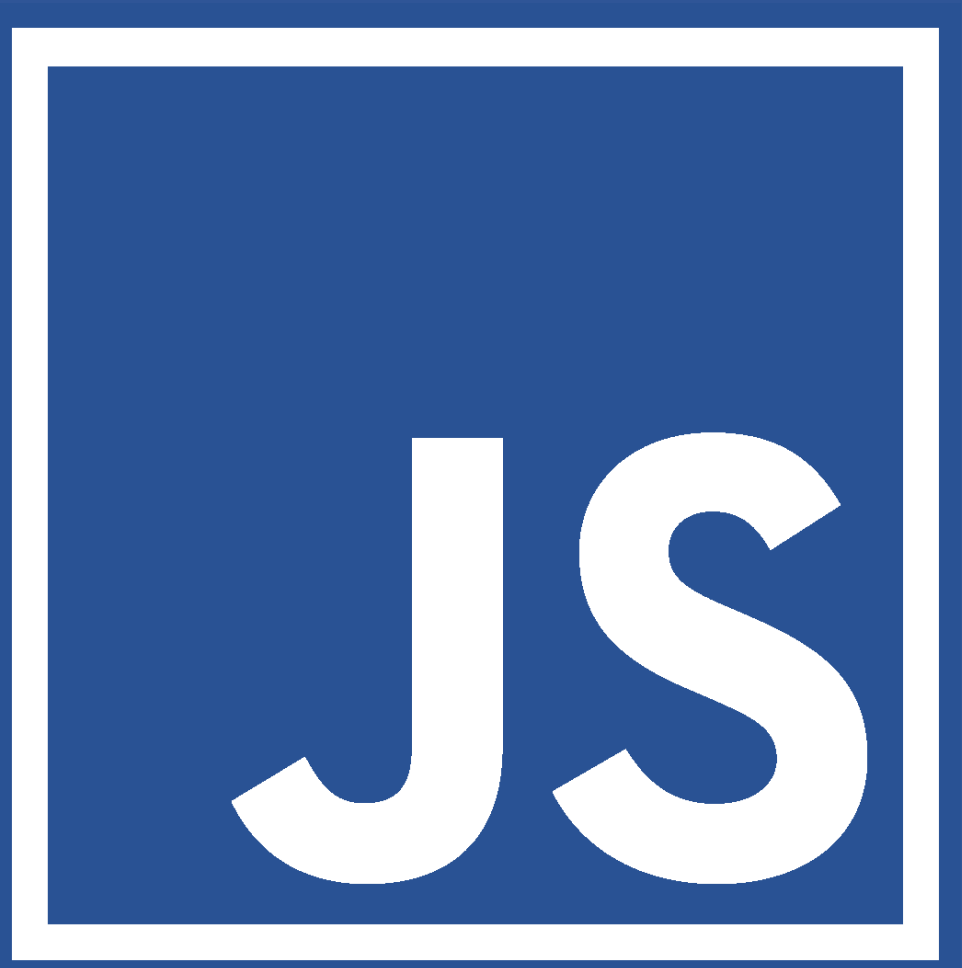


JavaScript

Conceptos básicos

Profesor Vladimir Bataller

Características

JavaScript es un lenguaje interpretado que sigue el estándar ECMAScript. Hoy en día todos los navegadores tienen capacidad de interpretarlo, pero, además existen otros motores interpretadores de JavaScript como son Node.js y Nashorn.

- Permite programación estructurada y/o orientada a objetos. Tiene sintaxis muy parecida a Java.
- Las variables se declaran sin especificar el tipo de datos que contendrán (tipado dinámico).
- Javascript se ejecuta en un único hilo principal y la respuesta a operaciones de entrada-salida y procesos lentos se resuelven de forma asíncrona.
- Actualmente casi todos los navegadores ya implementan ECMAScript 6 (También llamada ECMAScript 2015).

Etiqueta <script> en HTML5

En HTML5, con poner la etiqueta <script></script> sin atributos es suficiente.

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8"/>
        <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
        <script>
                ...
        </script>
</head>
<body>
</body>
</html>
```

Etiqueta <script> en XHTML

Los bloques CDATA permiten indicar al interprete de xhtml que considere el contenido como texto plano a la hora de validar el documento.

type = "tipo mime" (Indica el formato en el que está codificado el contenido, en este caso será "text/JavaScript").

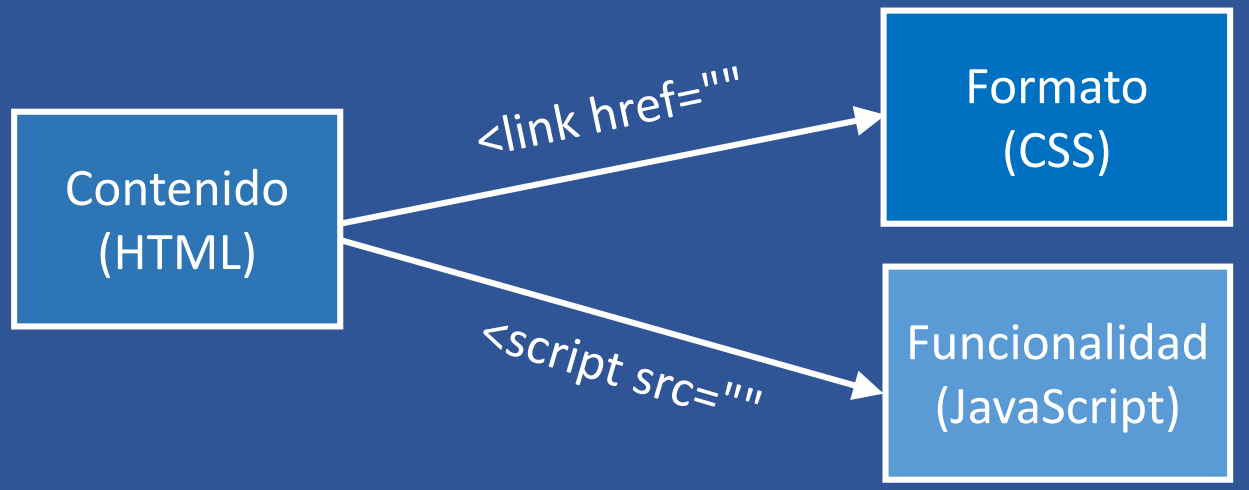
```
<script type="text/JavaScript">
        //<![CDATA[
                document.write("Hola desde Javascript");
        //]]>
</script>
```

Conceptos básicos

Profesor Vladimir Bataller

Separación en el cliente

Para estructurar adecuadamente una aplicación cliente, se debe dividir en contenido (HTML), formato (CSS) y funcionalidad (JavasScript).




