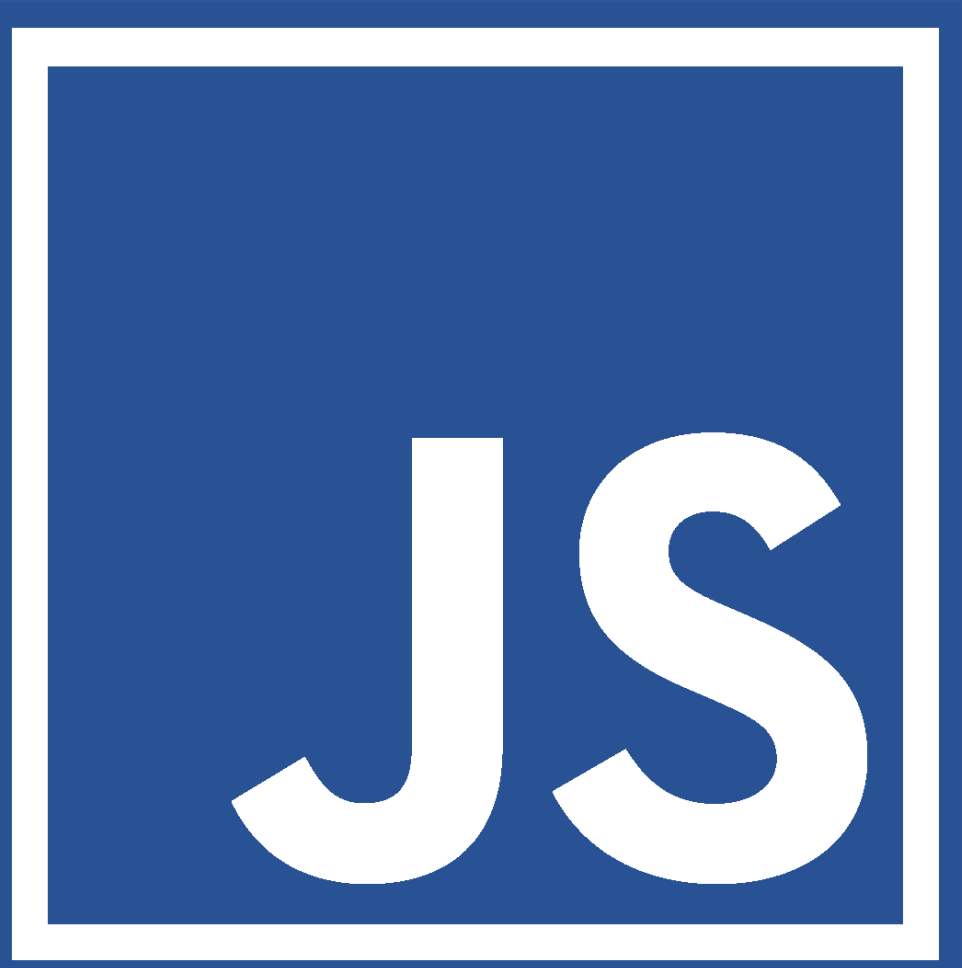


JavaScript

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").

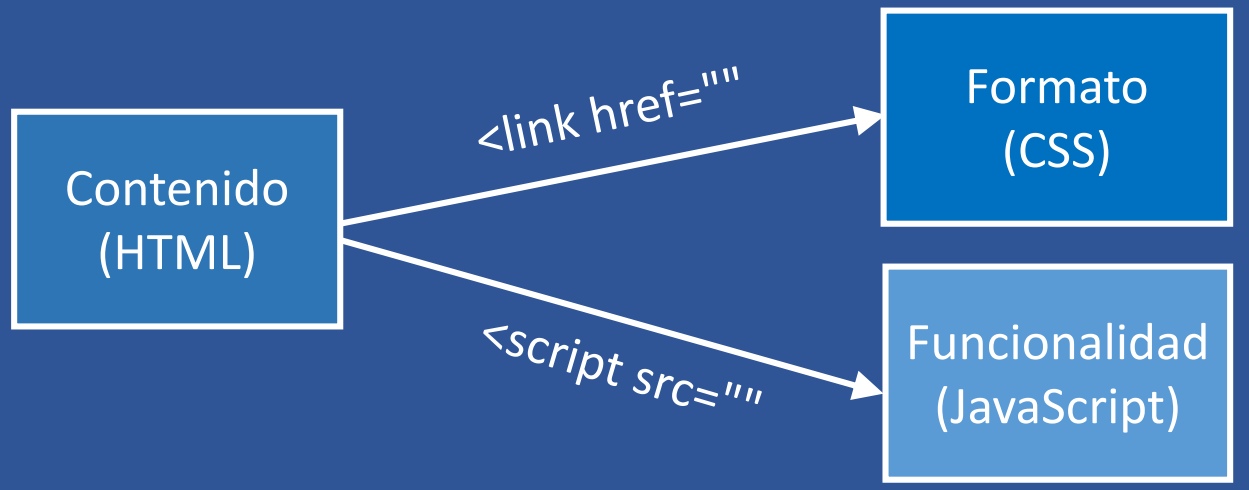
```
<script type="text/JavaScript">
        //<![CDATA[
                document.write("Hola desde Javascript");
        //]]>
</script>
```

