaScript

Una variable de cadena es aquella que contiene texto. Las cadenas de texto en JavaScript se delimitan mediante comillas dobles o simples y pueden contener cualquier tipo de carácter. También pueden tener un valor nulo. Ejemplo:

```
var pais=”México”;
var entrada_codigo;
entrada_codigo=”54054”;
var valor=” ”;
```

Las comillas simples serán utilizadas dentro de fragmentos de código delimitados por comillas dobles o viceversa.
Ejemplo:

```
document.write(“Que quiere decir la palabra ‘nuncupatorio’ “)
alert(‘Pulsa la tecla “amigo” ‘)
```

Hay una serie de caracteres que permiten representar funciones especiales, como podría ser un salto de línea en un texto o, por ejemplo, las comillas. A continuación se presenta una lista:

\b	carácter anterior
\f	salto de página
\n	salto de línea
\r	retorno de carro
\t	tabulador
\\	carácter
\‘	comilla simple
\“	comilla doble

```
<html>
<head>
<title>Ejemplo de Cadena de texto</title>
<script language="javascript">
var cadena1="Esto es una cadena de texto que no utiliza ningún caracter especial";
var cadena2="Esto es una cadena \nde texto que si utiliza \ncaracateres especiales";
alert (cadena1);
alert (cadena2);
 </script>
 </head>
<body>
</body>
</html>
```

2.4. Tipos numéricos en JavaScript

Las variables numéricas son aquellas que contienen números enteros o de coma flotante.

Enteros

JavaScript puede utilizar tres bases distintas para números enteros: la decimal (base 10), la hexadecimal (base 16) y la octal (base 8). Por defecto el sistema es el decimal.

```
var numero;
numero = 100;
numero = -1000;
```

Si queremos especificar un número en base octal deberemos anteponer un cero 0 al número. Recordemos que los números en base octal solo pueden contener los dígitos del 0 al 7.

```
var numero_octal;
numero_octal = 03254;
numero_octal = 02;
```

Para un valor hexadecimal deberemos anteponer al número el prefijo 0x. Los números en hexadecimal incluyen los dígitos del 0 al 9 y las letras comprendidas entre la A y la F ambas inclusive.

```
var numero_hex;
numero_hex = 0xff;
numero_hex = 0x12f;
```

Coma flotante

Un valor de coma flotante se compone de un valor entero seguido de un punto y de una fracción decimal. El exponente se indica mediante una e o E seguida por un entero positivo o negativo.

```
var numero_coma_flotante;
numero_coma_flotante=10.236;
numero_coma_flotante=43.433e+2;
numero_coma_flotante= -56.25;
```

2.5. Variables Booleanas

Las variables booleanas o lógicas se especifican mediante los valores verdadero (true) o falso (false).

```
var estoy_contento;
estoy_contento=false;
estoy_contento=true;
```

La utilización de tipos booleanos es de suma importancia cuando se quieren comparar datos o tomar decisiones.

2.6. Operadores matemáticos

Los operadores aritméticos son los encargados de realizar las operaciones básicas como sumar, restar, multiplicar y dividir.

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Módulo (%). Resto de la división.
- Incremento, Preincremento, Posincremento (++)
- Decremento, Predecremento, Posdecremento (--)
- Negación (!)

Operador Aritmético	Código ejemplo	Descripción
+	valor1 + valor2	Suma
-	valor1 – valor2	Resta
*	valor1 * valor2	Multiplicación
/	valor1 / valor2	División
%	valor1 % valor2	Resto división
++	++ valor1	Preincremento
++	valor1 ++	Posincremento
--	-- valor1	Predecremento
--	valor1 --	Posdecremento

-	- valor1	Negación

```
var valor1, valor2, valor3;
 var suma, resta, multiplicacion, division, resto_division, varios;
 valor1=50;
 valor2=10;
 valor3=20;
 suma=valor1+valor2;
 resta=valor1-valor2;
 multiplicacion=valor1*valor2;
 division=valor1/valor2;
 resto_division=valor1%valor2
 varios=(valor1+valor3)*valor2;
 document.write("El resultado de la suma es"+suma+"<br>");
 document.write("El resultado de la resta es"+resta+"<br>");
 document.write("El resultado de la multiplicacion es"+multiplicacion+"<br>");
 document.write("El resultado de la division es"+division+"<br>");
 document.write("El resto de la division es"+resto_division+"<br>");
 document.write("El resultado de la variable varios es"+varios+"<br>");
```

Las variables valor1, valor2, valor3 son las encargadas de contener los números con los que se van a realizar las operaciones aritméticas básicas. Los resultados de dichas operaciones aritméticas básicas. Los resultados de dichas operaciones se guardan en una variable a la que se le ha dado el mismo nombre que el operador utilizado. Así pues, la variable que contiene el resultado de sumar dos números se llama suma, y el resultado de la sustracción se almacena en resta y así sucesivamente.

2.7. Operadores de comparación

Los operadores de comparación son similares a los lógicos, solo que estos no tiene porque ser booleanos. Son los clásicos mayor que, menor que.

== Devuelve true si los dos operandos son iguales.
!= Devuelve true si los dos operandos son diferentes.
> Devuelve true si el operando de la izquierda es mayor que el de la derecha
< Devuelve true si el operando de la izquierda es menor que el de la derecha.
>= Devuelve true si el operando de la izquierda es mayor o igual que el de la derecha.
<= Devuelve true si el operando de la izquierda es menor o igual que el de la derecha.
=== Igualdad estricta
!== Desigualdad estricta

Operador Comparación	Código ejemplo	Descripción
==	valor1 == valor2	Igualdad
! =	valor1 != valor2	Distinto de
<	valor1 < valor2	Menor que
<=	valor1 <= valor2	Menor o igual que
>	valor1 >= valor2	Mayor que
>=	valor1 >= valor2	Mayor o igual que

2.8. Operadores lógicos

Los operadores lógicos o booleanos se emplean para que un programa tome una decisión en función de un cálculo lógico. Los valores que se obtienen son true o false. Los operadores son los siguientes:

- && Suma lógica (Y o And). Este operador devuelve un valor true cuando las dos condiciones comparadas se cumplen. En el supuesto de que una de ellas sea false, el resultado será siempre false.
- || Producto lógico (O u Or). Este operador devuelve true siempre que una de las dos condiciones sea verdadera. En caso contrario devuelve false.

- ! Negación (No o Not). Este operador devuelve siempre el valor contrario, es decir, si la condición o variable es true devuelve false y viceversa.

La verdadera utilidad de estos operadores se ve al trabajar con estructuras condicionales. A continuación verá un resumen con todos los operadores lógicos.

```
true && true   devuelve true
true && false devuelve false
false && false devuelve false

true || true     devuelve true
true || false    devuelve true
false || false   devuelve false
```

Operador lógico	Código ejemplo	Descripción
&&	valor1 && valor2	AND lógico
||	valor1 || valor2	OR lógico
!	! valor1	NOT lógico

2.9. Operadores unarios y atajos

Otro tipo de operadores aritméticos son los incrementales o decrementales, que se aplican antes o después del operando.

```
variable++  //Devuelve el valor de variable y después incrementa su valor en uno.
++variable  //Aumenta el valor de variable en uno y después devuelve su valor.
variable --  //Devuelve en valor de variable y después disminuye su valor en uno.
-- variable  //Disminuye el valor de variable en uno y después devuelve su valor.
```

Veamos la eficacia de este tipo de operadores:

```
pesos=pesos+1; //Aumenta el valor de pesos en 1
pesos++  //Aumenta el valor de pesos en 1
            //El último operador aritmético es la negación.
numero+=100
numero= -numero
```

Un operador de asignación se utiliza para asignar un valor a una variable. Veamos cuáles son, y su sintaxis.

+=	valor1 += valor2	valor1 = valor1 + valor2
-=	valor1 -= valor2	valor1 = valor1 – valor2
*=	valor1 *= valor2	valor1 = valor1 * valor2
/=	valor1 /= valor2	valor1 = valor1 / valor2
%=	valor1 %= valor2	valor1 = valor1 % valor2

Operador Asignación	Código ejemplo	Descripción
+=	valor1 += valor2	valor1=valor1+valor2
-=	valor1 -= valor2	valor1=valor1-valor2
*=	valor1 *= valor2	valor1=valor1*valor2
/=	valor1 /= valor2	valor1=valor1/valor2
%=	valor1 %= valor2	valor1=valor1%valor2

2.10. Operadores de bits (*)

Operador Bit a bit	Código ejemplo	Descripción
&	valor1 & valor2	AND de bits
|	valor1 | valor2	OR de bits
^	valor1 ^ valor2	XOR de bits
>>	valor1 >> valor2	Shift a la derecha
<<	valor1 << valor2	Shift a la izquierda
>>>	valor1 >>> valor2	Shift sin signo

2.11. El operador typeof() (*)

El operador typeof devuelve una cadena que describe el tipo de operando. Así pues, podremos saber si el operando es una cadena, una variable, método, etc. Su funcionamiento es sencillo, basta con escribir a continuación de typeof la variable o elemento.

```
var variable=”Pepe”;
var numero=1;
var fecha=new Date();
document.write(typeof variable + ¨<br>¨);
document.write(typeof numero + ¨<br>¨);
document.write(typeof fecha + ¨<br>¨);
```

El resultado es:

```
variable = string
```

```
numero = number
fecha = object
```

2.12. Palabras reservadas

En la gran mayoría de los lenguajes de programación contamos con ciertas palabras que no las podemos utilizar para crear variables o funciones, las cuales las llamamos como palabras reservadas para el lenguaje. En el caso de JavaScript no podemos utilizar las siguientes palabras como variables, funciones u objetos:

abstract	boolean	break	byte
case	match	char	class
const	continue	default	do
doublé	else	extendí	false
final	finally	float	for
function	goto	int	implements
input	in	instanceof	interface
long	native	new	null
package	private	protected	public
return	short	static	super
switch	this	throw	synchronized
throws	transient	true	try
var	val	while	with

3. Sentencias condicionales

Objetivo: El alumno comprenderá el uso de las diferentes sentencias condicionales con los if, estructuras anidadas o el condicional switch.

3.1. Sentencias condicionales

Se realizan las instrucciones, si la condición en los paréntesis es verdadera
Las llaves { } agrupan el código:

Sintaxis:

```
if(){
        //instrucciones
}
```

Ejemplo:

```
<script>
        var edad = prompt("¿Cuál es tu edad?");
        if(edad > 18){
                alert("Bienvenido a nuestra página");
        }
 </script>
```

Nota: No se muestran las etiquetas de HTML básicas para ahorrar espacio. Puede ver el listado en los archivos del curso.

3.2. El condicional Else

La sentencia ***else*** se ejecuta si no se cumple la condición, si la condición es falsa.

Sintaxis:

```
if(condicional){
        //instrucciones
} else {
        //instrucciones
}
```

Ejemplo:

```
var edad = prompt("¿Cuál es tu edad?");
if(edad > 18){
        alert("Bienvenido a nuestra página");
} else {
        alert("Lo sentimos, no tienes acceso a la página");
}
```

3.3. Estructuras condicionales anidadas

Podemos “anidar” sentencias condicionales. Meter una condicional dentro de otra, sin límites. No se recomienda más de tres anidaciones.

Una estructura de tipo “else if” podemos hacer después de una pregunta falsa, otra pregunta. Si todas son falsas, se ejecuta el “else” Si una pregunta es verdadera, las demás preguntas no se realizan.

```
var nombre = prompt("¿Cuál es tu nombre?");
if(nombre=="Picapiedra"){
        alert("Bienvenido Pedro Picapiedra");
} else if (nombre=="Bond") {
        numero = prompt("¿Cuál es tu número de agente");
        if(numero=="007"){
                clave = prompt("¿Cuál es la clave secreta?");
```

```
                if (clave=="Diamante") {
                        alert("Bienvenido James Bond");
                } else {
                        alert("Lo sentimos, usted es un impostor");
                }
        }
} else if (nombre=="Pablo") {
        alert("Bienvenido Pablo Marmol");
} else {
        alert("Hola, a ti no te conozco");
}
```

3.4.Operadores lógicos en las estructuras condicionales

Podemos hacer varias preguntas dentro de un mismo “if” por medio de los operadores lógicos, AND y OR (&& y ||).

Ejemplo:

```
var dinero = false;
var auto = false;
var amiga = "Martha";
if (dinero || auto || amiga=="Lola") {
        alert("Voy al teatro");
} else if(!dinero && auto && amiga=="Lola"){
        alert("Voy al cine");
} else if(!dinero && !auto && amiga=="Lola"){
        alert("Me quedo en la casa");
} else if(!dinero && !auto && amiga=="Martha"){
        alert("Vemos televisión en la casa");
} else if(!dinero && auto && amiga=="Martha"){
        alert("Ir al futbol");
```

```
} else if(dinero && auto && amiga=="Martha"){
        alert("Vamos a la Opera");
} else {
        alert("Me quedo en casa");
}
```

3.5. La sentencia condicional Switch

Switch, podemos preguntar pero solo de una variable.