```
<script type="text/javascript" src="eventos.js"></script>
<link type="text/css" rel="stylesheet" href="estilos.css"/>
```

Reglas sintácticas

Se pueden incluir líneas de comentarios precedidas por dos barras de dividir (//).
Los comentarios de varias líneas se abren con /* y se cierran con */.

-En Javascript se hace distinción entre mayúsculas y minúsculas.

-Las instrucciones terminan en punto y coma ";".

-Los bloques de código se delimitan con llaves "{}";

-Las cadenas de caracteres se encierran entre comillas (simples o dobles).

Variables

Los datos en Javascript se almacenan en variables a las cuales se le asigna un nombre

No es obligatorio declarar las variables, aunque si es recomendable, para ello se emplea la palabra reservada var antepuesta al nombre de la variable declarada.

```
var a;
```

Se pueden declarar varias variables separando sus nombres con comas.

```
var nombre, apellidos, edad;
```

La asignación de valores a una variable se hace con el signo igual, colcando la variable a la izquierda del signo igual y el valor a la derecha.

```
edad =47;
```

Los valores explicitos asignados a una variable se llaman literales.

Los nombres de variables no pueden contener espacios. Pueden empezar en una letra, guión bajo (_) o dólar ($). Además, pueden contener caracteres numéricos, pero no pueden empezar con ellos.

Los nombres de la variables no pueden coincidir con palabras reservadas de javascript.

Tipos de datos

Cuando se declara una variable tienen un tipo indefinido (Undefined). Cuando se le asigna un valor este será de uno de los siguientes tipos.

string: Textos delimitados entre comillas (simples o dobles).

number: Valores numéricos (enteros o decimales).

boolean: Valores lógicos (true o false).

object: Objetos

undefined: Cuando una variable no ha sido inicializada.

Para conocer el tipo de un dato contenido en una variable disponemos del operador ***typeof*** que retorna una cadena indicando el tipo.

Un objeto nulo se indica con la palabra reservada null.

Ejemplo:

```
var apellidos = "García"; alert (typeof(apellidos));
var edad = 22;
alert(apellidos);
alert(edad);
```

Uso de funciones

JavaScript dispone de funciones predefinidas que podemos emplear.

Para indicar la sintaxis de una función en JavaScript se usa la siguiente convención.

1.- Se antepone al nombre de la función el tipo de dato que retorna como resultado la función (si es que retorna alguno).

2.- Se ponen entre paréntesis las variables que recibe. Si recibe varios valores, se separan con comas.

Una función declarada así:

string ***prompt***(mensaje)

Se emplearía así:

```
var nombre = prompt("Como te llamas");
```

Cuadros de dialogo

Las funciones alert, prompt y confirm nos permiten lanzar pequeñas ventanas para comunicarnos con el usuario.

alert (String mensaje)

muestra el mensaje pasado en forma de String en una ventana de dialogo emergente.

String prompt(String mensaje, valorPorDefecto)

Muestra el mensaje y presenta un campo en el que el usuario puede insertar un valor texto que será el que retorne esta función en forma de String. Si se pulsa cancelar, se retorna null.

boolean confirm(String mensaje)

Si el usuario pulsa *aceptar* se retorna ***true***, si pulsa *cancelar*, se retornará ***false***;

Ejemplo:

```
var provincia = prompt ("Inserte su provincia");
alert ("Usted es de " + provincia);
var confirmacion = confirm("¿Quiere continuar?");
alert("La respuesta es: "+confirmación);
```

Conversión de tipos

Cuando se hacen operaciones entre tipos de datos diferentes, Javascript hace una conversión implícita. Sin embargo en ocasiones puede interesar forzar la conversión.

number parseFloat (String texto)

Convierte el texto pasado como parámetro en un número con parte entera y decimal.

number parseInt (String texto, Number base)

Retorna el número entero (sin parte decimal) correspondiente al texto suministrado. Se le puede agregar un segundo argumento que será un número indicativo de la base de numeración en la que está el número.

string toString(Number base)

Se aplica sobre un número con el operador punto, como argumento se le indica la base y retorna la representación en forma de String del número.

Ejemplo:

```
var texto = (13).toString(2); // Retorna el número 13 en binario
alert( "El 13 en binario es " + texto);
```

Nota: *parseFloat* y *parseInt*, si no reciben un *string* con un número dentro no producen excepción, retornan un *NaN* (not-a-number). Entonces se puede usar la función *isNaN*(variable) para saber si la variable contiene un *NaN*.

Operadores Aritméticos

Cuando ambos operandos son números y el resultado obtenido será también número.

- suma (+)
- resta (-)
- multiplicación (*)
- división (/)
- resto de la división (%)

Operadores Aritméticos y de asignación

Son operadores que abrevian una asignación y una operación aritmética.

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Operadores Unarios

Unarios de signo

más (+) y menos (-) -> Mantiene o cambia el signo de la variable.

```
var a = 3;
var b = -a; // b = -3;
var c = -b; // c = 3;
```

Unarios incrementales

incremento (++) y decremento (--).

```
var a = 3;
a ++; // a=4;
```

Concatenación de cadenas

El operador + permite concatenar literales y variables de cadena de caracteres.

Concatenación de cadenas linerales de caracteres (+)

```
miTexto = "Hola" + " Adios";      // miTexto = "Hola Adios";
```

Concatenación de cadenas variables o mixtas de caracteres (+)

```
var b = “Hola”
miTexto = b + " Adios"; // miTexto = "Hola Adios";
```

Operadores Relacionales

Reciben dos números y retornan true o false para indicar si la relación se cumple o no.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Operadores lógicos

Reciben dos valores lógicos (true o false) y retornan un valor lógico.

Op.	Nombre	Utilización	El resultado es true
&&	AND	op1 && op2	si op1 y op2 son true.
||	OR	op1 || op2	si op1 u op2 son true.
!	negación	! op	si op es false y false si op es true
&	AND	op1 & op2	si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	si op1 u op2 son true. Siempre se evalúa op2

Operadores bit a bit

Operadores de manipulación de bits: Los operandos son variables numéricas que se operan bit a bit. El resultado también es un número.

Op	Utilización	Resultado
>>	op1 >> op2	Desplaza los bits de op1 a la derecha una distancia op2
<<	op1 << op2	Desplaza los bits de op1 a la izquierda una distancia op2
>>>	op1 >>> op2	Desplaza derecha (positiva)
&	op1 & op2	Operador AND a nivel de bits
|	op1 | op2	Operador OR a nivel de bits
^	op1 ^ op2	Operador XOR a nivel de bits
~	~op2	Operador complemento (invierte el valor de cada bit)

Funciones

Permiten agrupar instrucciones para poder ser ejecutadas desde otras partes del código.

Las funciones pueden recibir varios valores como argumentos. La definición de una función tiene la siguiente estructura:

```
function nombre_funcion(argumento1, argumento2, ... argumentoN) {
        instrucciones;
}
```

Para ejecutar la función escribiremos su nombre seguido de los parentesis con los valores para los argumentos dentro.