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.
```

Expresiones y sentencias

En las estructuras de control del código, empleamos las siguientes definiciones.

Expresiones: Una expresión es un conjunto variables y literales unidos por operadores que retornan un valor.

a + b

Condición: Expresión que retorna un valor booleano (true o false).

a > b

Sentencias: Son las mínimas unidades de ejecución de un programa. En Javascript cada sentencia termina en punto y coma. Se pueden indicar varias sentencias en una misma línea.

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.
}
```

Operador asignación condicional

Evalúa una condición pasada como primer término y retorna el segundo o tercer término, en función de que la condición sea true o false respectivamente.

Sintáxis

condición **?** *valor_si_true* **:** *valor_si_false***;**

Ejemplo

```
var a = 5;
var texto = a%2==0 ? "es par" : "es impar";
console.log(a , texto); // Imprme: a es impar;
```

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.

replace(subStringBuscado, subStringNuevo);

Retorna un string resultante de sustituir el subStringBuscado por el subStringNuevo en todos los puntos del String donde aparezca el subStringBuscado.

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

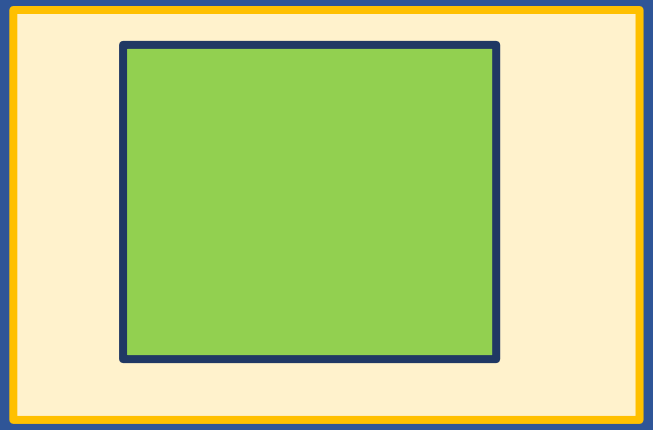
Para validar un formulario antes de enviarlo, en su evento submit, indicamos la función que realiza la validación. si retorna true se envia el formulario. Si retorna false, no lo envia.

```
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.

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().

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.

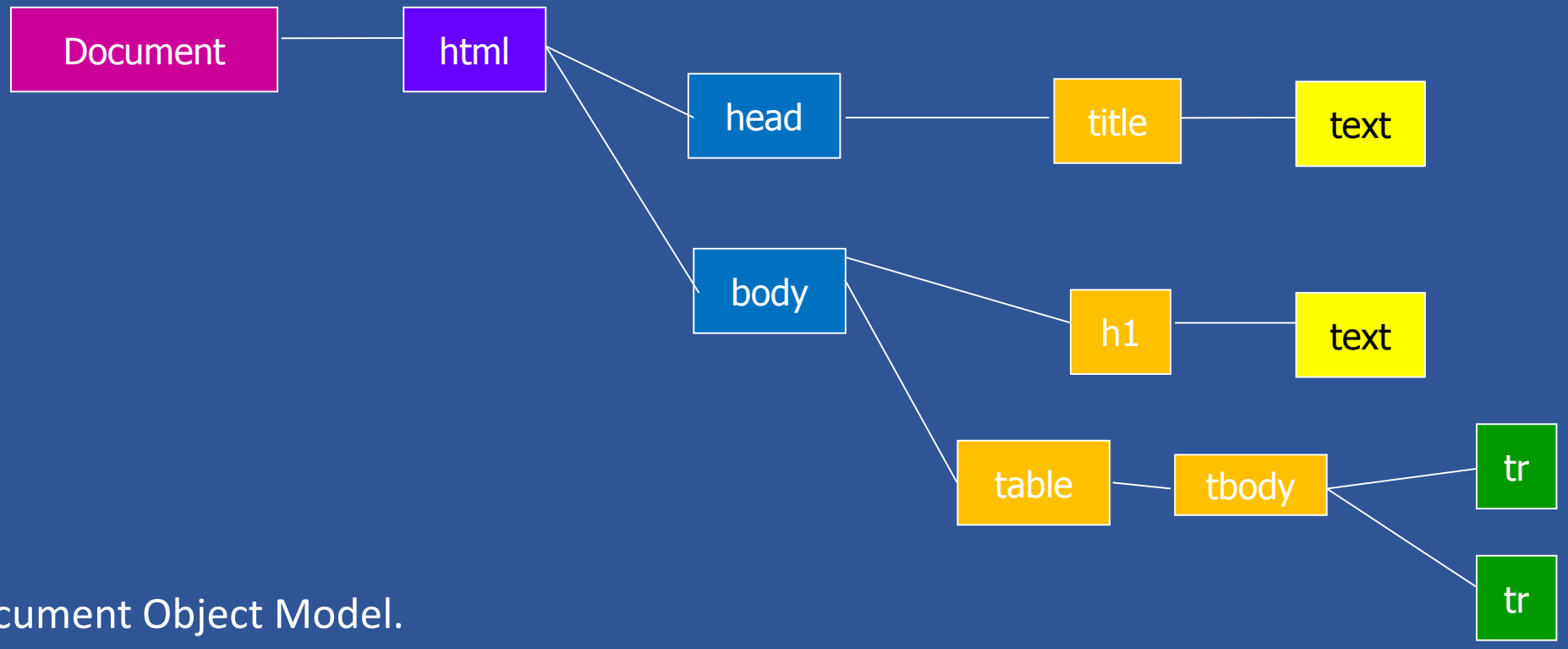
JavaScript - DOM

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 conseguir 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);).

DOM: Modificación de propiedades de estilo

Accediendo a los elementos del DOM se pueden modificar las propiedades CSS de los elementos. Para ello cuentan con la propiedad ***style*** que da acceso a todas las propiedades de estilo del elemento.

Las propiedades de estilo son accedidas desde JavaScript con el mismo nombre que en CSS, pero en lugar de separadas con guiones, con sintaxis camello (cuando cambia la palabra, la primera letra de la palabra va en mayúsculas y el resto en minúsculas).

Por ejemplo para modificar el color de fondo:

```
document.getElementById("capaColor").style.backgroundColor = "yellow";
```

Por ejemplo para modificar la posición de un elemento con posicionamiento absoluto:

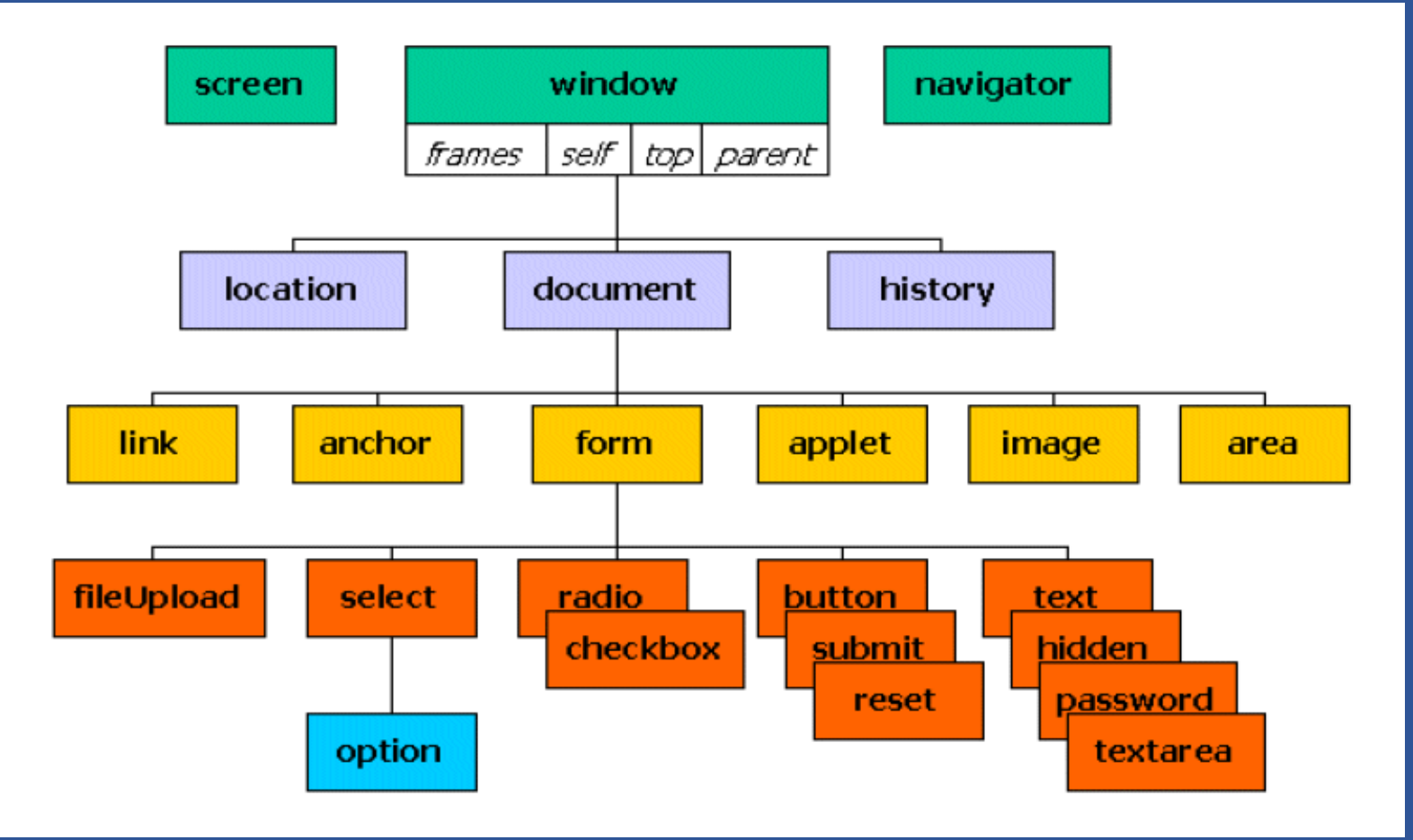
```
document.getElementById("capaMover").style.left = "100px";
```

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 de tipo 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");
```