Ejemplo:

```
var estadoCivil = "union libre";
var mensaje = "Lo invitamos a un viaje a Chiapas ";
var mensaje2="";
switch(estadoCivil){
        case "soltero":
                mensaje2 = "donde podrás conocer a más personas solteras ";
                break;
        case "casado":
                mensaje2 = "donde podrás disfrutar de la familia ";
                break;
        case "divorciado":
                mensaje2 = "donde podrás olvidar tus penas ";
                break;
        case "viudo":
                mensaje2 = "donde podrás recordar tus mejores tiempos ";
                break;
        case "arrejuntado":
                mensaje2 = "donde podrás tomar esa importante decisión ";
                break;
        default:
```

```
                mensaje2 = "donde podrás definir tu situación personal ";
                break;
}
mensaje3 = " en una extraordinaria experiencia en la naturaleza";
document.write(mensaje+mensaje2+mensaje3);
```

3.6. La sentencia break dentro de un condicional switch

La sentencia “break” rompe la estructura “switch” para continuar con el script.

Ejemplo:

```
var estadoCivil = prompt("¿Cuál es tu estado civil?");
var mensaje = "Lo invitamos a un viaje a Chiapas ";
var mensaje2="";
switch(estadoCivil){
        case "soltero":
        case "soltera":
                mensaje2 = "donde podrás conocer a más personas solteras ";
                break;
        case "casado":
        case "casada":
                mensaje2 = "donde podrás disfrutar de la familia ";
                break;
        case "divorciado":
        case "divorciada":
        case "separado":
        case "separada":
                mensaje2 = "donde podrás olvidar tus penas ";
                break;
        case "viudo":
        case "viuda":
```

```
        case "abandonado":
        case "abandonada":
                mensaje2 = "donde podrás recordar tus mejores tiempos ";
                break;
        case "arrejuntado":
        case "arrejuntada":
                mensaje2 = "donde podrás tomar esa importante decisión ";
                break;
        default:
                mensaje2 = "donde podrás definir tu situación personal ";
}
mensaje3 = " en una extraordinaria experiencia en la naturaleza";
document.write(mensaje+mensaje2+mensaje3);
```

3.7. El operador condicional

Un operador condicional devuelve un valor determinado en función del resultado obtenido al evaluar una condición.

```
(condición)? valor1 : valor2
```

Si la condición es verdadera se devuelve valor1 (verdadero). En caso contrario valor2 (falso).

```
(nombre==”Juan”) ? “Me llamo Juan”:”Tu no eres Juan”;
```

Si la variable **nombre** es Juan, el resultado será “Me llamo Juan”. En caso contrario el resultado será “Tú no eres Juan”.

Cuando sólo tenemos una instrucción en una estructura condicional, podemos omitir las llaves:

```
if(condición) verdadera
else otraSentencia
```

Ejemplo:

```
var importe = 1000;
var tipoCambio = 19.50;
var total = (tipoCambio>0)?importe*tipoCambio : importe;
var mensaje = (tipoCambio>0)?", tipo de cambio "+tipoCambio : "";
document.write("El total de tu pedido es de "+total+mensaje);
```

4. Ciclos en JavaScript

Objetivo: El alumno creará ciclos como whiles, do... while o for.

4.1. El ciclo while

En este ciclo primero preguntamos y luego disparamos.

```
//Valor inicial
var i = 10;
while(i>0){
	document.write(i+"<br>");
	//incremento o decremento
	i -= 2.25;
}
document.write("Salida :"+i+"<br>");
```

4.2. Ciclo do... while

En este ciclo primero disparamos y luego preguntamos.

```
var clave="";
var i = 0;
do{
	clave = prompt("Escribe tu clave de acceso");
	i++;
	if (clave=="007") {
		document.write("Bienvenido agente 007");
		break;
	}
	if(i>3){
		alert("Lo sentimos, no tiene acceso al sistema");
		break;
	}
```

```
}while(true)
```

4.3. El ciclo for

Con el ciclo for(), sabemos el número de iteraciones.

```
for (var i = 10, j=0; i >= 0; i--,j++) {
	if (i%2==1) {
		document.write(i+", "+j+"<br>");
	}
}
document.write("Valor de i al final del ciclo: "+i+"<br>");
```

4.4. Los comandos break y continue en los ciclos

Dentro de un ciclo podemos utilizar las sentencias break, para cortar el ciclo y continue para repetirlo sin ejecutar las instrucciones inferiores.
Ambas se ejecutan dentro de un “if”

Ejemplo:

```
//continue
for (var i = 0; i < 10; i++) {
	if (i%2==0) {
		continue;
	}
	document.write("El valor de i es :"+i+"<br>");
}
//break
var numero = 0;
var intentos = 1;
var magico = 7;
```

```
while(numero!=magico){
	numero=prompt("Adivina el número mágico entre 0 -10 intento "+intentos+" de
3");
	if (numero==magico) {
		alert("El "+magico+" es el número mágico");
		break;
	}
	//incrementamos
	intentos++;
	if (intentos==4) {
		alert("Lo sentimos, eres muy mal adivinador");
		break;
	}
}
```

5: Funciones en JavaScript

Objetivo: El alumno aprenderá a crear funciones y estructurar sus programas a base de ellas.

5.1. Funciones en JavaScript

Las funciones son uno de los pilares en los que se basa JavaScript. Una función es un conjunto de sentencias JavaScript que realizan alguna tarea específica. Las partes que definen una función son:

- El nombre de la función.
- La lista de argumentos de la función, encerrados entre paréntesis y separados por comas(“,”).
- Las sentencias en JavaScript que definen la función, encerradas entre llaves({,});

Una función puede incluir llamadas a otras funciones definidas en la aplicación, pero únicamente de la página actual. Una opción interesante es definir las funciones en el encabezado del documento de manera que, cuando se inicialice la página, las funciones se definan antes de cualquier acción del usuario. La sintaxis de definición de una función sería:

```
function nombreFuncion(parametro1, parametro2…){
    Instrucciones
}
```

La definición de una función no implica su ejecución; ésta sólo se ejecutará cuando se le realice la llamada pertinente.

Ejemplo: Se define una función llamada saludo que será posteriormente llamada para mostrar la cadena de texto especificada.

```
<html>
```

```
<head>
<title>Ejemplo de funciones</title>
<script language="javascript">
function saludo()
{
   document.write("Hola amigos, esto es un saludo");
}
 </script>
 </head>
<body>
<script language="javascript">
   saludo();
 </script>
</body>
</html>
```

5.2. Parámetros en las funciones

Los argumentos de una función permiten que el resultado varíe según el valor indicado de la misma. Estos pueden ser variables, números u objetos.

En el siguiente script, podrá ver el método de definición de los argumentos de la función y posteriormente, el modo de llamar a cada argumento.

```
<html>
<head>
<title>Ejemplo de funciones</title>
<script language="javascript">
function Mensaje(Respuesta)
{
  if(Respuesta==0)  alert("La respuesta es correcta");
  if(Respuesta==2)  alert("La respuesta es incorrecta.  Repasa la leccion 10");
```

```
   if(Respuesta==1)  alert("Vaya disparate. Debes repetir curso");
   if(Respuesta>=3)  alert("Respuesta fuera de rango");
}
 </script>
 </head>
<body>
<script language="javascript">
Mensaje(0);
Mensaje(1);
Mensaje(2);
Mensaje(5);
 </script>
</body>
</html>
```

5.3. Regreso de valores en una función con la sentencia return

Para que una función devuelva el resultado de una serie de operaciones procedentes de la misma función, utilizaremos la instrucción return.

```
<html>
<head>
<title>Ejemplo de funciones</title>
<script language="javascript">
function mitad(valor)
{
   return valor/2;
}
 </script>
 </head>
<body>
<script language="javascript">
```

```
document.write("La mitad de 100 es"+mitad(100));
 </script>
</body>
</html>
```

5.4. Variables locales y globales en las funciones

Las variables pueden ser globales y locales según el lugar en que se hayan definido. Las primeras pueden usarse en cualquier parte del código, mientras que las segundas sólo pueden hacerlo dentro de la función que las define.

```
//Definicion de variable global
var variable1="Esta cadena de texto se ha definido FUERA de una funcion. Es global";
//Definicion de una variable local
function ejemplo()
{
   var variable2="Esta cadena de texto se ha definido DENTRO de una funcion. Es
local";
   document.write(variable2);
}
//Impresion en pantalla de la variable global
document.write(variable1+"<br>");
//Impresion en pantalla de la variable local
ejemplo()
```

Si se intenta utilizar una variable local en un ámbito global, JavaScript dará un mensaje de error diciendo que la variable no está definida.

6: Manejo de cadenas

Objetivo: El alumno modificará las cadenas y subcadenas, así como leerá los caracteres de una subcadena.

6.1. Concatenación de cadenas y conversión de datos

El objeto String ofrece distintas formas de manejar cadenas de texto. Siempre que se asigne un String una variable o propiedad, se crea un objeto de tipo String.

En este caso, al definir una variable con una cadena de texto ya estamos utilizando un objeto String, es decir, no será necesaria si declaración.

```
var cadenaTexto;
cadenaTexto=”Aquí esta la cadena de texto”;
```

Los objetos de tipo String disponen de una serie de métodos que permiten modificar y devolver el valor de la cadena de texto.

Los métodos disponibles para este objeto son:

- anchor(nombre) Crea un enlace
- big() Muestra la cadena de caracteres con una fuente grande.
- blink() La cadena se representa intermitentemente.
- bold() Muestra la cadena en negrita.
- charAt() Muestra el carácter situado en la posición n de la cadena.
- fixed() Muestra la cadena con una fuente no proporcional.
- fontcolor(color) Determina el color de la cadena.
- fontsize(n) Muestra la cadena con una tamaño n. Éste puede oscilar entre 1 y 7.
- indexOf(smallSting,start) Facilita la posición de un fragmento de la cadena a partir de una posición determinada.
- italics() Muestra la cadena en cursiva.

- lastIndexOf() Da como resultado la última posición de un carácter.
- link(URL) Convierte la cadena en un vínculo.
- small() Muestra la cadena con una fuente pequeña.
- strike() Muestra la cadena subrayada.
- sub() Muestra la cadena en formato subíndice.
- substring(x,y) Muestra un fragmento de cadena que empieza en posición x y termina en la posición y.
- sup() Muestra la cadena en formato superíndice.
- toLowerCase() Muestra toda la cadena en minúsculas.
- toUpperCase() Muestra toda la cadena en mayúsculas.

Ejemplo en que se utilizan algunos de estos métodos:

```
var cadena;

 cadena="Bienvenidos al apasionante mundo de JavaScript";
 document.write(cadena+"<br>");
 document.write(cadena.bold()+"<br>");
 document.write(cadena.big()+"<br>");
 document.write(cadena.toUpperCase()+"<br>");
 document.write(cadena.small()+"<br>");
 document.write(cadena.italics()+"<br>");
```

Los objetos **String()** disponen de las propiedades length, que determina el número de caracteres de la cadena, y prototype, que permite añadir nuevas propiedades y métodos (más adelante serán tratadas).

6.2. Métodos para convertir las cadenas a mayúsculas y minúsculas

La propiedad “length” te indica el número de caracteres o letras.
toUpperCase() o toLowerCase() convierten una cadena a mayúsculas o minúsculas.

```
var correo1 = "pacoarce@gmail.com.mx";
var correo2 = "PacoArce@gmail.com.mx";
document.write("La longitu de la clave "+correo1+" es de "+correo1.length+"
caracteres<br>");
if (correo1.length<20) {
        document.write("La clave de acceso es muy corta<br>");
} else {
        if (correo1.toUpperCase()==correo2.toUpperCase()) {
                document.write("Las claves coinciden<br>");
        } else {
                document.write("Las claves NO coinciden<br>");
        }
}
```

6.3. Buscar subcadenas con el método indexOf

El método indexOf() busca una subcadena en una cadena. Inicia en y si no encuentra la subcadena, regresa un -1. El método **indexOf()** es sensible a mayúsculas y minúsculas. Su sintaxis es:

```
cadena.indexOf(valorBusqueda[, indiceDesde])
```

Ejemplo:

```
var correo1 = "pacoarce@gmail.com.mx";
var correo2 = "PacoArce@gmail.com.mx";
//PacoArce@gmail.com.mx
//012345678
var pos = correo2.indexOf("@");
console.log(pos);
var punto = correo2.indexOf(".",pos);
```

```
console.log(punto);
//Primera validación: longitud
if (correo1.length<10) {
        document.write("La clave de acceso es muy corta<br>");
} else if(pos==-1){
        document.write("El correo no tiene la arroba<br>");
} else if(punto==-1){
        document.write("El correo debe de tener al menos un punto<br>");
} else if (!(correo1.toUpperCase()==correo2.toUpperCase())) {
        document.write("Las claves NO coinciden<br>");
} else {
        document.write("Las claves coinciden<br>");
}
```

6.4. Manejo de subcadenas con subString(), subStr() y slice()

Partimos una cadena de una posición con slice()

```
email.slice(pos+1);
```

substr() divide una cadena de un punto determinado cierto número de caracteres:

```
email.substr(0,pos);
```

Ejemplo:

```
var correo1 = "pacoarce@gmail.com.mx";
var correo2 = "PacoArce@gmail.com.mx";
//PacoArce@gmail.com.mx
//012345678
var pos = correo2.indexOf("@");
```

```
console.log(pos);
var punto = correo2.indexOf(".",pos);
console.log(punto);
//Primera validación: longitud
if (correo1.length<10) {
        document.write("La clave de acceso es muy corta<br>");
} else if(pos==-1){
        document.write("El correo no tiene la arroba<br>");
} else if(punto==-1){
        document.write("El correo debe de tener al menos un punto<br>");
} else if (!(correo1.toUpperCase()==correo2.toUpperCase())) {
        document.write("Las claves NO coinciden<br>");
} else {
        servicio = correo2.slice(pos+1).toLowerCase();
        usuario = correo2.substr(0,pos).toLowerCase();
        document.write("Tu servicio de correo es: <b>"+servicio+"</b><br>");
        document.write("Tu usuatrio de correo es: <b>"+usuario+"</b><br>");
        document.write("Las claves coinciden<br>");
}
```