```
nombre_funcion(valor1, valor2,  ... valor n);
```

Las funciones se suelen declarar en la sección *<head>* del documento o en un archivo js de código.

Variables globales

Son variables definidas fuera de las funciones que están accesibles en todo el document. Las variables globales se suelen declarar en la sección <head> del documento.

```
<head> <title id="t"></title>
<script>
        var titulo = "Pagina principal";
        document.getElementById("t").innerHTML = titulo;
        function getTitulo (){ return titulo;}
</ script>
</head>
<body>
        <h1 id="h"></h1>
        < script>
                document.getElementById("h").innerHTML = titulo;// getTitulo();
        </script >
</body>
```

Archivos externos

En Javascript podemos emplear funciones definidas en archivos externos. Para ello emplearemos el atributo src de la etiqueta script

Para emplear un archivo externo en una página incluiremos:

```
<script type="text/javascript" src="biblioteca.js">
</script>
```

En el archivo externo se escribirá el código JavaScript sin ningún tipo de etiquetas HTML.

Nota: El elemento script siempre debe llevar etiqueta de apertura y cierre. Para importar archivos js también.

Trabajo con números (I)

boolean isNaN(valor)

Devuelve true sólo si el argumento es NaN (si no es un número).

boolean isFinite(numero)

Devuelve true si el número es un número válido (finito).

Notación de literales

Los literales se pueden expresar de la forma 3.453e7 que equivale a 3.453*10^7.

Se pueden indicar literales en hexadecimal precedidos de 0x: 0x78DF.

Se pueden indicar literales en octal precedidos de cero: 07372

Trabajo con números (II)

String toString(base)

- Retorna la representación en formato String del número al que se aplica el método, expresado en la base indicada como argumento.
- Ejemplo:

```
var numero = 31;
var hexa = numero.toString(16);
alert(numero +" en hexadecimal es "+ hexa); // 1F
```

Objetos

Los objetos son entidades de programación que albergan varias variables para guardar información que tiene cierta relación semántica. Además incluyen funciones (llamadas métodos) que albergan el comportamiento del objeto.

REFERENCIAS OBJETOS (De la clase persona) = Instancia (ejemplar) de Persona

p1

Persona

dni: 83882
nombre: Pepe
apellidos:García
saldo: 200

p2

Persona

dni: 35434
nombre: María
apellidos: González
saldo: 450

PUNTO ESCRITURA

p2.apellidos = "González";

REFERENCIA: Variable que apunta a un objeto

ATRIBUTO o PROPIEDAD: Variable que apunta un dato dentro del objeto

Valor

LECTURA

alert(p2.apellidos)

Objetos

JavaScript proporciona objetos para representar diferentes funcionalidades.

propiedades

Los objetos tienen variables internas las cuales pueden ser accedidas mediante el nombre del objeto, un punto y el nombre de la propiedad.

métodos

Son funciones asociadas al objeto. Se ejecutan poniendo el nombre del objeto, un punto y el nombre del método seguido de los argumentos entre paréntesis. Si no tiene argumentos se ponen los paréntesis vacíos.

Ejemplo:

```
window.width=300; //  Propiedad que establece a 300 píxel el ancho de la ventana del navegador
window.close(); // Método que cierra la ventana actual del navegador.
```

Bifurcaciones If

Permiten ejecutar un bloque de sentencias si se cumple una condición.

Bifurcación If simple

```
if (condición) {
        sentencias;
}
```

Bifurcación if-else

```
if (condición) {
        sentencias;
} else {
        sentencias;
}
```

Bifurcaciones else-if concatenados

En la sección else, se puede agregar otro if. De este modo se van ejecutando condiciones en cascada

Bifurcación if-else

```
if (condición1) {
        sentencias; // Si se cuplé 1.
} else if (condición2) {
        sentencias; // Si no se cumple la 1 y se cumple 2.
} else if (condición3) {
        sentencias; // Si no se cumple ni 1,  ni 2 y se cumple 3.
} else {
        sentencias; // Si no se cumple ni 1 , ni 2, ni 3.
}
```

Bifurcaciones Switch

Permiten ejecutar diferentes bloques de sentencias en función de los valores que retorne una expresión.

```
switch (expresión)
{
        case valor1:
                sentencia 1;
                break;
        case valor2:
                sentencia 2;
                break;
        default:
                sentencia 3;
}
```

Bucles for

```
for (inicialización; condición; incremento)
        {    sentencias;    }
```

Inicialización: Es una o varias sentencias (separadas por comas) que se ejecutan una sola vez, cuando se entra en el bucle. (Generalmente consiste en la inicialización de un contador. var i=0).

Condición: Es la condición que se debe cumplir para ejecutar el bucle en cada repetición. (Generalmente es una comparación del contador con el valor límite. i<100).

Incremento: Es una o varias sentencias (separadas por comas) que se ejecutan en cada repetición. (Generalmente consiste en el incremento del contador. i++).

Permiten ejecutar un bloque de sentencias un determinado numero de veces.

```
for(var i=0; i<100; i++){
        document.getElementById(“resul”).innerHTML = i + "<br/>";
}
```

Bucles do ... while

Permiten ejecutar un bloque de sentencias una o más veces mientras se cumpla una condición de permanencia.

```
do {
        sentencias;
} while (condición);    // observese que termina en punto y coma.
```

La condición de permanencia se evalúa tras ejecutar el bloque de sentencias.

Bucles while

Permiten ejecutar un bloque de sentencias cero o más veces mientras se cumpla una condición de permanencia.

```
while (condición)
{
        sentencias;
}
```

Si al entrar en el bucle, no se cumple la condición, las sentencias no se ejecutan ninguna vez.

Ejemplo:

```
while (password!="Alejandría"){
        password = prompt("Inserte la contraseña");
}
alert("Bienvenido!!!");
```

Sentencias break y continue

Permiten alterar la ejecución normal de bucles y bifurcaciones.

La sentencia ***break*** es válida tanto para las bifurcaciones como para los bucles. Hace que se salga inmediatamente del bucle o bloque que se está ejecutando, sin realizar la ejecución del resto de las sentencias.

La sentencia ***continue*** se utiliza en los bucles (no en bifurcaciones). Finaliza la iteración "i" que en ese momento se está ejecutando (no ejecuta el resto de sentencias que hubiera hasta el final del bucle). Vuelve al comienzo del bucle y comienza la siguiente iteración (i+1).

Objetos String (I)

Los String se pueden considerar objetos sobre los cuales se puede ejecutar los siguientes métodos.

charAt(pos), charCodeAt(pos)

Devuelven el carácter o el código numérico del carácter en la cadena.

indexOf(subcadena)

Devuelven la posición (del primer carácter) de la subcadena dentro de la cadena, o -1 en caso de no estar.

length

Propiedad que devuelve la longitud de la cadena.

split(separador)

Divide un objeto string en una matriz de cadenas, obtenidas separando la cadena por el carácter separador.

Ejemplo:

```
var cadena = "Navidad,Semana Santa,Verano";
var  matriz= cadena.split(",");
alert(matriz[1]);//Semana Santa
```

Objetos String (II)

Los String se pueden considerar objetos sobre los cuales se puede ejecutar los siguientes métodos.

substr(pos, longitud)

- Devuelven una subcadena de '*longitud*' caracteres de largo empezando en la posición '*pos*'.

substring(i,j)

- Devuelve la subcadena existente entre los índices i y j-1 de la cadena. El primer carácter es el 0.

toLowerCase(), toUpperCase()

- Retona la misma cadena pero en minúsculas o mayúsculas respectivamente.

Ejemplo:

```
cadena = "abcdefghijk";
subcadena = cadena.substring(2,4); // subcadena = "cd";
```

Objetos String (III)

Los String se pueden considerar objetos sobre los cuales se puede ejecutar los siguientes métodos.

String.fromCharCode(cod1, cod2, cod3, ...);

Retorna un string con los caracteres correspondientes a los códigos suministrados.**Ejemplo:**

```
var cadena = String.fromCharCode(65,66,67); // La variable cadena contendrá "ABC"
```

charCodeAt(posicion);

Retorna el código del caracter indicado por posición en la cadena sobre la que se aplica el método. **Ejemplo:**

```
var nombre = "abcdefghij";
var cod = nombre.charCodeAt(2); //Devolverá 99, que es el código de la letra 'c', el tercer carácter.
```

Caracteres especiales

Los caracteres precedidos de \ indican un carácter especial.

Carácter	Significado
\n	Nueva línea
\r	Retorno carro
\t	Tabulador
\'	Comilla simple
\"	Comilla doble
\\	Barra invertida

El objeto Math

El objeto Math pone a nuestra disposición una serie de funciones matemáticas. Para ejecutarlas pondremos el nombre del objeto Math seguido de un punto y del nombre de la función.

abs()	valor absoluto	round()	entero más cercano
cos(), acos()	coseno y arco coseno	pow(,)	el primer argumento elevado al segundo
tan(), atan(), atan2(,)	tangente, arco tangente entre -π/2 y π/2, arco tangente entre -π y π.	exp(), log(), log10()	el número "e" elevado al argumento. Logaritmo neperiano. Logaritmo en base 10.
ceil(), floor()	redondeo por exceso y por defecto.	random()	número aleatorio entre 0 y 1 (sin llegar a valer 1)
sin(), asin()	seno y arco seno	toDegrees()	convierte radianes a grados
sqrt( )	raíz cuadrada.	toRadians()	convierte grados a radianes
E	constante, número E	PI	constante, numero π

Matrices (I)

En JavaScript las matrices son objetos y deben ser declaradas con el operador *new*. seguido del *Array* con el número de elementos entre paréntesis. Otra forma de declarar una matriz es con la notación JSON empleando corchetes [].

Declaración:

```
var m= new Array(4);  // Sintáxis clásica
var m = []; // Sintáxis JSON
```

Asignación y lectura:

```
m[3]="jueves";  // Escritura en un elemento de la matriz.
alert(m[3]); // Lectura de un elemento de la matriz.
```

Indices: Los índices de una matriz empiezan en 0.

Redimensión automática: Si se asigna un valor a un elemento con índice superior al máximo, la matriz se redimensiona para abarcar hasta ese nuevo índice. Los elementos no asignados toman el valor *undefined.*

Inicialización y declaración simultanea:

```
var m = new Array("Lunes", "Martes", "Miércoles", "Jueves", "Viernes"); // Sintáxis clásica
var m = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes"]; // Sintáxis JSON
```

La propiedad **length** retorna el número de elementos de una matriz.

Matrices Asociativas

Los elementos de matrices asociativas son accedidos mediante una clave (un string) en lugar de mediante un indice.

Matrices Asociativas

Cada elemento de la matriz viene determinado por un string, en lugar de por un indice.

```
var m = new Array();
m["lunes"]="Soleado";
m["martes"]="Viento";
alert("Predicción para el martes: " + m["martes"]);
alert("Predicción para el lunes: " + m.lunes);
```

Bucle para recorrer una matríz asociativa

```
for(clave in m)
{document.write( clave + "----&gt; " + m[clave]);}
```

En general, en programación a este tipo de estructuras se les llama Matrices asociativas, diccionarios , mapas o hash array.

Matrices (II)

Las matrices tienen la propiedad **length** que retorna la longitud de la matriz. Así mismo disponen de los siguientes métodos.

String join(String separador)

Retorna un String como resultado de concatenar todos los elementos de la matriz, empleando opcionalmente el separador suministrado.

Array reverse()

Retorna otra matriz con los mismos elementos pero en orden inverso. También invierte los elementos de la matriz sobre la que se aplica.

void push(Object nuevo)

Inserta un nuevo elemento al final de la matriz.

splice(i, n)

Elimina 'n' elementos del array a partir de la posición indicada por 'i'. Los elemento que había detrás de los eliminados ocupan el espacio de estos. Es decir el length del array se reduce en 'n' unidades.

Matrices (III)

Las matrices tienen la propiedad **length** que retorna la longitud de la matriz. Así mismo disponen de los siguientes métodos.

Array sort()

Ordena la matriz y retorna otra con los mismos elementos en orden.

Si se desea un criterio de ordenación distinto al orden alfabético, se puede pasar como parámetro al método sort el nombre de una función de comparación que recibe dos parámetros y retorna negativo, cero o positivo en función de que el primer parámetro sea respectivamente menor, igual o mayor que el segundo.

Array sort(nombreFuncionComparacion) -> El nombre se indica sin comillas ni paréntesis.

Ejemplo:

```
function comparacion(a,b){
        if(a>b)return +1; if(a<b) return –1; return 0;
}
m = m.short(comparacion);
```

Matrices (IV)

Array filter(function)

Retorna una matriz resultante de eliminar de la matriz original todos los elementos que no cumplen cierta condición. Para indicar dicha condición se pasa como argumento una función que se ejecutará para cada elemento de la matriz y segun dicha función retorne true o false, el elemento se incluirá o no en la matriz resultante. Dicha función recibe los siguientes tres argumentos (los dos últimos opcionales0):

- elemento: elemento de la matriz de la iteración actual.
- índice: posición en la matriz original del elemento de la iteración actual.
- matriz: matriz original.

Ejemplo:

```
var m_destino = m_origen.filter(function (elemento, i, matriz){
        return elemento.saldo>100;
});
```

Matrices multidimensionales

Las matrices multidimensionales son matrices de una dimensión donde cada elemento que le asignamos será una matriz.

Declaración de la matríz de filas:

```
var m= new Array(4);
```

Declaración de las matrices de columnas:

```
for (var i =0; i<m.length; i++)
        m[i] = new Array(6);
```

Acceso a los elementos de la matriz:

```
m[3][5] = "azul"; // Último elemento de la última fila.
```

Eventos (I)

Javascript nos permite responder a acciones del usuario como pulsar sobre un botón o pasar el ratón sobre una imagen.

Evento	Descripción	Elementos que lo admiten
onload	Cuando la página HTML (o la Imagen) termina de cargarse en el navegador	<BODY> <IMG>
onunload	Cuando salimos de una página (se descarga)	<BODY>
onmouseover	Al pasar el ratón por encima de un objeto	<A HREF> <AREA>
onmouseout	Cuando el ratón deja de estar encima.	<A HREF> <AREA>
onsubmit	Cuando se envía un formulario.(return true)	<FORM>
onclick	Se produce al pulsar un elemento	<A HREF> <AREA> <INPUT TYPE="button, checkbox, link, radio">
onblur	Cuando se pierde el enfoque del objeto	<INPUT TYPE="button, checkbox, link, radio, text"> <TEXTAREA>

Eventos (II)

Javascript nos permite responder a acciones del usuario como pulsar sobre un botón o pasar el ratón sobre una imagen.

Evento	Descripción	Elementos que lo admiten
onchange	Cuando se pierde el foco de un control cuyo contenido ha cambiado.	<INPUT TYPE="text"> <TEXTAREA><SELECT>
oninput	Cuando cambia el valor de un control.	<INPUT TYPE="text"> <TEXTAREA><SELECT>
onfocus	cuando un elemento recibe el foco	<INPUT TYPE="text, button, checkbox,radio, password, select, frame"><TEXTAREA>
onselect	Al seleccionar parte del contenido de un campo o zona de texto	<INPUT TYPE="text"> <TEXTAREA>
onabort, onerror	Si se aborta o se produce un error en la carga de una imagen	<IMG>

Evento:keypress (III)

El evento onkeypress detecta cuando se ha pulsado una tecla. Si retorna false, se cancela la entrada del caracter. onkeyup y onkeydown también detectan la tecla pulsada, pero no pueden cancelar su escritura

```
function teclapulsada(event)
{
        var codigo= event.keyCode;
        var caracter = String.fromCharCode(codigo);
        console.log("se ha pulsado la tecla",código, "del carácter", caracter);
}
```

Para que la función reciba el event, hay que pasárselo en la asignación del evento:

```
<input type="text" id="txtNombre"  onkeypress="return teclapulsada(event)"/>
```

Eventos (IV)

La asignación de respuesta a un evento se hace suministrando al atributo con el nombre del evento una cadena con las instrucciones a ejecutar.

```
<button onclick="alert('Se ha pulsado el botón');" >Pulsar<button>
```

Validación y envio de formularios

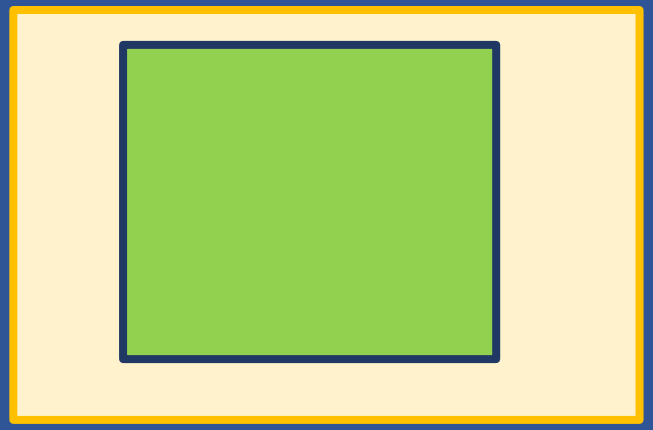
Para validar un formulario antes de enviarlo, en su evento submit, indicamos la función que realiza la validación. si retorna true se envia el formulario. Si retorna false, no lo envia.

```
onsubmit="return validarFormulario();"
```

Modelo de eventos W3C

La propagación de eventos en elementos anidados consta de dos fases: captura y burbuja.

Captura: si están las dos fases, se ejecuta la primera y su sentido de propagación es desde fuera hacia dentro.
Burbuja: es la propagación por defecto y es de dentro hacia fuera.
Si los manejadores de eventos se asignan con las propiedades onxxxxx (p.e: onclick) entonces se propagan en la fase de la burbuja.
Si se desea establecer la fase en la que se propaga un evento se empleará la función **addEventListener** que recibe tres argumentos: **nombreDelEvento, funciónManejadora, fase**.



addEventListener (nombreDelEvento, funciónManejadora, fase)

- **nombreDelEvento**: String con el nombre del evento sin "on". (p.e.: "click").
- **funciónManejadora**: función de gestión de evento. Recibe un objeto event.
- **fase**: opcional y por defecto es *false* (burbuja). Si se asigna *true* el manejador se ejecutará en la fase de captura.

Temporizaciones

Con la instrucción ***setTimeout*** podemos realizar la llamada a una función en diferido.

NúmeroDeTemporizador **setTimeout**(*función*, *milisegundos*)

La función setTimeout la función que se debe ejecutar transcurrido el tiempo indicado en *milisegundos.* Así mismo, retorna un número que identifica la temporización.

clearTimeout (*NúmeroDeTemporizador*)

Detiene el temporizador asociado al número indicado como argumento.

Ejemplo:

```
var temp = setTimeout(saludo,2000);
....
<a href="javascript: clearTimeout(temp);">Parar temporizacion</a>
```

Temporizaciones

Con la instrucción ***setInterval*** podemos realizar la llamada a una función repetidas veces separadas por un tiempo.

NúmeroDeTemporizador **setInterval**(función, milisegundos)

Recibe la función que debe ejecutar transcurrido el tiempo indicado en *milisegundos*. Así mismo, retorna un número que identifica la temporización.

clearInterval (NúmeroDeTemporizador)

Detiene el temporizador asociado al número indicado como argumento.

NúmeroDeTemporizador **setInterval**(funcion, milisegundos, argumentos ....)

setInterval tiene esta forma adicional de ser llamada, en este caso los parámetros se pueden indicar después de los millisegundos.

La clase Date

Los objetos de la clase Date representan un instante temporal.

Declaración

miFecha = new Date(); // Contiene la fecha y hora actual.
miFecha = new Date(año, mes, día);// enero=0
miFecha = new Date(año, mes, día, horas, minutos, segundos);

getTime(), setTime(milisegundos)

Obtiene y establece, respectivamente, la fecha y la hora tomados como milisegundos transcurridos desde el 1 de enero de 1970.

getFullYear(), setFullYear(año)

Obtiene y establece, respectivamente, el año con cuatro dígitos para una fecha determinada.
miFecha.getFullYear();// Retorna el año con 4 digitos

Otras funciones

getHours(), setHours(horas), getMinutes(), setMinutes(minutos), getSeconds(), setSeconds(segundos), getMonth(), setMonth(mes). //enero=0
getDate(), setDate(dia) : Día del mes.
getDay(): Día de la semana(0=Domingo, 6=Sábado).
toLocaleTimeString(), toLocaleDateString().

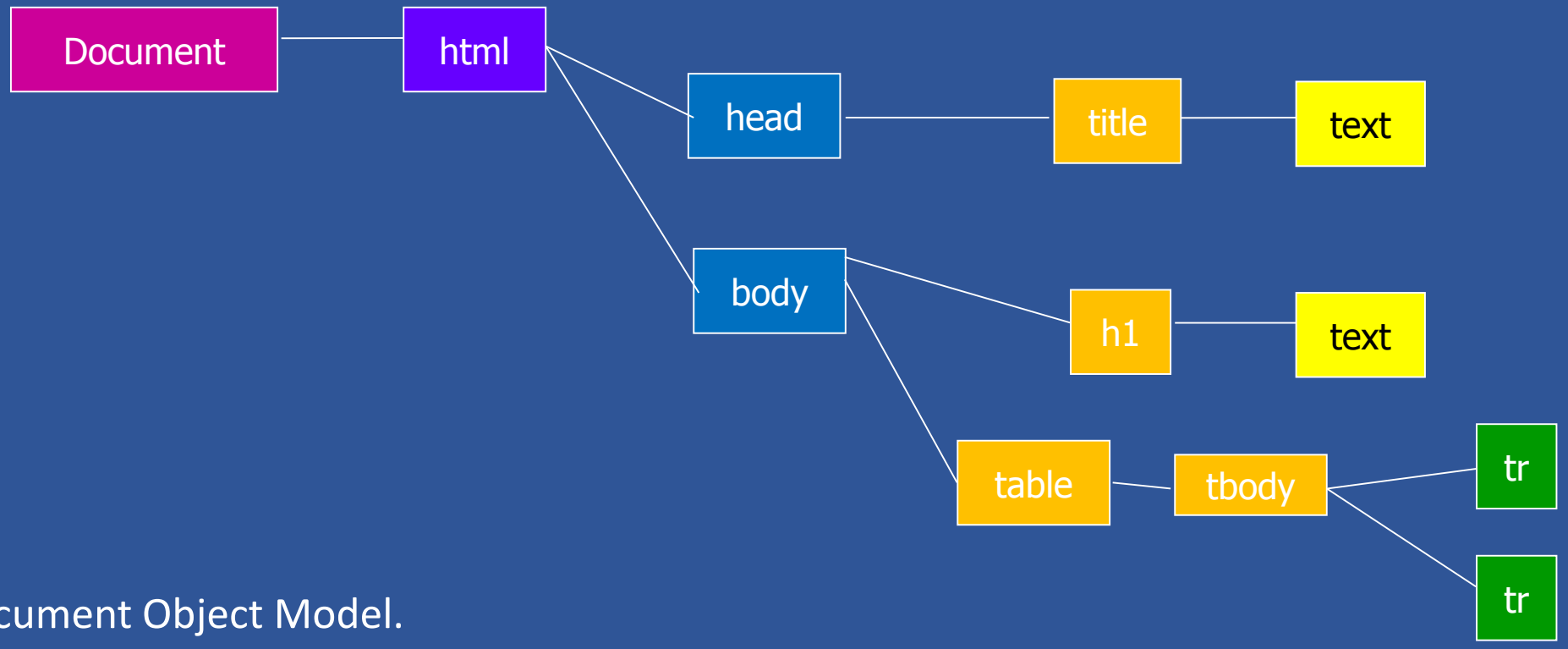
JavaScript - DOM

Profesor Vladimir Bataller

DOM

DOM es una especificación de la w3c que indica como debe ser la representación jerárquica de un documento XML.

DOM define una serie de interfaces con métodos que se llaman igual en todos lo lenguajes de programación.



D.O.M. : Document Object Model.

DOM:Nodos (I)

Propiedades de los Nodos

parentNode: nodo padre.

childNodes: matriz de nodos hijo

firstChild: primer hijo

lastChild: último hijo

id: identificador

className: clase de estilo

tagName y nodeName: Nombre de la etiqueta html

nodeType: 1=Etiqueta; 3=texto; 9= objeto document

previousSibling: hermano anterior

nextSibling: hermano posterior

nodeValue: texto de un nodo de texto

DOM:Nodos (II)

Métodos de los nodos

appendChild(nuevoNodo): Inserta un nodo hijo.

replaceChild(nodoNuevo,nodoAntiguo): reemplaza un hijo por otro.

removeChild(nodoAQuitar): Elimina un nodo hijo.

insertBefore(nodoNuevo, nodoActual): inserta un hijo delante de otro

boolean hasChildNodes(): true si tiene hijos.

String getAttribute("atributo"): Retorna el valor de un atributo.

setAttribute("atributo", "valor"): Establece el valor de una atributo.

Nodo cloneNode(boolean): retorna un nodo igual al que se le aplica el método. Si el boolean es true, copia además los nodos hijos.

DOM: Document (III)

atributos del objeto document.

location

Contiene la dirección o URL completa del documento.

referrer

URL del documento desde el que se archivo el actual.

title

Título de la ventana.

DOM: Document(IV)

métodos de Document relacionados con el Dom.

Nodo getElementById("id"): Retorna el elemento que tiene el id indicado.

Nodo createElement("etiqueta"): Crea un nodo con la etiqueta indicada.

Nodo createTextNode(texto): Crea un nodo de texto.

NodeList getElementsByTagName("etiqueta"): retorna una matriz de nodos que tienen la etiqueta indicada.

Element documentElement: Contiene una referencia a un objeto Element que es el elemento raíz del documento.

NodeList querySelectorAll("selector css"): retorna una matriz de los nodos que cumplen el selector css indicado.

Nodo querySelector("selector css"): retorna el primer elemento que que cumple el selector css indicado.

clear(): Borra el contenido de un documento

DOM: Document (V)

El objeto document representa al documento que se está mostrando en el navegador.

el método write(String texto)

Si se ejecuta mientras se está creando el documento, agregará texto pasado como parámetro al contenido del documento.

Si se ejecuta una vez que se ha creado el documento, se reescribirá todo el contenido del documento con el texto pasado como parámetro.

El texto puede contener etiquetas html.

Ejemplo:

```
if(prompt("Color de fondo:")=="azul")
        document.write("<BODY bgcolor='blue' >");
else
        document.write("<BODY bgcolor='green' >");
```

DOM: Consideraciones

La manipulación del contenido de un documento mediante el DOM es una garantía de compatibilidad ya que algunos navegadores presentan problemas con otras alternativas para consguir lo mismo.

boton.onclick = "borrarLibro("+i+");" --> NO FUNCIONA

Solución: boton.setAttribute("onclick","borrarLibro("+i+");");

tbody.innerHTML = ""; --> NO FUNCIONA

Solución:

```
while(tbody.hasChildNodes())
{tbody.removeChild(tbody.lastChild);}
```

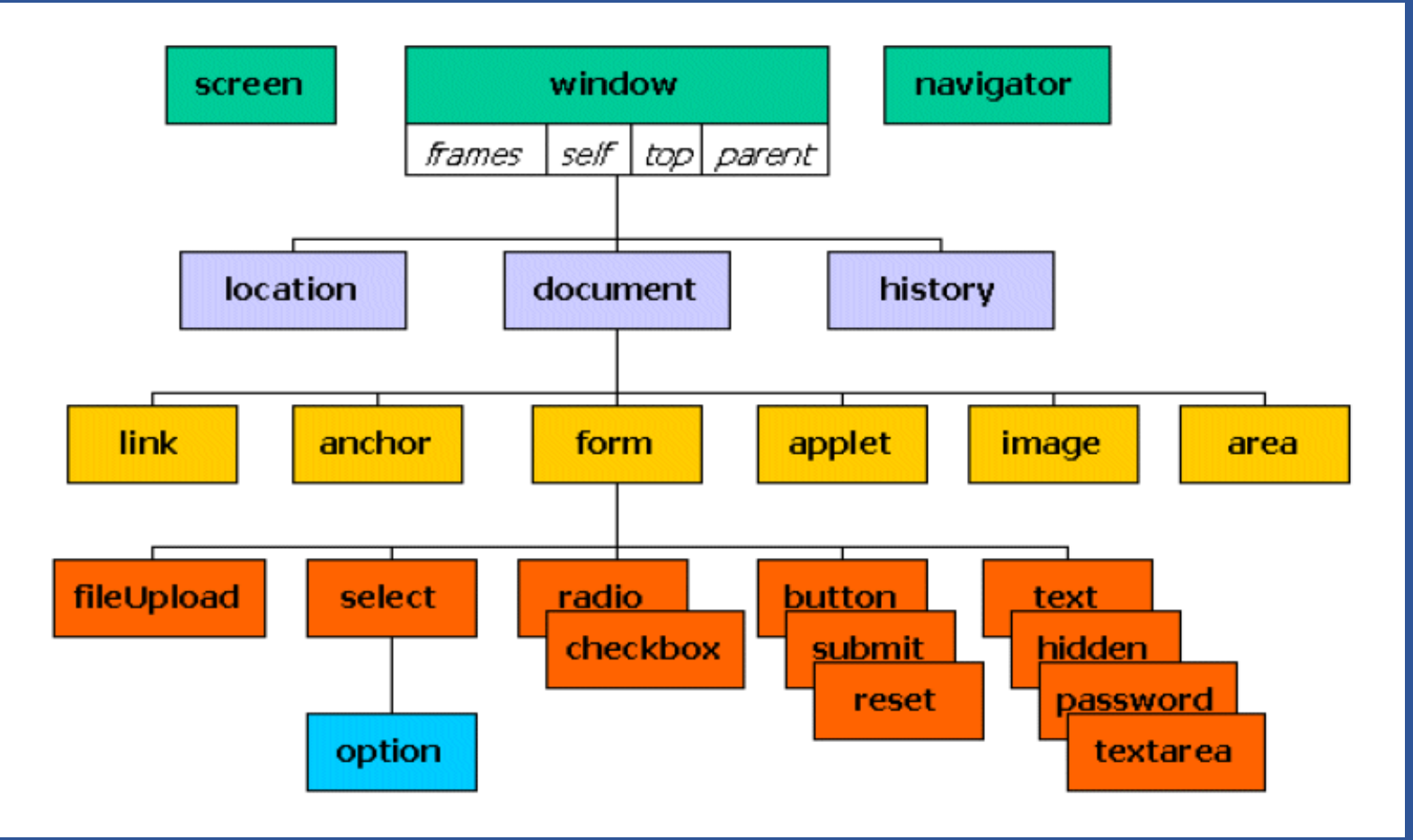
Para insertar algo dentro de un tbody, ocurre lo mismo que para borrarlo, no se puede usar el innerHTML, la forma de modificarlos es mediante objetos DOM. (tbody.appendChild(tr);).

JavaScript - Objetos del navegador

Profesor Vladimir Bataller

Objetos del nevegador

JavaScript proporciona una serie de objetos que representan los elementos del documento y del navegador.



Objetos Window (I)

La variable window representan un objeto de tipo Window asociado a la ventana actual.

Otros objetos Window

Cuando hay varias ventanas del navegador, se puede acceder a ellas como objetos Window. Así mismo una ventana puede contener varios marcos (frames) que pueden ser accedidos mediante la matriz **frames**. Los objetos de esta matriz son también objetos Window.

window como objeto predeterminado

El objeto window que representa a la ventana actual, es el objeto predeterminado por lo tanto cuando se escribe:

```
document.write("Hola");
```

Equivale a escribir:

```
window.document.write("Hola");
```

Objetos Window (II)

Mediante el método open() podemos abrir un documento en otra ventana.

var variable=window.open("URL", "nombre", "propiedades");

Si ya existe una ventana con el nombre indicado, el documento indicado por la URL se abrirá allí, si no existe, se abrirá en una ventana nueva.
Propiedades es una lista separada por comas con los pares:
toolbar[=yes|no] location[=yes|no] fullscreen [=yes|no]
directories[=yes|no] menubar[=yes|no]
scrollbars[=yes|no] status[=yes|no]
resizable[=yes|no] width=pixels height=pixels

Ejemplo:

```
var nuevaventana=window.open("http://www.yahoo.es","yahoo","resizable=no,
width=300,height=150");
```

JavaScript - Objetos del navegador

Profesor Vladimir Bataller

Objetos Window (III)

Propiedades

status

Cadena de caracteres se muestra en la barra de estado.

location

Contiene un objeto con la información del URL actual.

frames

Matriz que contiene todos los marcos de una ventana. Se pueden referenciar por el índice o por el nombre del marco.

parent

Referencia a la ventana dentro de la cual se encuentra la actual.

Ejemplo:

```
alert("Esta página tiene "+ window.frames.length+" marcos");
```

Objetos Window (IV)

Propiedades

opener
Referencia a al ventana/marco desde la cual se ha abierto la actual.

top
Referencia a la ventana de orden superior del navegador (la que contiene a todas las demás)

self
Referencia a la ventana/marco actual.

name
Nombre de la ventana/marco actual.

closed : retorna true si la ventana está cerrada.

Objetos Window (V)

Métodos

focus()
Hace que la ventana reciba el foco.

blur()
Hace que la ventana pierda el foco.

close()
Cierra la ventana.

scroll(x,y)
Mueve el scroll a la posición indicada por *x* e *y*.

moveTo(x,y): Mueve la ventana a las condenadas x e y.

moveBy(incx,incy): Incrementa la posición de la ventana en incx e incy

Ejemplo:

```
alert("Esta página tiene "+ window.frames.length+" marcos");
```

el objeto location

Representa la url de una ventana.

http://www.google.es/buscar?dni=3&nom=pepe

Propiedades

href

Contiene una cadena de texto con la url de la ventana. Si se modifica, se cargará el documento asociado a la nueva url.

hostname (www.google.es)

Nombre del servidor desde el que se ha descargado la página

port (80)

Puerto por el que se ha recibido la página.

protocol (http)

Protocolo por el que se ha transferido la página

pathname: Ruta y nombre del archivo (/buscar)

search: Cadena de parámetros de búsqueda del método get (?dni=3&nom=pepe)

Métodos

reload(): Recarga la página desde su ubicación original.

el objeto screen

Permite conocer la configuración de pantalla del cliente.

height
Altura de la resolución de la pantalla.

width
Anchura de la resolución de la pantalla.

pixelDepth
Número de bits por píxel (resolución de la pantalla).

Ejemplo:

```
texto=screen.width + "x" + screen.height + "x" + Math.pow(2,screen.colorDepth)
        + " colores.";
alert (texto);
```

JavaScript - Objetos del navegador

Profesor Vladimir Bataller

El ViewPort

Como conocer las dimensiones del ancho del area de visualizacion del navegador (ViewPort).

```
var vpw;
var vph;
if (self.innerWidth)
{           vpw = self.innerWidth;      vph = self.innerHeight;}
else if (document.documentElement           &&document.documentElement.clientWidth)
{           vpw = document.documentElement.clientWidth;
            vph = document.documentElement.clientHeight;}
else {                  vpw = document.body.clientWidth;            vph = document.body.clientHeight; }
```

JavaScript - Objetos del navegador

Profesor Vladimir Bataller

Acceso a formularios

Los controles de formulario pueden ser accedidos mediante su id o mediante el nombre del formulario y el nombre del control.

Los controles que reciben un texto tienen la propiedad *value* que equivale al dicho texto.

Los controles Checkbox y Radio tienen también la propiedad *checked* que retorna un booleano para indicar si están seleccionadas.

Los objetos form tienen el método *submit()* para enviarlo al servidor y un evento onsubmit para capturar el envío antes de producirse incluso para evitarlo.

formularios: Text

Los controles de formulario pueden ser accedidos mediante su id o mediante el nombre del formulario y el nombre del control.

Escribir en una caja de texto

```
document.nombreformulario.nombrecajatexto.value ="hola";
document.getElementById("idCajaTexto").value="hola";
```

Habilitar/Desabilitar una caja de texto (o cualquier otro control):

```
document.getElementById("idCajaTexto").disabled=true;
```

formularios: Listas

Las listas (<select>) son objetos que contiene varios objetos Option.

Propiedades de los objetos **select**:

options[ ]-> Matriz que contiene los objetos Option de la lista.

selectedIndex->Indice del elemento seleccionado (el primero es cero)

Eliminar un elemento de un **select**:

```
document.formulario.lista.options[indice]=null;
```

Insertar un nuevo elemente en un **select**:

```
document.formulario.lista.options[indice]= new Option("texto","valor");
```

Limpiar por completo una lista de un **select**:

```
document.formulario.lista.length=0;
```

Los objetos **Option** tienen las siguientes propiedades:

value: valor asociado al elemento.

text: texto que muestra.

selected: booleano que indica si el elemento está seleccionado.

formularios: Validación con HTML5 y JS (I)

HTML5 proporciona nuevos tipos de input y nuevos atributos que facilitan la validación de formularios.

```
<form action="#" >
<p>
Nombre:<input name="nombre" type="text" required="required"/>
<span></span><br/>
Año graduacion:
<input name="graduacion" type="number"   max="2016"  required="required" />
<span></span><br/>
</p>
</form>
```

formularios: Validación con HTML5 y JS (II)

HTML5 proporciona nuevos tipos de input y nuevos atributos que facilitan la validación de formularios.

disabled Indica que el control está deshabilitado.

max Valor máximo de un campo numérico

min Valor mínimo de un campo numérico.

pattern Expresión regular que debe cumplir el campo.

required Indica si el campo es obligatorio

type Indica el tipo de dato esperado. Los tipos que introduce HTML5 son:

color, date, datetime, datetime-local, email, month, number, range, search, tel, time, url, week.

formularios: Validación con HTML5 y JS (III)

checkValidity() indica si un campo es correcto y **validity** con sus propiedades indica que tipo de error tiene.

```
function validarUno(event){ // Asignado al evento key up de cada control de formulario
        var t = event.currentTarget;
        if(t.className.indexOf("dirty")<0) t.className += " dirty";
        var msg = "";
        if(t.checkValidity() == false){     msg = "error";
                if(t.validity.valueMissing)   msg = "Requerido";
                if(t.validity.badInput) msg = "Formato incorrecto";
                else if(t.validity.rangeOverflow){
                        msg = "Valor demasiado alto, el máximo es " + t.max;
                }
        }
        t.nextSibling.innerHTML = msg;
}
```

formularios: Validación con HTML5 y JS (IV)

validity es un objeto **ValidityState** con las siguientes propiedades. No todas las implementan todos los navegadores.

customError	Si se ha especificado un mensaje de validación personalizado.
patternMismatch	Si no cumple un *pattern* especificado.
rangeOverflow	Si es mayor que el valor asignado al atributo *max*.
rangeUnderflow	Si es menor que el valor asignado al atributo *min*.
stepMismatch	Si el valor no coincide con los disponibles segun el *step* indicado.
tooLong	Si la longitud supera la indicada en *maxLength*.
typeMismatch, badInput	Si el valor asignado no cumple el tipo especificado.
valueMissing	Si el campo está vacio y la propieda required está establecida.
valid	Si el campo supera todas las reglas de validación.

formularios: Validación con HTML5 y JS (V)

Además existen una serie de pseudo-clases de estilo que se activan en función del estado de validación de el elemento en cuestión.

Selector css	comportamiento
:disabled	inputs que están deshabilitados (disabled="disabled").
:invalid	inputs que no ha superado las validacines establecidas para ellos.
:required	inputs con el atributo required.
:optional	inputs sin el atributo required.
:valid	inputs que han superado las pruebas de validación.

Ejemplo:

```
input.dirty:invalid{
        background-color:pink;
}
input.dirty:invalid + span{
        background-color:pink;color:white;font-weight:bold;padding:3px 15px;
}
```