JavaScript - Objetos del navegador

Profesor Vladimir Bataller

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 área de visualización del navegador (ViewPort).

```
var ancho = Math.max(document.documentElement.clientWidth, window.innerWidth || 0);
var alto = Math.max(document.documentElement.clientHeight, window.innerHeight || 0);
```

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 retorna dicho texto.

Los controles Checkbox y Radio tienen 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.

JSON

JavaScript dispone de un lenguaje especifico para anotar la estructura de objetos y matrices.

Matrices

Se declaran como un lista entre corchetes [] de valores separados por comas.

```
var matriz = ["Lunes","Martes",2008,true];
```

Objetos

Se declaran como una lista entre llaves {} de pares nombre:valor separados por comas.

```
var obj = {"dni":234, "nombre":"pepe", "saldo":5.43};
```

Los nombres de los atributos en los navegadores actuales no hace falta que vayan entre comillas.

JSON -> JavaScript Object Notation

JSON

Las propiedades de los objetos y los elementos de una matriz a su vez pueden ser objetos y elementos.

```
var obj = {dni:234,
        nombre:"pepe",
        saldo:5.43,
        hijos:["pepito","pepita"] ,
        domicilio:{calle:"Serrano", numero:32, codigoPostal:28003}
        };



alert(obj.nombre); // pepe
alert(obj.hijos[1]); // pepita
alert(obj.domicilio.calle); // Serrano
```

Conversión JSON - String

La notación JSON se puede emplear para codificar la información enviada en comunicaciones Ajax.

Para convertir un objeto JSON recibido como un string en una petición asíncrona como valor de retorno del atributo responseText, emplearemos el método ***parse*** de la clase JSON.

```
var txt = request.responseText; // String
//var txt = '{dni:234, nombre:"Pepe", saldo:5.43}';
var obj = JSON.parse(txt);  // parse: Convierte String ----> Objeto Javascript
alert(obj.nombre);//pepe
```

La clase JSON también dispone de un método ***stringify*** que recibe un objeto JavaScript y retorna un String con la representación JSON del objeto suministrado.

```
var obj= {dni:234, nombre:"Pepe", saldo:5.43};
var txt = JSON.stringify(obj); // stringify: Convierte un Objeto Javascript ----> String en formato JSON.
alert(txt);
```