6.5. Leer los caracteres de una cadena

La función ***charAt*** regresa un caracter (i) de una cadena

```
var c = invalidos.charAt(i);
```

En un correo electrónico todos los caracteres son válidos a excepción de:

- las letras acentuadas
- los caracteres de control (CTRL + tecla del teclado)
- los espacios
- los signos especiales como ()<>@,;:"[]ç%&

```
function caracteresInvalidos(cadena){
	var invalidos = "áéíóúÁÍÓÚÑñ ()<>,;:[]ç%&";
	var bandera = false;
	for(i=0; i<invalidos.length; i++){
		var c = invalidos.charAt(i);
		if(cadena.indexOf(c)!=-1){
			bandera = true;
			break;
		}
	}
	return bandera;
}
//Variables
var correo1 = "pacoarce@gmail.com.mx";
var correo2 = "Pacoarce@gmail.com.mx";
//PacoArce@gmail.com.mx
//012345678
var pos = correo2.indexOf("@");
console.log(pos);
var punto = correo2.indexOf(".",pos);
console.log(punto);
//Primera validación: longitud
if (correo1.length<10) {
	document.write("La clave de acceso es muy corta<br>");
} else if(pos==-1){
	document.write("El correo no tiene la arroba<br>");
} else if(punto==-1){
	document.write("El correo debe de tener al menos un punto<br>");
} else if(caracteresInvalidos(correo2)){
	document.write("El correo tiene caracteres inválidos<br>");
} else if (!(correo1.toUpperCase()==correo2.toUpperCase())) {
```

```
        document.write("Las claves NO coinciden<br>");
} else {
        servicio = correo2.slice(pos+1).toLowerCase();
        usuario = correo2.substr(0,pos).toLowerCase();
        document.write("Tu servicio de correo es: <b>"+servicio+"</b><br>");
        document.write("Tu usuatrio de correo es: <b>"+usuario+"</b><br>");
        document.write("Las claves coinciden<br>");
}
```

7: Colecciones: arreglos y objetos

Objetivo: El alumno aprenderá a crear, poblar y eliminar objetos y arreglos.

7.1. Fundamentos en el manejo de arreglos

Una técnica importante de programación es el uso de los arrays, arreglos o vectores. Una matriz es un conjunto de datos de un mismo tipo identificados por un índice.

Desarrollaremos un vector de ejemplo. Primero deberemos definir la matriz, para ello utilizaremos el objeto Array(). Después se deberán ir incluyendo los elementos de esa matriz, los cuales estarán identificados mediante un número al que llamaremos índice.

```
OpenAustralia=new Array();
OpenAustralia[0]=”Sergi Bruguera”;
OpenAustralia[1]=”Alex Corretja”;
OpenAustralia[2]=”Felix Mantilla”;
OpenAustralia[3]=”Peter Sampras”;
OpenAustralia[4]=”Gustavo Kuerten”;
```

Para arreglos mayores hay una propiedad del objeto Array() llamada length, que facilita el número de elementos disponibles en la matriz.

```
document.write(“Tengo”+OpenAustralia.length+”elementos en mi matriz.”);
```

El objeto Array() también tiene tres métodos:

- **join()** Método encargado de agrupar todos los elementos en una cadena de caracteres, separados por comas.
- **reverse()** Este método invierte el orden de colocación de los elementos. El primero pasa a ser el último y viceversa.
- **sort()** Este método devuelve de manera ordenada los elementos de la matriz.

JavaScript no permite eliminar de manera directa elementos de un vector o arreglo, sin embargo, se le puede dar un valor nulo al elemento. Mostraremos distintas sintaxis para eliminar los elementos 3 y 4 de la matriz.

```
OpenAustralia[3]="";
OpenAustralia[4]=null;
```

Con esto los elementos tres y cuatro no tendrán contenido. Ejemplo completo de la matriz OpenAustralia en el que se aplican los métodos vistos.

```
OpenAustralia=new Array();
OpenAustralia[0]="Sergi Bruguera";
OpenAustralia[1]="Alex Corretja";
OpenAustralia[2]="Feliz Mantilla";
OpenAustralia[3]="Peter Sampras";
OpenAustralia[4]="Gustavo Kuerten";
document.write("Tengo "+OpenAustralia.length+" elementos en mi matriz."+"<br>");
document.write("Aqui estan relacionados"+"<br>"+OpenAustralia.join()+"<br>");
document.write("Ahora al reves"+"<br>"+OpenAustralia.reverse().join()+"<br>");
document.write("Ahora ordenados
alfabeticamente"+"<br>"+OpenAustralia.sort().join()+"<br>");
document.write("Ahora ordenados alfabeticamente y al
reves"+"<br>"+OpenAustralia.sort().reverse().join()+"<br>");
```

7.2. Poblar y barrer un arreglo

En sistemas, se le conoce a “barrer” un arreglo a la acción de recorrer todo un arreglo (o archivo u objeto) desde el inicio hasta el final.

Podemos recorrer o barrer un arreglo con un:

un ciclo for (con la propiedad length)

un ciclo for… in

En sistemas, se le conoce como “***poblar***” un arreglo a la acción de agregar elementos como “***poblar***”. Proviene del inglés “***populate***”. La función para añadir un elemento al final utilizamos *push()*.

```
//variable asociada
var dias = new Array();
//crear un arreglo "al vuelo"
var meses = [];

//populate / poblar
dias[0] = "Domingo";
dias[1] = "Lunes";
dias[2] = "Martes";
dias[3] = "Miércoles";
dias[4] = "Jueves";
dias[5] = "Viernes";
dias[6] = "Sábado";

document.write("Hoy es el día "+dias[3]+"<br>");

//barrer el arreglo
for (var i = 0; i < dias.length; i++) {
        document.write("Ho es el día "+dias[i]+"<br>");
}

//arreglos por referencia o asociados
meses["enero"] = 473633;
meses["febrero"] = 6563463;
meses["marzo"] = 635226;
```

```
document.write("Alumnos inscritos en enero "+meses["enero"]+"<br>");

for(alumnos in meses){
        document.write("Alumnos inscritos en el mes de "+alumnos+" es de
"+meses[alumnos]+"<br>");
}
document.write("<hr>");
var jugadores = new Array("Pepe", "Toño", "Gustavo", "Paco");
jugadores.push("Rafael");
for (var i = 0; i < jugadores.length; i++) {
        document.write(jugadores[i]+"<br>");
}
```

7.3. Métodos para manipular arreglos

- sort(): Ordena el arreglo en orden alfabético.
- reverse(): Ordena en orden inverso.
- push(): introduce un elemento al final del arreglo.
- pop(): extrae el último elemento de un arreglo.
- unshift(): introduce un elemento al inicio del arreglo.
- shift(): extrae el primer elemento del arreglo.
- splice(): añade o remueve elementos de un arreglo.
    - index: indica la posición donde se insertarán o removerán los elementos. El primer elemento es cero.
    - howmany: número de elementos a ser removidos. Si es cero, no se remueven.
    - item1...itemX: (opcional) Elementos a ser añadidos al arreglo a partir de la posición indicada.

```
//variable asociada
var dias = new Array();
//crear un arreglo "al vuelo"
```

```
var meses = [];

//populate / poblar
dias[0] = "Domingo";
dias[1] = "Lunes";
dias[2] = "Martes";
dias[3] = "Miércoles";
dias[4] = "Jueves";
dias[5] = "Viernes";
dias[6] = "Sábado";

document.write("Hoy es el día "+dias[3]+"<br>");

//barrer el arreglo
for (var i = 0; i < dias.length; i++) {
        document.write("Ho es el día "+dias[i]+"<br>");
}

//arreglos por referencia o asociados
meses["enero"] = 473633;
meses["febrero"] = 6563463;
meses["marzo"] = 635226;

document.write("Alumnos inscritos en enero "+meses["enero"]+"<br>");

for(alumnos in meses){
        document.write("Alumnos inscritos en el mes de "+alumnos+
        " es de "+meses[alumnos]+"<br>");
}
document.write("<hr>");
var jugadores = new Array("Pepe", "Toño", "Gustavo", "Paco");
```

```
jugadores.push("Rafael");
for (var i = 0; i < jugadores.length; i++) {
        document.write(jugadores[i]+"<br>");
}

document.write("<hr>");
document.write("Los jugadores en orden alfabético son: "+jugadores.sort()+"<br><br>");

document.write("<hr>");
document.write("Los jugadores en orden inverso son:
"+jugadores.reverse()+"<br><br>");

document.write("<hr>");
jugadores.push("Carlos","Messi");
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");

//pop() elimina el último elemento y opcionalmente podemos recuperarlo
document.write("<hr>");
var ultimo = jugadores.pop();
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");
document.write("El jugador expulsado es "+ultimo);

//unshift añade un elemento al inicio del arreglo
document.write("<hr>");
jugadores.unshift("Laura");
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");

//shift() extrae el primer elemento y opcionalmente podemos recuperarlo
document.write("<hr>");
var primero = jugadores.shift();
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");
```

```
document.write("El mejor jugador es "+primero);

//splice() extraemos los elementos del arreglo a partir del "n" elemento
//extraemos "m" elemento
document.write("<hr>");
var expulsados = jugadores.splice(1,2);
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");
document.write("La lista de elementos expulsados es: "+expulsados);

//splice() añadir "n" elementos después de el índice indicado
document.write("<hr>");
jugadores.splice(1,0,"Cristiano","Javier");
document.write("La nueva lista de jugadores es: "+jugadores+"<br><br>");

//join() concatenamos un arreglo por medio de un separador
document.write("La nueva lista de jugadores es: "+jugadores.join(" * "));
```

7.4. Creación de objetos en JavaScript

Un objeto es una agrupación de variables denominadas propiedades que realizan operaciones con las variables propias del mismo, es decir, un conjunto de datos definidos por el usuario junto con las operaciones que actúan sobre ellos. Por ejemplo, las propiedades de un computador, serían su procesador, el disco duro, memoria, etc., y además con él podemos jugar, estudiar, dibujar, por ejemplo.

Propiedades de un objeto

Un objeto en JavaScript tiene una serie de propiedades asociadas a él. Para acceder a dichas propiedades utilizaremos la notación punto.

Ejemplo: Imaginemos un objeto llamado computadora, con las propiedades marca, cpu y memoria.

```
computadora.marca=”HP”;
computadora.cpu=”pentium150”;
computadora.memoria=”64mb”;
```

Un método es una función asociada a un objeto y particular a los objetos del tipo que las define. Así pues, un método es una acción que ejecutamos sobre los datos de un objeto.

Los métodos

Los métodos se definen en el mismo sitio que las funciones y de la misma manera, asociándose posteriormente a un objeto ya existente.

```
Objeto.nombreMetodo=nombreFuncion
```

Y para hacer llamada a método:

```
Objeto.nombreMetodo(parametro1,parametro2..)
```

JavaScript dispone de una serie de objetos predefinidos, pero también podemos crear nuestro propios objetos según sea necesario. Los pasos a seguir para definir un objeto nuevo son dos:

- Definir un tipo de objeto a través de una función
- Crear una instancia del objeto usando la palabra new.

Para definir un tipo de objeto, deberemos especificar el nombre de la función, sus propiedades y métodos.

```
function nombreTipoObjeto(propiedad1,propiedad2,…)
```

```
{
   this.propiedad1=propiedad1
   this.propiedad2=propiedad2
   …
}
```

La palabra reservada ***this*** se utiliza para hacer referencia al objeto actual. La usaremos generalmente cuando pasamos objetos como parámetros a funciones y dentro de éstas para acceder a las propiedades de los objetos.
Ejemplo: Definamos una función llamada Computador con tres propiedades: marca, cpu y memoria.

```
function Computadora(marca,cpu,memoria)
{
   this.marca=marca;
   this.cpu=cpu;
   this.memoria=memoria;
 }
```

Crearemos un objeto para ese tipo, al que llamaremos miComputadora, y le asignaremos valores específicos a sus propiedades.

```
miComputadora = new Computadora(“HP”,”Pentium150”,”64MB”)
```

Recuerde que pueden crearse tantas instancias de un tipo como sean necesarias.

```
miComputadora=new Computadora(“HP”,”Pentium150”,”64Mb”)
miComputadora_Dos=new Computadora(“Philips”,”Pentium200”,”64Mb”)
miComputadora_Tres=new Computadora(“Acer”,”Pentium133”,”164Mb”)
```

Ahora podemos mostrar en pantalla las propiedades de los objetos definidos.

```
function Computador(marca,cpu,memoria)
{
  this.marca=marca;
  this.cpu=cpu;
  this.memoria=memoria;
}
miComputador=new Computador("HP","Pentium150","64Mb")
miComputador_Dos=new Computador("Philips","Pentium200","64Mb")
miComputador_Tres=new Computador("Acer","Pentium133","16Mb")
var mensaje;

mensaje="miComputador\nmarca:"+miComputador.marca+"\n"
+"cpu:"+miComputador.cpu+"\n"
+"memoria:"+miComputador.memoria+"\n";
alert(mensaje);

mensaje="miComputador_Dos\nmarca:"+miComputador_Dos.marca+"\n"
+"cpu:"+miComputador_Dos.cpu+"\n"
+"memoria:"+miComputador_Dos.memoria+"\n";
alert(mensaje);

mensaje="miComputador_Tres\nmarca:"+miComputador_Tres.marca+"\n"
+"cpu:"+miComputador_Tres.cpu+"\n"
+"memoria:"+miComputador_Tres.memoria+"\n";
alert(mensaje);
```

7.5. El objeto Date

El objeto ***Date*** permite trabajar con horas y fechas. JavaScript maneja las fechas en milisegundos desde 1/1/1970 a las 00:00:00 horas. Por lo que no se puede trabajar con fechas anteriores.