AJAX

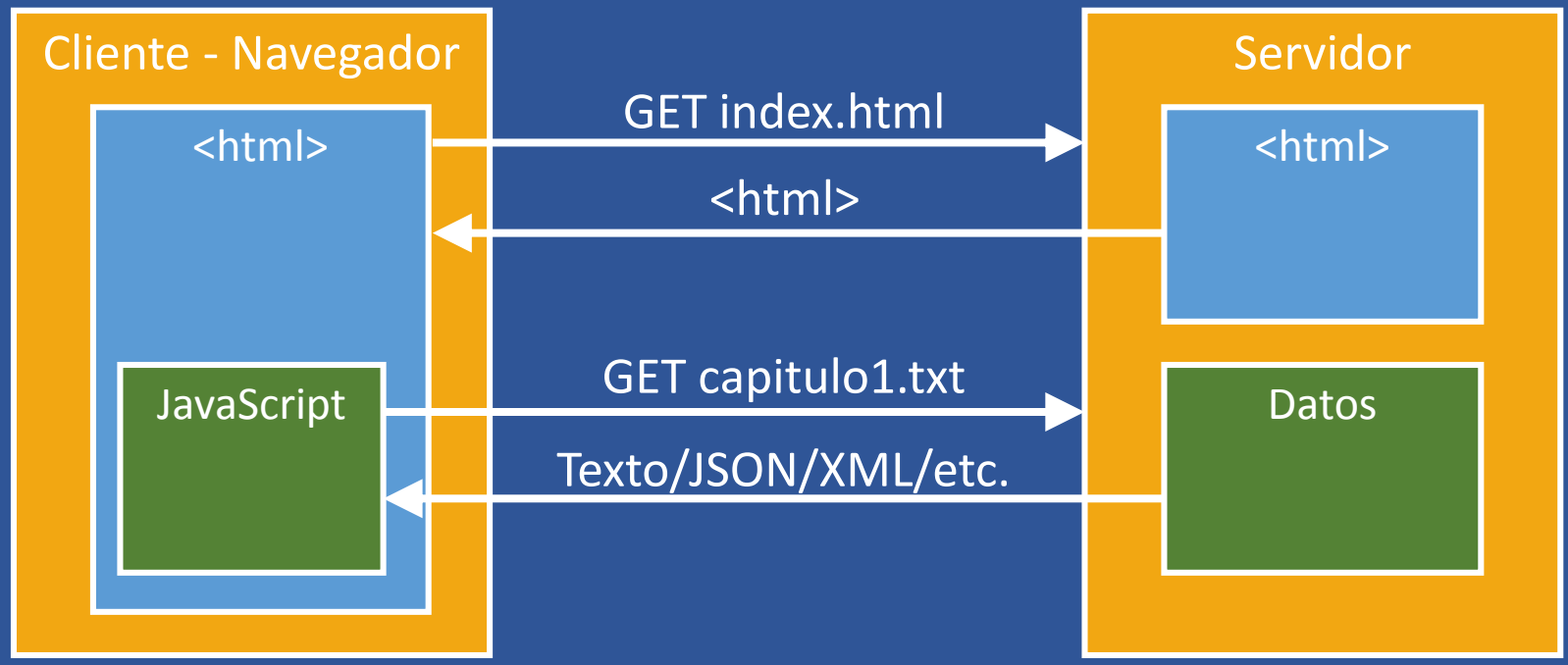
Asynchronous JavaScript And XML

AJAX

Profesor Vladimir Bataller

AJAX

Una vez cargada una página HTML, esta desde su JavaScript puede realizar peticiones http al servidor para enviar/recibir datos desde este.



Ajax

Ajax es una tecnología basada en peticiones asíncronas con JavaScript, DOM, XML .

Una sola página muestra, a efectos del usuario, varias páginas web. La información de dichas páginas las recibe mediante JavaScript de forma asíncrona y construye parcial o totalmente una nuevo árbol DOM.

Los objetos que permiten la comunicación asíncrona con el servidor son de la clase XMLHttpRequest (para navegadores todos los navegadores incluido IE7).

```
var req = new XMLHttpRequest();
```

Ajax: Pasos

Ajax es una tecnología basada en Javascript, Css, XML y XHTML .

pasos a seguir para realizar una comunicación asíncrona desde JavaScript:

1.- Obtener un objeto XMLHttpRequest.

2.- Asociar una función al evento de llegada de la respuesta (función de callback o función de respuesta) .

3.- Enviar la petición mediante el objeto XMLHttpRequest.

4.- En el evento de llegada de la respuesta se deberá modificar el documento (DOM) con los cambios que correspondan al análisis de la respuesta.

Ajax: cambio de estado

Paso2 y 3:Enviar la petión y asociar la función que se ejecutará al recibirse la respuesta.

```
var url = "paginaQueRecibeLaPetición";
request.open("GET",url);
request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status=200) {
                //---- Codigo a ejecutar cuando se recibe la respuesta ----;
                request.responseText // contiene el texto de la respuesta
                request.responseXML// Nodo XML con la respuesta
        }
}
request.send(null);
```

Ajax: Metodos

métodos de los objetos XMLHttpRequest

open("method", "URL"[, asyncFlag[, "userName"[, "password"]]])

Asigna el método (GET o POST), la URL de destino, si la petición será asíncrona (true) , usuario y contraseña

send(content)

Envía la petición, opcionalmente se puede enviar los parámetros post en formato "nombre=valor&nombre=valor" o un objeto DOM

setRequestHeader("label", "value")

Asigna un valor al par label/value para la cabecera enviada.

abort()

Detiene la petición actual

Ajax: Propiedades

Propiedades de los objetos XMLHttpRequest

onreadystatechange: El manejador del evento llamado en cada cambio de estado del objeto (Se le debe asignar la función que procese los cambios de estado en la comunicación, sobre todo cuando se reciba la respuesta). Esta función recibe un *event* que en su propiedad ***target*** y ***currentTarget*** apuntan al XMLHttpRequest de esta comunicación.

readyState: Entero que indica el estado del objeto request y de la comunicación: 0 = sin inicializar(antes de open), 1 =open ejecutado, 2 = cabecera recibida, 3 = recibiendo el cuerpo, 4 = Operación completada.

responseText: Cadena de texto con los datos devueltos por el servidor

responseXML: Respuesta XML, objeto document del DOM resultante de analizar el XML recibido.

status: Código numérico http devuelto por el servidor, ejemplos: 404 si la página no se encuentra,200 si todo ha ido bien, etc.

statusText: Mensaje que acompaña al código de estado.