En la representación de los meses para valores enteros JavaScript comienza a contar desde 0, por lo tanto, enero será el mes 0 y diciembre el mes11. Los días de la semana se cuentan empezando por 0 para el domingo, 1 el lunes y así sucesivamente.

Para crear una variable de fecha se debe establecer un nombre para la instancia del objeto y su respectivos parámetros. Los formatos pueden ser los siguientes:

- Crea un objeto con la fecha y hora actual
    - Fecha=new Date()
- Crea un objeto con la fecha y la hora en una variable de cadena
    - Fecha=new Date("month day, year[hours:minutes:seconds]")
- Crea un objeto con la fecha en valores enteros
    - Fecha=new Date(year, month, day[,hours,minutes,seconds])

El objeto ***Date*** posee muchos métodos para manejar fechas y horas, así pues, veamos dichos métodos:

- getDate() Devuelve el día del mes actual.
- getDay() Devuelve el día de la semana actual.
- getHours() Devuelve la hora actual.
- getMinutes() Devuelve los minutos actuales.
- getMonth() Devuelve el mes actual.
- getSecond() Devuelve los segundos actuales.
- getTime() Devuelve la hora actual.
- getTimeZoneoffset() Devuelve la diferencia en minutos entre la hora actual y la GTM.
- getYear() Devuelve el año actual.

- parse(datestring) Devuelve el tiempo pasado entre las 00:00:00 horas del 1 de enero de 1970 con respecto a la fecha especificada en datestring.
- setDate(valor) Establece el día del mes.
- setHours(valor) Establece la hora actual.
- setMinutes(valor) Establece los minutos actuales.
- setMonth(valor) Establece el mes actual.
- setSeconds(valor) Establece los segundos actuales.
- setTime(valor) Establece el valor del tiempo actual.
- setYear(valor) Establece el año actual.
- toGMTString() Devuelve el valor del objeto actual en GMT como una cadena.
- toLocateString() Devuelve el valor del objeto actual como una cadena.
- UTC() Devuelve el número de milisegundos transcurridos desde las 00:00:00 del día 1 de enero de 1970.

Los métodos ***set*** los utilizaremos para fijar la fecha y la hora, los métodos ***get*** para obtener fechas y horas, ***to*** para obtener dichos valores como cadenas y ***parse*** para convertir cadenas que tengan fechas y horas.

Ejemplo que imprime el día actual de la semana.

```
fecha=new Date;
dia=fecha.getDay();
if(dia==0){
  document.write("Hoy es Domingo");
}
if(dia==1){
  document.write("Hoy es Lunes");
}
if(dia==2){
  document.write("Hoy es Martes");
}
```

```
if(dia==3){
  document.write("Hoy es Miercoles");
}
if(dia==4){
  document.write("Hoy es Jueves");
}
if(dia==5){
  document.write("Hoy es Viernes");
}
if(dia==6){
  document.write("Hoy es Sabado");
}
```

7.6. El objeto Math

El objeto Math tiene propiedades y métodos que representan constantes y funciones matemáticas. Las propiedades de este objeto son las siguientes:

- E Constante de Euler o número e.
- LN2 Logaritmo de 2.
- LN10 Logaritmo de 10.
- LOG2E Logaritmo de e en base 2.
- LOG10E Logaritmo de e en base 10.
- PI Número PI.
- SQRT1_2 Raíz cuadrada de 0.5.
- SQRT2 Raíz cuadrada de 2.

Ejemplo:

```
var valorPi;
valorPI=Math.PI;
document.write("La variable Pi tiene como valor:"+valorPI);
```

A continuación relacionamos los métodos, que son los elementos que nos permiten realizar operaciones.

- abs(n) Calcula el valor absoluto de n.
- acos(n) Calcula el arcocoseno de n.
- asin(n) Calcula el arcoseno de n.
- atan(n) Calcula el arcotangente de n.
- ceil(n) Redondea un número hacia el superior.
- cos(n) Calcula el coseno de un número n.
- exp(n) Calcula un exponencial del número e.
- floor(n) Redondea un número hacia el inferior.
- log(n) Calcula el logaritmo de un número n.
- max(x,y) Devuelve x o y, en función de cuál de los dos es mayor.
- min(x,y) Devuelve x o y, en función de cuál de los dos es menor.
- pow(x,y) Calcula la potencia de dos números.
- random() Genera un número entero más cercano.
- round(n) Redondea al número entero más cercano.
- sin(n) Calcula el seno de un número n.
- sqrt(n) Calcula la raíz cuadrada de un número n.
- tan(n) Calcula la tangente de un número n.

Es recomendable utilizar la sentencia ***with*** cuando se desean realizar varios cálculos matemáticos seguidos para no tener que poner *Math* en cada referencia.

```
var valorPi;
valorPI=Math.PI;
document.write("La variable Pi tiene como valor."+valorPI+"<br>");
valorPI=Math.ceil(valorPI);
document.write("La variable Pi redondeada mediante ceil tiene  como valor:"+valorPI);
```

Otro ejemplo:

```
var alumnos = new Array("Alejandro", "Claudia", "Eduardo", "Jaime", "Armando",
"Javier");
var numero = Math.random();
document.write("Número Aleatorio: "+numero+"<br>");
numero *= alumnos.length;
document.write("Número Aleatorio: "+numero+"<br>");
numero = Math.floor(numero);
document.write("Número Aleatorio: "+numero+"<br>");
document.write("El alumno que realizará el examen es: "+alumnos[numero]+"<br>");
```

7.7. El objeto Boolean() (*)

El objeto ***Boolean()*** se utiliza para convertir datos no booleanos en booleanos.

Su sintaxis es la siguiente:

```
var nuevoObjeto=new Boolean(valor)
```

Cuando se crea un objeto sin proporcionar datos, el nuevo objeto toma el valor de false. Cuando se facilita el valor true o cualquier otro texto entre comillado, el resultado del nuevo objeto es siempre true. El nuevo objeto contendrá *false* si se proporciona un valor 0, *false* sin comillas o una cadena de texto vacía.

```
 var objeto1,objeto2,objeto3,objeto4;
 objeto1=new Boolean();
 objeto2=new Boolean(false);
 objeto3=new Boolean(true);
 objeto4=new Boolean("texto");
 document.write("El valor booleano de objeto 1 es "+ objeto1 +"<br>");
 document.write("El valor booleano de objeto 2 es "+ objeto2 +"<br>");
 document.write("El valor booleano de objeto 3 es "+ objeto3 +"<br>");
```

```
document.write("El valor booleano de objeto 4 es "+ objeto4 +"<br>");
```

7.8. La función eval() (*)

La función eval() toma una cadena que puede contener cualquier expresión de JavaScript. Así pues, su valor podría ser una cadena representando una expresión JavaScript, una sentencia o una secuencia de sentencias. Esta función puede contener variables y propiedades de objetos ya existentes.

Esta función es útil cuando se trabaja con formularios, ya que los valores introducidos en un formulario son siempre cadenas o valores numéricos.

```
var numero_x=10
var numero_y=20
document.write(eval("numero_x+numero_y"))
```

7.9. Las funciones parseInt() y parseFloat()

Estas dos funciones devuelven un valor numérico a partir de una cadena. La función ***parseFloat(cadena)*** convierte una cadena en un número real en coma flotante. Si la función encuentra un símbolo diferente a los signos + y -, los dígitos del 0 al 9 o el punto o exponente (E o e), se devuelve el valor obtenido hasta ahí ignorando el resto de los caracteres. Si el primer carácter no puede ser convertido se devuelve el valor ***NaN*** (*Not a number*).

La función parseInt(cadena,base) convierte una cadena a una base numérica especificada. Si se encuentra un carácter que no es válido en la base especificada, se devuelve el valor obtenido hasta el momento. Si el primer carácter encontrado no es válido, se devuelve el valor *NaN*. El siguiente script muestra la utilización de *parseInt* y *parseFloat*.

```
var resultadoParseint, resultadoParsefloat;
```

```
var numero_int="1500.230";
var numero_float="+126.256";
resultadoParseint=parseInt(numero_int,10);
resultadoParsefloat=parseFloat(numero_float);
document.write(resultadoParseint+"<br>");
document.write(resultadoParsefloat);
```

7.10. La función escape (*)

La función escape se utiliza para obtener el código ASCII de un carácter en el juego de caracteres ISO Latín 1. Los caracteres alfanuméricos como letras y números serán devueltos tal cual, sólo los símbolos obtendrán su correspondiente código ASCII precedido del símbolo %.

```
document.write(escape("abcdefghi")+"<br>");
document.write(escape("!.$%&/()=?¿")+"<br>");
document.write(escape("123456789")+"<br>");
```

7.11. La función unescape

La función ***unescape*** es opuesta a la anterior. Su misión consiste en representar un carácter a partir de su código *ASCII*[1].

```
document.write(unescape("abcdefghi")+"<br>");
document.write(unescape("%21%B7%24%25%26%28%29%3D%3F%BF")+"<br>");
document.write(unescape("123456789")+"<br>");
```

[1] ASCII (acrónimo inglés de American Standard Code for Information Interchange —Código Estándar Estadounidense para el Intercambio de Información—), es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. Fue creado en 1963 por el Comité Estadounidense de Estándares (ASA, conocido desde 1969 como el Instituto Estadounidense de Estándares Nacionales, o ANSI) como una refundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía. Más tarde, en 1967, se incluyeron las minúsculas, y se redefinieron algunos códigos de control para formar el código conocido como US-ASCII. (Fuente Wikipedia)

8. Document Object Model (DOM)

Objetivo: El alumno comprenderá el modelo orientado de un documento en HTML y su manipulación con JavaScript.

8.1. Documento Object Model (DOM)

- Dom Level 1: publicada en 1998[2].
- Dom Level 2: publicada en 2000.
- Dom Level 3: publicada en 2004.
- Dom Level 4: se esperaba como recomendación oficial en 2016.

El modelo de representación de documentos o DOM (***Document Object Model***) es la forma en que podemos representar un documento. Por "documento" no se refiere a un documento tipo Word, si no a una página en HTML. Podemos representar una página en el modelo DOM como un árbol genealógico, por ejemplo:

Constante (Tipo de nodo)	Significado (hipervínculo al foro W3C)	Valor
ELEMENT_NODE	El nodo es del tipo Element	1
ATTRIBUTE_NODE	El nodo es del tipo Attr	2
TEXT_NODE	El nodo es del tipo Text	3
CDATA_SECTION_NODE	El nodo es del tipo CDATASection	4
ENTITY_REFERENCE_NODE	El nodo es del tipo EntityReference	5
ENTITY_NODE	El nodo es del tipo Entity	6
PROCESSING_INSTRUCTION_NODE	El nodo es del tipo ProcessingInstruction	7
COMMENT_NODE	El nodo es del tipo Comment	8
DOCUMENT_NODE	El nodo es del tipo Document	9
DOCUMENT_TYPE_NODE	El nodo es del tipo DocumentType	10
DOCUMENT_FRAGMENT_NODE	El nodo es del tipo DocumentFragment	11

[2] Las fechas son aproximadas

```
<!DOCTYPE html>
<html>
<head>
	<title>DOM Document Object Model</title>
	<meta charset="utf-8">
 </head>
<body>
<h1>Esta es una página HTML</h1>
<p>Las siglas DOM significan:</p>
<ul id="dom">
	<li>Document</li>
	<li>Object</li>
	<li>Model</li>
</ul>
</body>
	<script>
		var misLI = document.getElementsByTagName("li");
		document.write("El número de elementos de la lista son: 
"+misLI.length+"<br>");
		//Recuperamos un elemento por su identificador (id)
		var miDOM = document.getElementById("dom");
		document.write("El número de nodos de la lista es: 
"+miDOM.childNodes.length+"<br>");
		//Contenido de un nodo
		document.write("El contenido del nodo es: "+miDOM.innerHTML+"<br>");
		//Obtenemos el tipo de nodo
		document.write("El tipo de nodo es: "+miDOM.nodeType+"<br>");
	 </script>
</html>
```

8.2. DOM Seleccionar Nodos

- document.getElementById("identificador");
- document.getElementsByTagName("etiqueta");

miElemento.nodeType
miElemento.innerHTML
miElemento.childNodes.length

```
var misLI = document.getElementsByTagName("li");
document.write("El número de elementos de la lista son: "+misLI.length+"<br>");

//Recuperamos un elemento por su identificador (id)
var miDOM = document.getElementById("dom");
document.write("El número de nodos de la lista es:
"+miDOM.childNodes.length+"<br>");

//Contenido de un nodo
document.write("El contenido del nodo es: "+miDOM.innerHTML+"<br>");

//Obtenemos el tipo de nodo
document.write("El tipo de nodo es: "+miDOM.nodeType+"<br>");
```