Peticiones sucesivas

Cuando el servidor no envía cabeceras para que el navegador no guarde la página en caché, se puede dar la siguiente situación.

En algunos navegadores, cuando se hacen varias peticiones sucesivas al mismo recurso sin ningún parámetro adicional, el navegador interpreta que va a retornar los mismos datos y retorna el mismo valor de la petición anterior, sin realizar realmente la petición al navegador.

Solución: Agregarle a la petición GET un parámetro adicional (que no se usará en el servidor) con un valor diferente en cada petición.

```
var url = "reloj?a="+(Math.round(100000*Math.random()));
request.open("GET",url);
```

responseText	Contiene el texto recibido desde el servidor

responseText: En Mozilla, se puede leer durante la descarga de información, antes de que esta termine. Por lo tanto se puede mostrar al usuario la cantidad de información que se lleva descargada. Sin embargo, otros navegadores, esta propiedad no contiene nada hasta que no ha finalizado correctamente la descarga (readyState==4 y status==200).

responseXML

Propiedades de los objetos XmlHttpRequest

responseXML: objeto de tipo Document resultante de analizar la respuesta XML. Con **documentElement** se obtiene el elemento raíz del xml-> request.responseXML.documentElement

Para evitar que Internet Explorer retorne null por no reconocer los acentos y ñ del XML. El servidor debe enviar este documente con la cabecera:

```
xml = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n"
```

Nota: debe terminar en un salto de línea como se muestra aquí.

Escape de parámetros

En las peticiones GET los parámetros se envía en la uri, por lo que hay que codificar los caracteres no permitidos.

cuando enviamos en una petición parámetros, hay que escaparlos. Esto tradicionalmente se hace con la función **escape()** de JavaScript. Esta no funciona correctamente para el carácter '+' que debería convertirse al código '%2B', en su lugar se puede emplear la función **encodeURIComponent()**.

Para realizar el **proceso** inverso JavaScript tiene la función **unescape()**;

Ajax: Peticiones Post

El las peticiones post, los valores se pasan en forma clave=valor&clave=valor en el método send.

```
var params = "clave1=valor1&clave2=valor2";
request.open("POST", url);
request.setRequestHeader("Content-type",
            "application/x-www-form-urlencoded");
request.onreadystatechange = function() {...}
request.send(params);
```

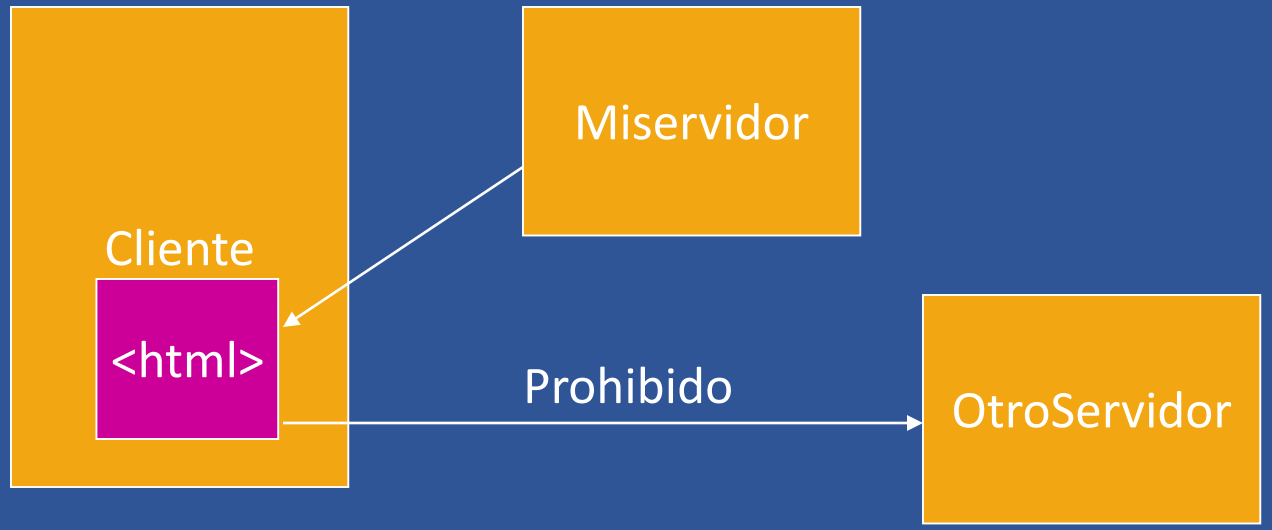
Ajax:Peticiones Post-Xml

El las peticiones post, los valores se pasan en forma clave=valor&clave=valor en el método send.

```
var xml = "<?xml version='1.0' encoding='UTF-8'?>"
          +"<clientes>...</cliente>";
request.open("POST", url);
request.setRequestHeader("Content-type","text/xml; charset=UTF-8");
request.onreadystatechange = function() {...}
request.send(xml);
```