8.3. Seleccionar nodos por clase: getElementsByClassName()

Contamos con la instrucción: ***getElementsByClassName()***
La cual nos reporta un arreglo o “nodeList” con los elementos que tienen esta clase.
Esta instrucción es soportada por el IE9+

```
<!DOCTYPE html>
<html>
<head>
```

```
        <title>DOM Document Object Model</title>
        <meta charset="utf-8">
        <script>
        window.onload = function(){
                //seleccionamos los elementos con la clase rojo
                var rojos = document.getElementsByClassName("rojo");
                console.log("Elementos que contienen la clase 'rojo' son: "+rojos.length);
                for (var i = 0; i < rojos.length; i++) {
                        console.log("El contenido de los nodos es :"+rojos[i].innerHTML);
                }
        }
         </script>
        <style>
                .rojo{ color:red; }
        </style>
 </head>
<body>
<h1>Esta es una página HTML</h1>
<p>Las siglas DOM significan:</p>
<ul>
        <li class="rojo">Document</li>
        <li class="rojo">Object</li>
        <li class="rojo">Model</li>
</ul>
</body>
</html>
```

8.4. Seleccionar nodos con querySelector() y querySelectorAll()

Contamos con las instrucciones:

- querySelector()
- querySelectorAll()

El primero reporta sólo un elemento y el segundo un arreglo o 'nodeList'. Esta instrucción es soportada por el IE8+

Podemos seleccionar por clase, identificador, atributo o pseudoclase (CSS).

```
<!DOCTYPE html>
<html>
<head>
	<title>DOM Document Object Model | QuerySelector()</title>
	<meta charset="utf-8">
	<script>
	window.onload = function(){
		//seleccionamos por clase
		var rojos = document.querySelectorAll(".rojo");
		console.log("Tenemos "+rojos.length+" elementos rojos");

		//seleccionamos por identificador
		var amarillos = document.querySelectorAll("#amarillo");
		console.log("Tenemos "+amarillos.length+" elementos amarillos");

		//seleccionamos por atributo
		var verdes = document.querySelectorAll("p[class]");
		console.log("Tenemos "+verdes.length+" elementos verdes");

		//seleccionamos por etiqueta
		var parrafos = document.querySelectorAll("p");
		console.log("Tenemos "+parrafos.length+" párrafos");

		//seleccionamos por clase ficticia
		var noexiste = document.querySelectorAll(".noexiste");
```

```
                console.log("Tenemos "+noexiste.length+" elementos de la clase
'noexiste'");
        }
         </script>
        <style>
                h1{color:blue;}
                .rojo{ color:red; }
                .verde{ color:green; }
                #amarillo{ color:yellow; }
        </style>
 </head>
<body>
<h1>Esta es una página HTML</h1>
<p class="verde">Las siglas DOM significan:</p>
<p>Este es otro párrafo</p>
<ul id="amarillo">
        <li class="rojo">Document</li>
        <li>Object</li>
        <li class="rojo noexiste">Model</li>
</ul>
</body>
</html>
```

8.5. Modificar Nodos de un documento bajo la estructura DOM

- Recuperamos el valor del atributo: **miElemento.getAttribute("align");**
- Modificamos el valor de un atributo: **miElemento.setAttribute("align","left");**

```
window.onload = function(){
        //recuperamos los li
        var misLI = document.getElementsByTagName("li");
        console.log("El número de elementos de tipo lista es de: "+misLI.length);
```

```
        var miDOM = document.getElementById("dom");
        console.log("El tipo de nodo de DOM es: "+miDOM.nodeType);
        //cambiamos el atributo de dom
        miDOM.setAttribute("class","rojo");
}
```

Nota: No desplegamos las etiquetas HTML por cuestiones de espacio.

8.6. Modificar los atributos de un nodo con hasAttribute y removeAttribute

Los métodos para modificar los nodos:

- getAttribute()
- setAttribute()
- removeAttribute()
- hasAttribute()

```
window.onload = function(){
        //recuperamos los li
        var misLI = document.getElementsByTagName("li");
        console.log("El número de elementos de tipo lista es de: "+misLI.length);

        var miDOM = document.getElementById("dom");
        console.log("El tipo de nodo de DOM es: "+miDOM.nodeType);

        //cambiamos el atributo de dom
        miDOM.setAttribute("class","rojo");
}
```

8.7. Modificar, recuperar y eliminar texto de un nodo

Por medio del siguiente método, podemos modificar, recuperar su valor o remover texto con o sin etiquetas:

- innerHTML

```
window.onload = function(){
	var noexiste = document.querySelector(".noexiste");
	if (noexiste.hasAttribute("class")) {
		noexiste.removeAttribute("class");
	} else {
		console.log("No existe la clase noexiste");
	}
}
```

8.8. Navegar en el DOM: nextSibling, nextElementSibling, previousElementSibling

Los métodos para moverse a través del árbol DOM son:

- parentNode (hacia arriba)
- previousSibling, previousElementSibling (izquierda)
- nextSibling, nextElementSibling(derecha)
- firstChild (abajo)
- lastChild (abajo)

En estos métodos todos los navegadores tomarán los espacios (entre las etiquetas) como nodos, a excepción de IE.

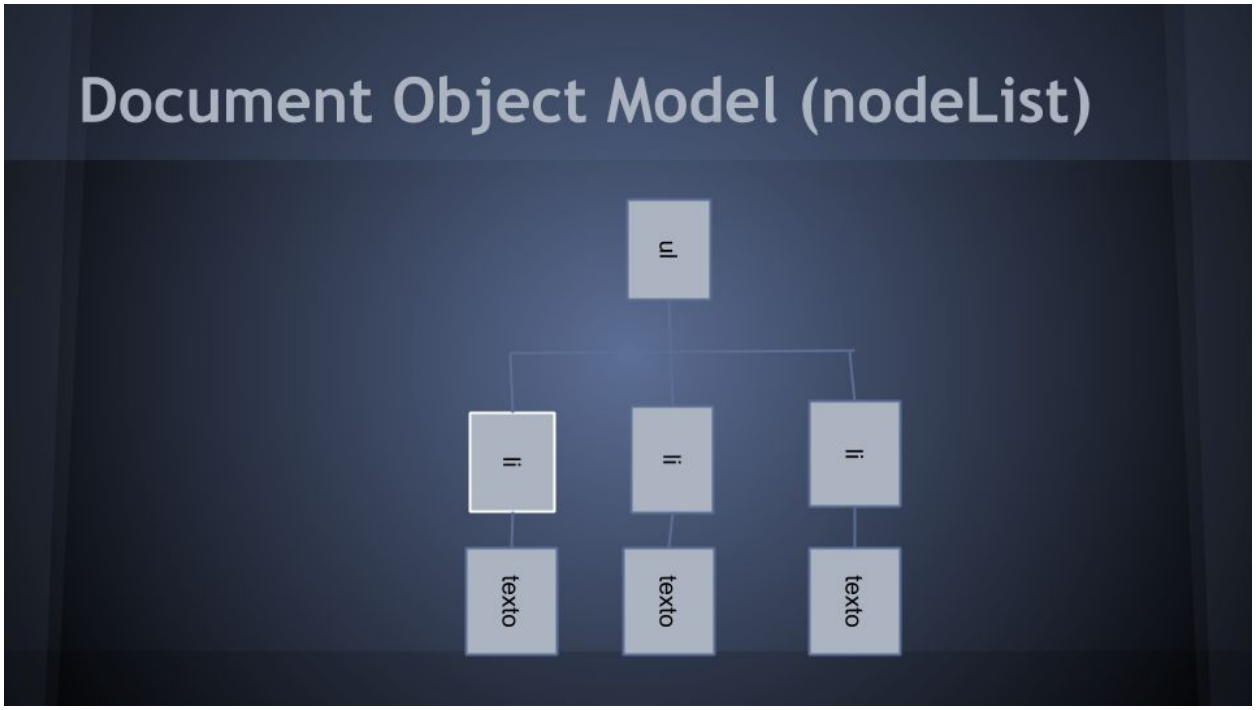

Imagen 8.8.1. DOM (Document Object Model)

```
window.onload = function(){
	var medio = document.getElementById("medio");
	do{
		console.log(medio.nodeName, medio.innerHTML);
		medio = medio.nextElementSibling;
	}while(medio)
}
```

8.9. Navegar en la estructura DOM con JavaScript

Ejemplo de navegación en un documento de tipo DOM:

```
window.onload = function(){
	var medio = document.getElementById("medio");
	medio.previousElementSibling.setAttribute("class","rojo");
	medio.nextElementSibling.setAttribute("class","verde");
```

```
        //medio.setAttribute("class","azul");
        medio.parentNode.parentNode.setAttribute("class","azul");
}
```

8.10. Navegar en el DOM: firstChild y lastChild

Los métodos para moverse a través del árbol DOM son:

- firstChild
- lastChild

```
window.onload = function(){
        var medio = document.getElementById("medio");

        medio.previousElementSibling.setAttribute("class","rojo");
        medio.nextElementSibling.setAttribute("class","verde");
        //medio.setAttribute("class","azul");
        medio.parentNode.parentNode.setAttribute("class","azul");
}
```

8.11. Crear nodos bajo la estructura DOM

Podemos crear nuevos nodos en un documento de tipo DOM por medio de los métodos:

- createElement("p"): creamos el nodo vacío.
- innerHTML: poblamos de contenido el nodo.
- appendChild(nodo): mostramos en el documento DOM en nodo.

```
<!DOCTYPE html>
<html>
<head>
        <title>DOM | Crear nodos</title>
        <meta charset="utf-8">
```

```
        <script>
        window.onload = function(){
                //creamos un arreglo
                var textos = new Array();
                textos.push("Afrodita nació de la espuma del mar...");
                textos.push("Una pelota de golf puede ser más letal que una bala...");
                textos.push("Las abejas tienen emociones...");
                textos.push("El arte es bueno para la salud...");
                textos.push("Las papas disminuyen la presión arterial...");
                textos.push("JavaScript es el lenguaje más utilizado en el mundo...");
                //creamos un número aleatorio
                var num = Math.floor(Math.random()*textos.length);

                //1) Creamo un nodo
                var titulo = document.createElement("h1");
                var texto = document.createElement("p");

                //2) poblar contenidos
                titulo.innerHTML = "¿Sabias que...?";
                texto.innerHTML = textos[num];

                //3) Añadimos a la lista de visualizacion del documento
                document.getElementById("sabias").appendChild(titulo);
                document.getElementById("sabias").appendChild(texto);
        }
         </script>
 </head>
<body>
<div id="sabias"></div>
</body>
</html>
```

8.12. Remover un nodo del DOM con removeChild()

Eliminamos un nodo con el método:

```
var ant = elemento.removeChild(child);
ó
elemento.removeChild(child);
```

Nota: **Un nodo no puede eliminarse a sí mismo. Tenemos que eliminarlo desde su nodo padre.**

Podemos recuperar el nodo eliminado en otro nodo. El nodo “eliminado” sigue existiendo pero ya no se visualiza en el DOM.

```
window.onload = function() {
	/*
	var medio = document.getElementById("medio");
	var ul = document.querySelector("ul");
	var ant = ul.removeChild(medio);
	console.log(ant.nodeName);
	*/
	var ul = document.querySelector("ul");
	while(ul.firstChild) ul.removeChild(ul.firstChild);
}
```

9: Hacer una galería con instrucciones DOM

Objetivo: El alumno utilizará lo aprendido hasta el momento para desarrollar una galería de imágenes con JavaScript.

9.1. Ejercicio terminado de la galería con intrusiones DOM

Los pasos para hacer esta galería son:

- Crear etiquetas HTML
- Detectar la imagen que se pulsó
- Crear el fondo negro
- Crear la imagen
- Centrar imagen y eliminar la imagen