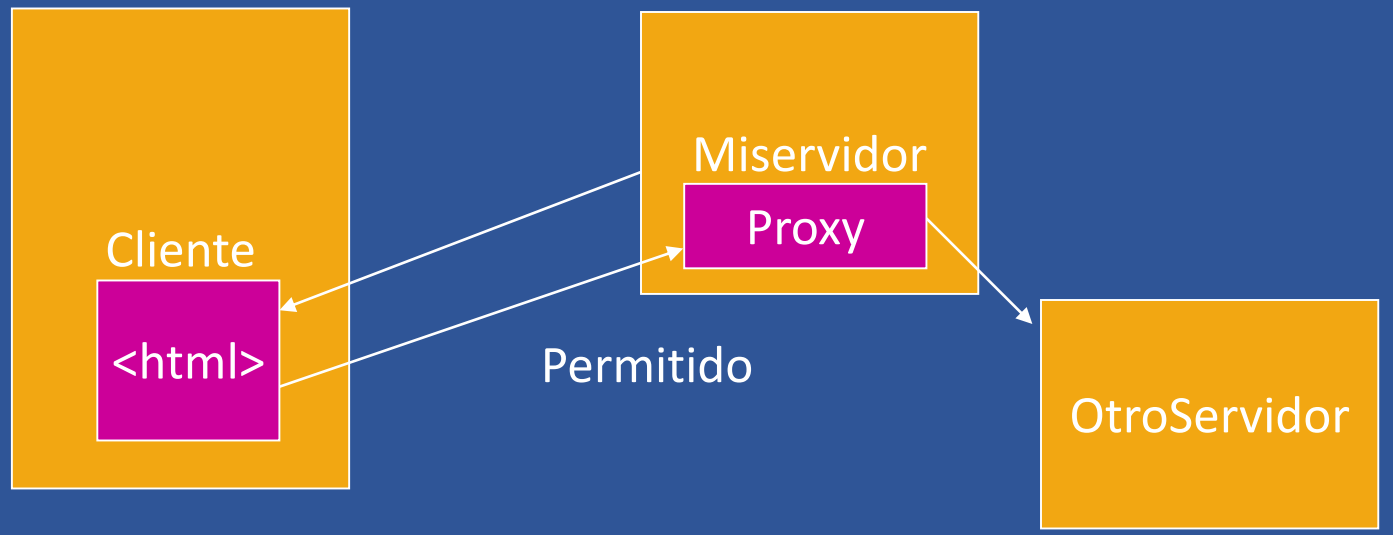
Política del mismo origen

Un documento HTML solamente puede hacer peticiones Ajax al mismo servidor que desde el que se ha cargado la página.



Política del mismo origen, solución con proxy

Para salvar esta restricción empleamos nuestro servidor como proxi. El cliente le hace la petición a nuestro servidor y este la transmite al otro.



cabeceras cross-domain

Para que un servidor permita peticiones cross-domain, este debe incluir en sus respuestas suministrar las siguientes cabeceras.

Cabeceras cross-domain

```
Access-Control-Allow-Origin:*
Access-Control-Allow-Headers: X-Requested-With
```

Ejemplo de servidor

En un servidor Node.js/Express, se podría incluir lo siguiente:

```
app.all('/', function(req, res, next) {
        res.header("Access-Control-Allow-Origin", "*");
        res.header("Access-Control-Allow-Headers", "X-Requested-With");
        next();
});
```

jQuery

JQuery

La biblioteca JQuery permite realizar operaciones típicas de JavaScript y Ajax con facilidad.

JQuery se basa en la clase jQuery, cada instancia de esta clase representa un elemento o colección de elementos del documento HTML, este objeto permite hacer modificaciones sobre las propiedades de esos elementos del documento.

La obtención de los objetos jQuery se puede realizar con la función $. Esta función permite hacer referencia a los elementos del documento mediante una sintaxis de selectores css y xpath.