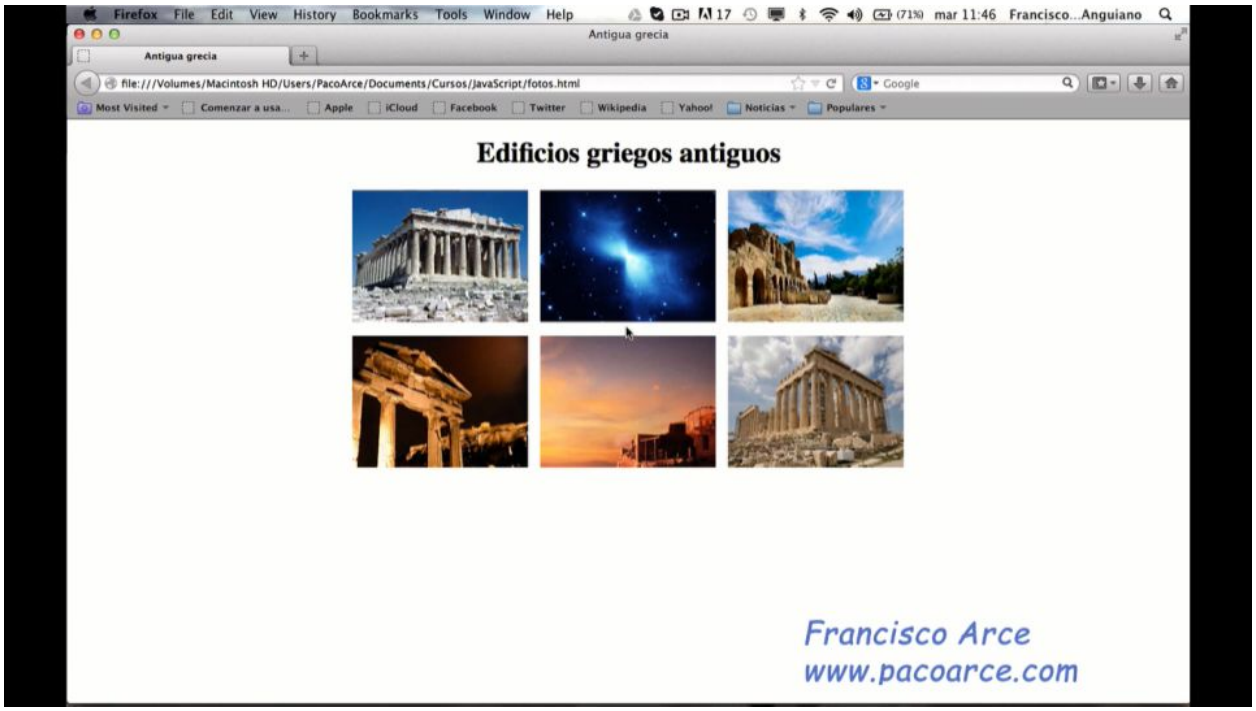

Imagen 9.1.1. Galería de imágenes

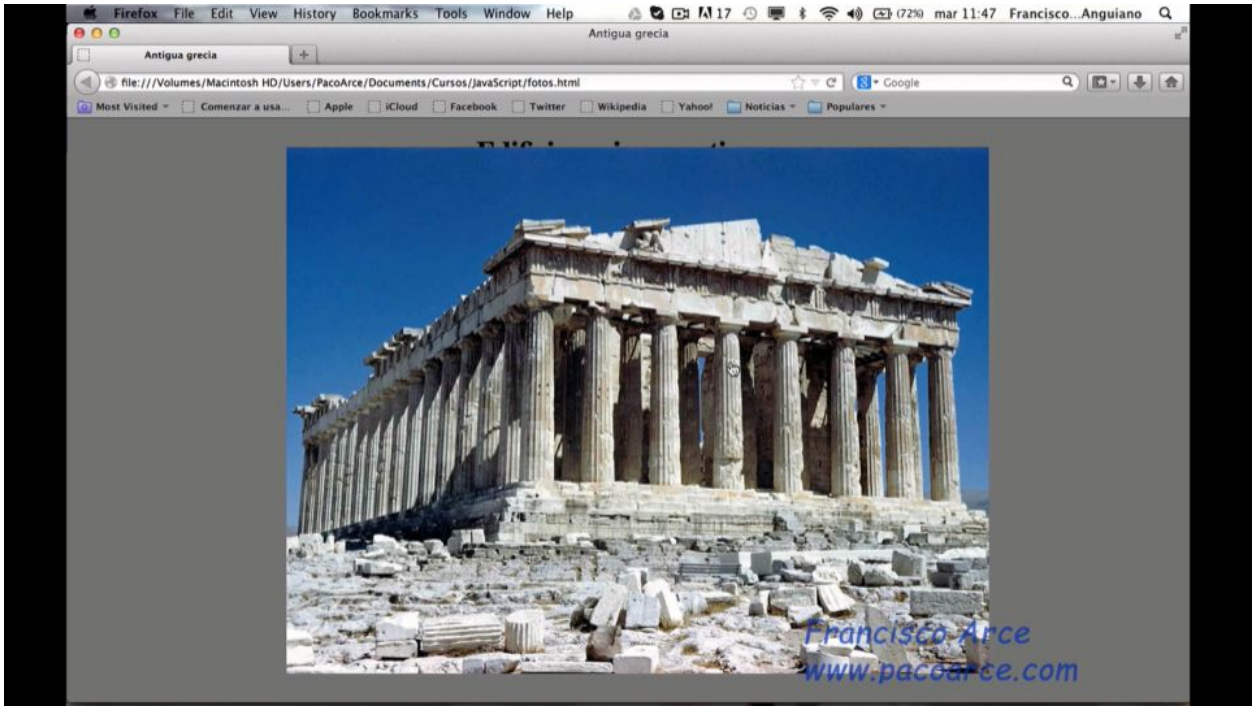

Imagen 9.1.2. Desplegar la imagen

9.2. Escribir el HTML y el CSS de la galería y su archivo JS

El archivo base HTML:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Edificios Griegas</title>
<script src="js/fotos01.js"> </script>
<style>
body{
        width:650px;
        margin:30px auto;
}
h1{
        text-align:center;
```

```
}
img{
        margin-left:10px;
        margin-top:10px;
        cursor:pointer;
}
</style>
 </head>
<body>
<h1>Edificios antiguos griegos</h1>
<div>
<img src="imagenes/fondos/minis/fondo1.jpg" width="200" height="150">
<img src="imagenes/fondos/minis/fondo2.jpg" width="200" height="150">
<img src="imagenes/fondos/minis/fondo3.jpg" width="200" height="150">
<img src="imagenes/fondos/minis/fondo4.jpg" width="200" height="150">
<img src="imagenes/fondos/minis/fondo5.jpg" width="200" height="150">
<img src="imagenes/fondos/minis/fondo6.jpg" width="200" height="150">
</div>
</body>
</html>
```

9.3. Detectar el evento "click" sobre las imágenes

El archivo JavaScript:

```
window.onload = function(){
        var fotos = document.querySelector("div");
        //Listener
        fotos.addEventListener("click",function(e){
                if(e.target.tagName==="IMG"){
                        console.log("Pulso una imagen");
                } else {
```

```
                console.log("NO pulso una imagen");
            }
        }, false);
}
```

9.4. Crear la pantalla negra o "blackout" al pulsar una imagen

```
window.onload = function(){
        var fotos = document.querySelector("div");
        //Listener
        fotos.addEventListener("click",function(e){
            if(e.target.tagName==="IMG"){
                var fondoNegro = document.createElement("div");
                fondoNegro.id = "overlay";
                document.body.appendChild(fondoNegro);
                //Darle estilo al fondo
                fondoNegro.style.position="absolute";
                fondoNegro.style.top = 0;
                fondoNegro.style.backgroundColor = "rgba(0,0,0,0.5)";
                fondoNegro.style.cursor = "pointer";
                //
                fondoNegro.style.width = window.innerWidth+"px";
                fondoNegro.style.height = window.innerHeight+"px";
                fondoNegro.style.top = window.pageYOffset+"px";
                fondoNegro.style.left = window.pageXOffset+"px";
            } else {
                console.log("NO pulso una imagen");
            }
        }, false);
}
```

9.5. Cargar la imagen grande en el fondo negro con instrucciones DOM

```
window.onload = function(){
	var fotos = document.querySelector("div");
	//Listener
	fotos.addEventListener("click",function(e){
		if(e.target.tagName==="IMG"){
			var fondoNegro = document.createElement("div");
			fondoNegro.id = "overlay";
			document.body.appendChild(fondoNegro);
			//Darle estilo al fondo
			fondoNegro.style.position="absolute";
			fondoNegro.style.top = 0;
			fondoNegro.style.backgroundColor = "rgba(0,0,0,0.5)";
			fondoNegro.style.cursor = "pointer";
			//
			fondoNegro.style.width = window.innerWidth+"px";
			fondoNegro.style.height = window.innerHeight+"px";
			fondoNegro.style.top = window.pageYOffset+"px";
			fondoNegro.style.left = window.pageXOffset+"px";
		} else {
			console.log("NO pulso una imagen");
		}
	}, false);
}
```

9.6. Centrar la imagen y eliminarla al pulsarla

El listado final de la galería:

```
window.onload = function(){
```

```
var fotos = document.querySelector("div");
//Listener
fotos.addEventListener("click",function(e){
	if(e.target.tagName==="IMG"){
		var fondoNegro = document.createElement("div");
		fondoNegro.id = "overlay";
		document.body.appendChild(fondoNegro);
		//Darle estilo al fondo
		fondoNegro.style.position="absolute";
		fondoNegro.style.top = 0;
		fondoNegro.style.backgroundColor = "rgba(0,0,0,0.5)";
		fondoNegro.style.cursor = "pointer";
		//
		fondoNegro.style.width = window.innerWidth+"px";
		fondoNegro.style.height = window.innerHeight+"px";
		fondoNegro.style.top = window.pageYOffset+"px";
		fondoNegro.style.left = window.pageXOffset+"px";
		//Crear la imagen
		var archivoMini = e.target.src;
		//console.log(archivoMini);
		var archivo = archivoMini.substr(archivoMini.length-10, 10);
		//console.log(archivo);
		var imagenGrande = "imagenes/fondos/"+archivo;
		//console.log(imagenGrande);
		var imagen = document.createElement("img");
		imagen.id = "imagen";
		imagen.src = imagenGrande;
		imagen.style.display = "block";
		imagen.style.position = "absolute";
		imagen.addEventListener("load", function(){
			centrar(this);
```

```
                        fondoNegro.appendChild(imagen);
                    }, false);
                    imagen.addEventListener("click", function(){
                        if(fondoNegro){
                            fondoNegro.parentNode.removeChild(fondoNegro);
                        }
                    }, false);
                } else {
                    console.log("NO pulso una imagen");
                }
        }, false);
        function centrar(imagen){
                var xx = (window.innerWidth - imagen.width)/2;
                var yy = (window.innerHeight - imagen.height)/2;
                imagen.style.top = yy+"px";
                imagen.style.left = xx+"px";
                return imagen;
        }
}
```

10: Objetos del navegador BOM (Browser Object Model)

Objetivo: El alumno manejara los objetos propios del navegador con JavaScript.

10.1. Objetos Navegador

En este capítulo, veremos de manera superficial las características más relevantes de los objetos del navegador JavaScript.

Cuando se carga una página en un navegador, se crea un número de objetos característicos del navegador según el contenido de dicha página. A continuación veremos los objetos y propiedades que tiene un documento:

- window. Es el objeto de más alto nivel, contiene las propiedades de la ventana y en el supuesto de trabajar con frames, un objeto window es generado para cada frame.
- location. Contiene las propiedades de la URL activa.
- history. Objeto que contiene las propiedades que representan a las URL que el usuario ha visitado anteriormente. Es una especia de caché.
- document. Este objeto contiene todas las propiedades del documento actual, como pueden ser su color de fondo, enlaces, imágenes, etc.

Los objetos en el navegador se rigen por una jerarquía que refleja la estructura de los documentos HTML. Según esto, el objeto window que es el de más alto nivel, tendría a un objeto location como descendiente.

Imaginemos un documento HTML(object document) que contiene un formulario llamado miformulario. Para hacer referencia al formulario se deberá escribir:

```
document.miformulario
```

Como norma general para referenciar una propiedad específica de un objeto para referenciar una propiedad específica de un objeto se deberá incluir el objeto y todos

sus antecesores teniendo en cuenta que el objeto window no es necesario incluirlo a no se que se esté trabajado con frames. Veamos a continuación la jerarquía de los objetos de un navegador.

window

- parent, frames, self, top
- history
- location
- document
    - links
    - anchor
    - form
        - Todos sus elementos

10.2. El objeto Window

El objeto window posee una serie de propiedades que determinan características básicas de la ventana y sus componentes. A continuación las propiedades mas elementales:

- closed. Propiedad que determina si una ventana se ha cerrado.
- defaultStatus. Propiedad que contiene el mensaje estándar que aparece en la barra de estado de windows.
- frames. Es una matriz que representa todos los frames de la ventana.
- lenght. Esta propiedad contiene el número de frames de la ventana.
- name. Contiene el nombre de la ventana.
- outerHeight. Altura de la totalidad de la ventana.
- outerWidth. La anchura de la totalidad de la ventana.
- parent. Hace referencia a la ventana con un código <FRAMESET>/
- self. Propiedad que hace referencia a la ventana activa.
- top. Hace referencia a la ventana superior del navegador.
- status. Determina el mensaje que aparece en la barra de estado del navegador.
- window. Al igual que self, hace referencia a la ventana activa.

Ejemplo: Diseñaremos un programa que muestre siempre un texto en la barra de estado y que oculte la dirección real de un enlace al pasar el cursor del ratón sobre el.

```
<body onload="window.defaultStatus='Curso basico de JavaScript';">
<a href="http://www.enlace.com/"
onMouseOver="window.status='Estas encima del enlace';return true">
pasa por encima mio.</A>
</body>
```

Ejemplo de aplicación de los métodos alert(), confirm() y prompt():

```
<html>
<head>
<title>Ejemplo de ABRIR y CERRAR una ventana</title>
<script language="javascript">

function Pregunta(){
var EntradaDatosPregunta=prompt("Intoduce tu nombre, por favor","en
minusculas,gracias");
  if(confirm("Estas conforme con el nombre introducido"+EntradaDatosPregunta+"?"))
  {
     alert("De acuerdo, escribiste"+EntradaDatosPregunta);
  } else {
    alert("Bueno, pero yo creo que escribiste"+EntradaDatosPregunta);
  }
}
 </script>
 </head>
<body>
<form>
```

```
<input type="button" value="Pulsa Aqui" onClick="Pregunta()">
</form>
</body>
</html>
```

10.3. Crear ventanas emergentes con los objetos *window* y *document*

El objeto window también posee una serie de métodos que permiten ejecutar funciones específicas con las ventanas, como por ejemplo crear ventanas y cuadros de diálogo.

- open() y close(). Métodos que abren y cierran una ventana.
- back(). Retrocede a la página anterior.
- blur(). Quita el foco de la ventana especificada.
- captureEvents(). Captura todos los eventos de un determinado tipo.
- clearInterval(). Cancela el tiempo de espera establecido mediante setInterval().
- close(). Cierra la ventana.
- alert(). Método que muestra una ventana de diálogo con un mensaje y el botón Aceptar.
- confirm(). Método similar al anterior, pero mostrando dos botones, Aceptar y Cancelar.
- find(). Abre una ventana de diálogo que permite efectuar búsquedas.
- prompt(). Método que representa una ventana de diálogo con mensaje y un campo de entrada.
- setTimeout. Este método retrasa la ejecución de una instrucción.
- clearTimeout. Método que permite anular el timeout fijado con el método anterior.

También podemos determinar el aspecto que tendría la nueva ventana del navegador mediante una serie de componentes que permiten configurar el menú, la barra de herramientas, la barra de estado, etc. Las opciones son:

- toolbar. Muestra la barra de herramientas.
- location. Muestra la barra de dirección.
- directories. Muestra los botones de directorio.

- status. Muestra la barra de estado.
- menubar. Muestra la barra de menús.
- scrollbars. Muestra las barras de desplazamiento.
- resizable. Permite ajustar el tamaño de la ventana.
- width. Ancho de la ventana en pixeles.
- height. Altura de la ventana en pixeles.

Para abrir una ventana utilizando los métodos y opciones anteriores, deberemos aplicar la siguiente sintaxis:

```
variableVentana=
nombreventana.open(“URL”,”NombreVentana”,”OpcionesVentana”)
```

Script que abre una ventana nueva cuando se pulsa un botón.

```
<html>
<head>
<title>Ejemplo de ABRIR y CERRAR una ventana</title>
<script language="javascript">
function AbrirVentana()
{
Ventana=open("","nueva","toolbar=no,directories=no,menubar=no,width=180,height=16
0");
Ventana.document.wrtie("<head><title>Ventana Nueva</title> </head><body>");
Ventana.document.write("<font size=4 COLOR=blue>VENTANA NUEVA</font><br>");
Ventana.document.write("<form><input type='button' VALUE =
'Cerrar'onClick='self.close()'> </form>");
}
 </script>
 </head>
<body>
```

```
<form>
<input type="button" value="Abrir una ventana"
onClick="AbrirVentana();"><br>
</form>
</body>
</html>
```

10.4. Las propiedades innerWidhth e innerHeight

```
<!DOCTYPE html>
<html>
<head>
	<title>Bom | Size</title>
	<meta charset="utf-8">
	<script>
	window.onload = function(){
		var w = window.innerWidth || document.documentElement.clientWidth ||
document.body.clientWidth;
		var h = window.innerHeight || document.documentElement.clientHeight ||
document.body.clientHeight;
		document.getElementById("salida").innerHTML = "Dimensiones del
viewport son "+w+" ancho y "+h+" altura en pixeles y las dimensiones externas son
"+window.outerWidth+" de ancho y "+window.outerHeight+" de altura en piexeles";
	}
	 </script>
 </head>
<body>
<div id="salida"></div>
</body>
</html>
```

10.5. BOM: el objeto screen

```
document.write("Ancho: "+window.screen.width+" pixeles<br>");
document.write("Altura: "+window.screen.height+" pixeles<br>");
document.write("Ancho disponible: "+window.screen.availWidth+" pixeles<br>");
document.write("Altura disponible: "+window.screen.availHeight+" pixeles<br>");
document.write("Profundidad de color: "+window.screen.colorDepth+" pixeles<br>");
document.write("Profundidad de pixel: "+window.screen.pixelDepth+" pixeles<br>");
```

10.6. BOM: El objeto location para manejar la URL de la página actual

La propiedad location del objeto window contiene información sobre el URL completo de un documento actual a diferencia de la propiedad location del objeto document que se encarga de cargar un nuevo documento.

```
window.location.propiedad
```

Propiedades del objeto location.

- protocol. Especifica el inicio de la dirección.
- hash. Especifica el nombre del enlace en la URL.
- host. Determina el nombre del servidor y el puerto.
- href. Especifica la dirección completa.
- port. Especifica el puerto de comunicaciones.

```
<!DOCTYPE html>
<html>
<head>
	<title>BOM | Location</title>
	<meta charset="utf-8">
	<script>
		document.write("URL: "+window.location.href+"<br>");
```

```
            document.write("Hostname: "+location.hostname+"<br>");
            document.write("Ruta: "+location.pathname+"<br>");
            document.write("Protocolo: "+location.protocol+"<br>");
            document.write("Puerto: "+location.port+"<br>");
            function salta(){
                location.assign("http://www.google.com");
            }
        </script>
</head>
<body>
    <input type="button" value="Ir a google" onclick="salta()" />
</body>
</html>
```

10.7. BOM: El objeto History

Este objeto contiene información sobre los enlaces que el usuario ha visitado. Su utilidad más aparente es la de generar botones de avance y retroceso.

- back(). Carga el URL anterior al actual.
- forward(). Carga el URL siguiente de la lista.
- go(). Muestra un URL de la lista history según un valor índice introducido.

```
<!DOCTYPE html>
<html>
<head>
    <title>BOM | History</title>
    <meta charset="utf-8">
    <script>
        function atras(){
            window.history.back();
        }
        function adelante(){
```

```
                window.history.forward();
            }
        </script>
 </head>
<body>
<input type="button" value="Atrás" onclick="atras()">
<input type="button" value="Adelante" onclick="adelante()">
</body>
</html>
```

10.8. BOM: El objeto navigator

```
document.write("Galletas activas: "+navigator.cookieEnabled+"<br>");
document.write("La app del navegador: "+navigator.appName+"<br>");
document.write("La app code name: "+navigator.appCodeName+"<br>");
document.write("El motor del navegador: "+navigator.product+"<br>");
document.write("Versión del navegador: "+navigator.appVersion+"<br>");
document.write("User Agent: "+navigator.userAgent+"<br>");
document.write("Plataforma: "+navigator.platform+"<br>");
document.write("Lenguaje: "+navigator.language+"<br>");
document.write("En linea: "+navigator.onLine+"<br>");
document.write("Java activo: "+navigator.javaEnabled())
```

10.9. BOM: manejo del tiempo con setTimeout() y setInterval()

Manejo del reloj de la computadora con setinterval()

```
<!DOCTYPE html>
<html>
<head>
        <title>BOM | Manejo de tiempo</title>
```

```
        <meta charset="utf-8">
        <script>
        var llave = setInterval(reloj,1000);
        function reloj(){
                var d = new Date();
                document.getElementById("salida").innerHTML = d.toLocaleTimeString();
        }
         </script>
 </head>
<body>
<p id="salida"></p>
<button onclick="clearInterval(llave);">Detener</button>
</body>
</html>
```

Manejo de tiempo son setTimeout()

```
<!DOCTYPE html>
<html>
<head>
        <title>BOM | Manejo de tiempo</title>
        <meta charset="utf-8">
        <script>
        function iniciar(){
                alert("Hola desde JavaScript");
        }
         </script>
 </head>
<body>
<input type="button" value="Iniciar" onclick="llave = setTimeout(iniciar,3000);">
<input type="button" value="Detener" onclick="clearTimeout(llave);">
```

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

10.10. BOM: Crear y leer galletas o cookies en JavaScript

```
<!DOCTYPE html>
<html>
<head>
	<title>BOM | cookies</title>
	<meta charset="utf-8">
	<script>
	function leeGalleta(galleta){
		var usuario = galleta +"=";
		var galletaLimpia = decodeURIComponent(document.cookie);
		var ca = galletaLimpia.split(";");
		for (var i = 0; i < ca.length; i++) {
			c = ca[i];
			while(c.charAt(0)==" "){
				c = c.substring(1);
			}
			if(c.indexOf(usuario)==0){
				return c.substring(usuario.length, c.length);
			}
			return "";
		}
	}
	function guardaGalleta(nombreGalleta, valorGalleta, dias){
		console.log(nombreGalleta, valorGalleta, dias);
		var d = new Date();
		d.setTime(d.getTime()+(dias*24*60*60*1000));
		var expira = "expires="+d.toGMTString();
```

```
            var galleta = nombreGalleta+"="+valorGalleta+";"+expira+";path=/";
            console.log(galleta);
            document.cookie = galleta;
        }
        window.onload = function(){
            var usuario = leeGalleta("usuario");
            if (usuario=="") {
                usuario = prompt("¿Cuál es tu nombre?");
                if (usuario!="" && usuario!=null) {
                    guardaGalleta("usuario",usuario,3);
                }
            } else {
                alert("Bienvenido "+usuario+" a nuestra página");
            }
        }
         </script>
 </head>
<body>
</body>
</html>
```

10.11. El objeto document

El objeto *document* hace referencia a todo el contenido de un documento HTML.

Todas las propiedades de este objeto hacen referencia a determinadas características de la página, como su color de fondo, su título, etc. A continuación se relacionan algunas de las más utilizadas:

- bgColor. Color del fondo.
- fgColor. Color del texto.
- vlinkColor. Color de los enlaces visitados.
- alinkColor. Color del enlace en el momento de la selección.

También podemos trabajar con algunos de sus métodos para controlar el proceso de abrir y cerrar un documento.

- clear(). Borra la página del navegador.
- close(). Cierra el documento.
- write(). Permite escribir en un documento.

11: Manejo de eventos en JavaScript

Objetivo: Al finalizar la unidad el participante comprenderá el uso de los eventos en JavaScript

11.1. Los eventos en JavaScript

Se llama evento a un suceso que ocurre en el sistema cuando un usuario ejecuta algún tipo de acción. Un evento debe asociarse a un elemento HTML, no a un código JavaScript.

Script que utiliza el evento onClick:

```
<form><input type="button" value="Pulsa" onClick="alert('Has pulsado el boton');"></form>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Eventos | onload y oncick</title>
	<meta charset="utf-8">
	<script>
	window.onload = function(){
		alert("Bienvenid@ a nuestra página");
		document.getElementById("color").onclick = function(){
			var bodys = document.getElementsByTagName("body");
			bodys[0].setAttribute("class","azul");
		}
	}
	 </script>
	<style>
		.azul{background-color:blue;}
```

```
        </style>
 </head>
<body>
<p id="color">Cambiar de color</p>
</body>
</html>
```

11.2. Los eventos onMouseOver y onMouseOut

```
<!DOCTYPE html>
<html>
<head>
        <title>Eventos | onmouseover onmouseout</title>
        <meta charset="utf-8">
        <script>
                var atenea, miBody;
                window.onload = function(){
                        atenea = document.getElementById("atenea");
                        miBody = document.getElementsByTagName("body");

                        atenea.onclick = function(){
                                alert("Atenea, diosa de la sabiduría");
                        }

                        atenea.onmouseover = function(){
                                miBody[0].setAttribute("class","rojo");
                        }

                        atenea.onmouseout = function(){
                                miBody[0].setAttribute("class","blanco");
                        }
```

```
                }
        </script>
        <style>
                .rojo{ background-color: red; }
                .blanco{ background-color: white; }
        </style>
</head>
<body>
<img src="imagenes/atenea.jpg" id="atenea"/>
</body>
</html>
```

11.3. El evento onSelect

Este evento hace que se ejecute un script cuando se selecciona texto dentro de un elemento de un formulario, ya sea una casilla de texto o de área de texto. Su sintaxis:

```
<input type="tipo de elemento" onSelect="funcion">
```

Ejemplo:

```
<html>
<head>
<title>Ejemplo de evento onSelect</title>
</head>
<body>
<form>
<textarea rows=3 COLS=40 onSelect="alert('Se ha seleccionado texto\nen el
formulario');">
Aquí hay un poco de texto para efectuar la prueba. Seleccione con el cursor un
fragmento.</textarea>
```

```
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Eventos | onselect</title>
	<meta charset="utf-8">
	<script>
	var cajas, miBody;
	window.onload = function(){
		cajas = document.getElementsByTagName("input");
		miBody = document.getElementsByTagName("body");

		cajas[0].onselect = function(){
			miBody[0].setAttribute("class","rojo");
		}

		cajas[1].onselect = function(){
			miBody[0].setAttribute("class","verde");
		}

		cajas[2].onselect = function(){
			miBody[0].setAttribute("class","azul");
		}

		cajas[3].onselect = function(){
			miBody[0].setAttribute("class","amarillo");
		}
```

```
        }
         </script>
        <style>
                .rojo{background: red;}
                .azul{background: blue;}
                .verde{background: green;}
                .amarillo{background: yellow;}
        </style>
 </head>
<body>
<p><input type="text" id="rojo" value="rojo"></p>
<p><input type="text" id="verde" value="verde"></p>
<p><input type="text" id="azul" value="azul"></p>
<p><input type="text" id="amarillo" value="amarillo"></p>
</body>
</html>
```

11.4. El evento onkeydown, onkeyup onkeypress

Estos eventos se generan cuando el usuario efectúa alguna acción con las teclas.

- onkeydown. Se ejecuta cuando se pulsa una tecla.
- onkeyup. Se ejecuta cuando se deja de pulsar una tecla.
- onkeypress. Se ejecuta cuando se mantiene pulsada una tecla.

Ejemplo:

```
<html>
<head>
<title>Ejemplo de evento onkeydown</title>
<script language="javascript">
function avisoPulsacion(){
     alert("Has pulsado una tecla");
```

```
}
 </script>
 </head>
<body>
<center><form name="datos">
<p>Escribe un caracter dentro de la casilla de texto<br>
<input type="text" value="Escribe aqui dentro" SIZE="40"
onkeydown="avisoPulsacion()">
</form>
</center></body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Eventos | onkeydown y onkeyup</title>
	<meta charset="utf-8">
	<script>
	var caja;
	window.onload = function(){
		caja = document.getElementById("caja");
		//detectamos el evento onkeyup
		caja.onkeyup = function(){
			var valor = caja.value;
			caja.value = valor.toUpperCase();
			var salida = "";
			for (var i = 0; i < valor.length; i++) {
				salida = valor[i]+salida;
			}
			document.getElementById("salida").innerHTML =
salida.toLowerCase();
```

```
                }
        }
         </script>
 </head>
<body>
<input type="caja" id="caja"/>
<p id="salida"></p>
</body>
</html>
```

11.5. Los eventos onfocus y onblur

El evento onFocus detecta la entrada (focus) en un elemento de un formulario, por ejemplo un text, textarea o window; mientras que onBlur actúa de manera contraría, es decir, detecta la pérdida del focus.

La sintaxis es la siguiente:

```
<input type=”tipo de entrada” onFocus=”funcion”>
<input type=”tipo de entrada” onBlur=”funcion”>
```

Por ejemplo, podemos mostrar un mensaje cuando el usuario salga de algún elemento de un formulario.

```
<html>
<head>
<title>Ejemplo de evento onFocus</title>
 </head>
<body>
<form>
<p>Nombre:<input type="text" name="nombre"><br>
```

```
<p>Apellidos:<input type="text" name="apellido"><br>
<p>Queja:<textarea name="queja" onBlur="alert('Gracias por expresar su
opinion.\nSaludos');"></textarea>
</form>
</body>
</html>
```

La utilización del evento onFocus puede provocar un bucle infinito que bloquee La computadora. Veamos el siguiente fragmento de script:

```
<p>Queja:<textarea name="queja" onFocus="alert('Gracias por expresar su
opinion.\nSaludos');></textarea>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Eventos | onblur y onfocus</title>
	<meta charset="utf-8">
	<script>
	var campo1, campo2;
	window.onload = function() {
		//referencias a los elementos
		campo1 = document.getElementById("campo1");
		campo2 = document.getElementById("campo2");
		//listeners
		campo1.onfocus = function(){
			enciende(this);
		}
		campo1.onblur = function(){
			apaga(this);
		}
```

```
            campo2.onfocus = function(){
                enciende(this);
            }
            campo2.onblur = function(){
                apaga(this);
            }
        }
        function enciende(campo){
            campo.setAttribute("class","enciende")
        }
        function apaga(campo){
            campo.setAttribute("class","apaga")
        }
         </script>
        <style>
            .enciende{background-color: #ff9;}
            .apaga{background-color: #fff;}
        </style>
 </head>
<body>
<p>
        <label for="campo1">Campo1</label>
        <input type="text" name="campo1" id="campo1">
</p>
<p>
        <label for="campo2">Campo2</label>
        <input type="text" name="campo2" id="campo2">
</p>
</body>
</html>
```

11.6. El evento onSubmit

Este evento se encarga de ejecutar un determinado código de JavaScript al realizarse el envío de un formulario.

El ejemplo más claro de utilización de este evento es evitar que un formulario sea enviado si determinadas condiciones no son cumplidas.

Ejemplo: Vamos a diseñar un formulario que verifique que se ha introducido un número entre uno y cien antes de proceder a enviar los datos del formulario. Si la condición no se cumple, aparecerá un mensaje de aviso y se cancelará el envío.

```
<html>
<head>
<title>Ejemplo de evento onSubmit</title>
<script language="javascript">
function verificarDato()
{
  var valor=document.miformulario,numero.value;
  if(valor>0 && valor<100)
  {
    return(true);
  } else {
    alert("ATENCION. El valor introducido no es correcto");
    return(false);
  }
}
 </script>
 </head>
<body>
<form name="miformulario" method="post"
action="mailto:correo@server.com" onSubmit="verificarDato()">
```

```
<p>Introduzca un numero comprendido entre 1 y 100
<input type="text"NAME="numero"><br>
<input type="submit" value="Pulsa aqui">
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Evento | onSubmit</title>
	<meta charset="utf-8">
	<script>
	var forma, clave1, clave2;
	window.onload = function(){
		forma = document.getElementById("forma");
		clave1 = document.getElementById("clave1");
		clave2 = document.getElementById("clave2");

		forma.onsubmit = function(){
			var valor1 = clave1.value;
			var valor2 = clave2.value;
			if (valor1=="" || valor2=="") {
				alert("Las claves de acceso no pueden estar vacías");
			} else if(valor1!=valor2){
				alert("Las claves de acceso no coinciden");
			} else {
				alert("Las claves de acceso son correctas");
				return true;
			}
			return false;
```

```
                }
        }
         </script>
 </head>
<body>
<form id="forma" name="forma" method="post" action="guarda.php">
        <p>
                <label for="clave1">Clave 1</label>
                <input type="password" name="clave1" id="clave1">
        </p>
        <p>
                <label for="clave2">Clave 2</label>
                <input type="password" name="clave2" id="clave2">
        </p>
        <p>
                <input type="submit" name="enviar" id="enviar" value="Enviar">
        </p>
</form>
</body>
</html>
```

11.7. Eventos onmousedown y onmouseup

Estos eventos permiten declarar el uso del ratón:

- **onmousedown**. Evento que se genera cuando el usuario pulsa un botón del ratón.
- **onmousemove**. Evento que se genera cuando el usuario mueve el cursor del ratón.
- **onmouseup**. Evento que se genera cuando se deja de pulsar un botón del ratón.

Estos eventos pertenecen a los objetos Button, document y Link.

El evento **onmouseover()** sucede cada vez que el cursor del ratón para por encima de un elemento de la página, mientras que el evento onMouseOut sucede cuando se deja de seleccionar este elemento. La sintaxis para los respectivos eventos es la siguiente:

```
<a href="URL" onMouseOver="funcion">Enlace</A>
<a href="URL" onMouseOut="funcion">Enlace</A>
```

En el siguiente ejemplo vemos cómo actúan estos dos eventos:

```
<html>
<head>
<title>Ejemplo de evento onMouseOver y onMouseOut</title>
 </head>
<body>
<a href="enlace" onMouseOver="alert('Has pasado por encima del
enlace');">Pasa por encima mio.</A>
<br><br><br><br><br><br>
<a href="enlace"onMouseOut="alert('Estas alejandote del enlace');">Acercate y luego
vete poco a poco.</A>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Eventos | onmusedown onmouseup</title>
	<meta charset="utf-8">
	<script>
	var atenea, miBody;
```

```
        window.onload = function(){
                atenea = document.getElementById("atenea");
                miBody = document.getElementsByTagName("body");
                atenea.onmousedown = function(){
                        miBody[0].setAttribute("class","naranja");
                }
                atenea.onmouseup = function(){
                        miBody[0].setAttribute("class","blanco");
                }
        }
         </script>
        <style>
                .naranja{background-color: orange;}
                .blanco{background-color: white;}
        </style>
 </head>
<body>
<img src="imagenes/atenea.jpg" id="atenea"/>
</body>
</html>
```

11.8. Evento onmousemove()

onmousemove(). Evento que se genera cuando el usuario mueve el cursor del ratón.

```
<!DOCTYPE html>
<html>
<head>
        <title>Evento | onmousemove</title>
        <meta charset="utf-8">
        <script>
                window.onload = function(){
```

```
                document.getElementById("cuadro").onmousemove = function(e){
                        var x = e.clientX;
                        var y = e.clientY;
                        var coordenadas = "Coordenadas ("+x+", "+y+")";
                        document.getElementById("salida").innerHTML = coordenadas;
                }
                document.getElementById("cuadro").onmouseout = function(e){
                        document.getElementById("salida").innerHTML = "";
                }
        }
    </script>
    <style>
        #cuadro{
                width: 199px;
                height: 99px;
                border: 1px solid black;
        }
        body{margin:0;}
    </style>
</head>
<body>
<div id="cuadro"></div>
<p id="salida"></p>
</body>
</html>
```

11.9. Evento onchange() para la etiqueta <select>

El evento ***onchange()*** se produce cuando se modifica el valor de un elemento select, text o textarea en un formulario HTML.

Para que el evento funcione, el usuario tiene que cambiar de elemento en el formulario, es decir, si estamos introduciendo datos en una casilla de texto, el evento onChange no se activará hasta que no pasemos a otra casilla. La sintaxis es la siguiente:

```
<input type=”tipo de elemento”onChange=”function”>
```

Sin duda es una interesante opción para validar la entrada de datos en los formularios.

```
<html>
<head>
<title>Ejemplo de evento onChange</title>
 </head>
<body>
<form>
<center><p>Mi nombre es:
<input type="text"NAME="apellido"onChange="alert('Esta casilla
a cambiado su contenido')"><br>
<p>Mis apellidos son:
<input type="text"NAME="apellido"onChange="alert('Y ahora esta otra')"><br>
</form>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
	<title>Evento | onchange()</title>
	<meta charset="utf-8">
	<script>
		window.onload = function() {
			document.getElementById("color").onchange = function(e){
				var color = this.value;
				//alert(color);
				var miBody = document.getElementsByTagName("body");
```

```
                        miBody[0].setAttribute("class",color);
                    }
                }
        </script>
        <style>
                .white{background-color:white;}
                .black{background-color:black;}
                .blue{background-color:blue;}
                .green{background-color:green;}
                .yellow{background-color:yellow;}
                .orange{background-color:orange;}
        </style>
</head>
<body>
        <p>¿Qué color prefieres?</p>
<select id="color">
        <option value="black">Negro</option>
        <option value="white">Blanco</option>
        <option value="blue">Azúl</option>
        <option value="green">Verde</option>
        <option value="yellow">Amarillo</option>
        <option value="orange">Naranja</option>
</select>
</body>
</html>
```

11.10. La propagación de los eventos: burbujeo

```
<!DOCTYPE html>
<html>
<head>
```

```
<title>Burbujeo</title>
<script>
	window.onload = function() {
document.getElementById("caja1").addEventListener("click",function(event) {
				console.log("clic sobre la caja 1",event.target)
		},{capture:true});
document.getElementById("caja2").addEventListener("click",function(event) {
				console.log("clic sobre la caja 2",event.target)
		},{capture:true});
document.getElementById("caja3").addEventListener("click",function(event) {
				console.log("clic sobre la caja 3", event.target);
				//event.stopPropagation();
				//event.stopImmediatePropagation();
		},{capture:true});
}
 </script>
<style>
	#caja1{
		width:400px;
		height: 400px;
		background:orange;
	}
	#caja2{
		width:300px;
		height: 300px;
		background:yellow;
	}
	#caja3{
		width:200px;
		height: 200px;
		background:red;
```

```
                }
        </style>
 </head>
<body>
<div id="caja1">
        <p>Caja 1</p>
        <div id="caja2">
                <p>Caja 2</p>
                <div id="caja3">
                        <p>Caja 3</p>
                </div>
        </div>
</div>
</body>
</html>
```

11.11. Los eventos onload y onunload (*)

El evento onLoad es aquel que se produce cuando un navegador carga un documento HTML o una imagen. Este evento se utilizará dentro de la etiqueta <body> del documento HTML o dentro de la etiqueta <FRAMESET> en el supuesto de estar trabajando con frames.

Su sintaxis para cada una de las opciones será la siguiente:

```
<body onLoad=”script a ejecutar”>
```

OnUnLoad es el evento opuesto a onLoad. Su misión consiste en ejecutar un script cuando la página web actual se descarga, ya sea porque se accede a otro web o bien porque se utilizan los botones de avanzar y retroceder.

La sintaxis es:

```
<body onUnLoad=”script a ejecutar”>
```

A continuación un script que muestra un mensaje al cargar un documento HTML y otro al abandonar dicho documento:

```
<html>
<head>
<title>Ejemplo de evento onLoad</title>
 </head>
<body onLoad="alert('Acabas de entrar en el web mas interesante de la RED');”
onUnLoad="alert('Gracias por tu visita. Saludos');">
<center><H1>BIENVENIDOS A MI WEB</H1></center>
</body>
</html>
```

11.12. El evento onReset (*)

El evento onReset se produce cuando se pulsa un botón de Reset en un formulario.

Sintaxis:

```
<input type="reset" onReset="funcion">
```

Este evento puede servir, para advertir al usuario antes de efectuar el ***reset*** del formulario.

```
<html>
<head>
<title>Ejemplo de evento onReset</title>
<script language="javascript">
```

```
function avisoreset(){
      if(confirm("!ATENCION!. Los datos del formulario se van a borrar.")){
         document.datos.reset()
      }
}
 </script>
 </head>
<body>
<center><FORM NAME="datos">
<p>Introduzca sus datos. Gracias.<br>
Nombre:<input type="text"value="Nombre"><br>
Apellidos:<input type="text"value="Apellidos"><br>
Direccion:<input type="text"value="Direccion"><br>
Provincia:<input type="text"value="Provincia"><br>
<input type="button" value="Resetar" onClick="avisoreset()">
</form>
</center>
</body>
</html>
```

11.13. Los eventos onmove() y onresize()

El evento onmove se genera cuando el usuario mueve una ventana o frame de la pantalla principal del navegador.

Cuando un usuario modifica el tamaño de la ventana o del frame actual, se genera un evento onresize.

Estos eventos están incorporados en los objetos window y frame.

```
<html>
<head>
```

```
<title>Ejemplo de evento onmove y onresize</title>
 </head>
<body onresize="alert('La ventana esta siendo redimensionada');"
onmove="alert('La ventana se esta moviendo');">
</body>
</html>
```

Índice