Ejemplos:

```
$("div#capa1"); // Retorna el objeto jQuery que representa al elemento <div id="capa1"></div>

$("div"); // Retorna un objeto jQuery que representa a todos los elementos <div></div> del documento.
```

Nota: $ puede recibir un segundo parámetro que es un documento XML sobre el cual realizar la búsqueda en lugar de realizarla sobre el HtmlDom.

jQuery

Profesor Vladimir Bataller

Cómo incluir jQuery en un html

Hay que incluir en nuestro HTML y mediante una etiqueta <script src=""></script> el archivo js de jQuery para poder usarlo en una página.

Opciones para obtener el archivo js de jQuery:

1.- Descargar el js de jQuery de su sitio web (http://jquery.com/download/).

2.- Emplear un cdn que nos sirva el archivo en lugar de tenerlo en nuestro servidor:

```
<script  src="http://code.jquery.com/jquery-3.3.1.js"
        integrity="sha256-2Kok7MbOyxpgUVvAk/HJ2jigOSYS2auK4Pfzbm7uH60="
        crossorigin="anonymous"></script>
```

3.- Descargarlo empleando npm (gestor de paquetes de node). Para ello habría que ejecutar el comando:

```
npm install --save  jquery
```

Tras esto, podríamos encontrar el archivo de jquery en la carpeta node_modules/jquery/dist

JQuery métodos (I)

La biblioteca JQuery permite realizar operaciones tipicas de javascript y ajax con facilidad.

css: Permite acceder a las propiedades de estilo CSS de un elemento HTML.

css(String nombrePropiedadEstilo): Retorna el valor de la propiedad de estilo especifcada.

css(String nombrePropiedadEstilo, String valorPropiedadEstilo): Establece el valor de la propiedad de estilo

css(Object json): Establece el conjunto de propiedades CSS indicadas en un objeto JSON.

attr: Permite acceder a los atributos de un elemento html.

attr(String nombreAtributo): Retorna el valor de atributo especificado.

attr(String nombreAtributo, String valorAtributo): Establece el valor del atributo

attr(Object json): Establece el conjunto de propiedades indicadas en un objeto JSON.

css: Permite acceder a las propiedades de estilo de un elemento html.

Tiene la misma sintaxis que *attr*.

height, width: Permite leer y modificar el alto y ancho de un elemento html

Para modificar, recibe como parámetro el valor nuevo para el alto o ancho.

Para leer no recibe ningún parámetro y retorna un number con los pixels.

JQuery métodos (II)

La biblioteca JQuery permite realizar operaciones tipicas de javascript y ajax con facilidad.

html(): leer o cambiar el innerHTML de un elemento.

html(): Retorna un string con el HTML interior del elemento asociado.

html(String html): Cambia el html suministrado dentro del elemento.

*Nota: html() no funciona sobre documentos xml, solamente con docs html.

text(): Permite acceder a texto interior a un elemento ignorando las etiquetas html.

text(): Retorna todo el texto interior del elemento y sublementos.

text(String texto): Inserta el texto dentro del elemento

Nota: text si funciona con documentos xml.

val(): Permite leer y modificar el valor de los controles de formulario

val(): Retorna el valor del control de fomulario.

val(String valor): Asigna el valor sumistrado al control de formulario.

JQuery métodos (III)

La biblioteca JQuery permite realizar operaciones tipicas de javascript y ajax con facilidad.

get(): permite acceder a los objetos del Dom asociados al objeto jQuery.

get(): Retorna una matriz de objetos del Dom asociados al objeto jQuery.

get(i): Retorna el elemento del Dom i-esimo de los asociados al objeto jQuery .

Número de objetos Dom contenidos en un objeto jQuery

~~size(): Método que retorna el número de elmentos de jQuery. (Deprecated 1.8)~~

length: Propiedad que retorna el número de elementos del jQuery.

each(function callback)

Ejecuta la función pasada como parámetro una vez por cada objeto Dom referenciado por el objeto jQuery. Dentro de esta función this apunta al objeto Dom que se está procesando en esa iteración. Si la función retorna false, se sale del bucle.

La función de callback, recibe como primer parámetro el índice de el elemento dentro de la colección y como segundo parámetro el elemento del dom de la iteración actual (igual que this)

Conversión jQuery-DOM

Convertir objetos Dom en objeto Jquery

```
var objJQuery = $(objDom);
```

Obtener el objeto Dom asociado a un objeto Jquery (el primero)

```
var objDom = objJQuery.get(0);
```

JQuery: búsquedas y filtrados

jQuery dispone de métodos para reducir el número de elementos encontrados mediante los selectores.

find(selector): retorna un objeto JQuery con los nodos hijos que cumplan el selector.

filter(selector): reduce los elementos del objeto JQuery actual a solamente los que cumplan la condición del selector.

Evento Ready

El evento ready se ejecuta una vez que el navegador ha creado el árbol DOM del documento.

Al objeto jQuery asociado al elemento document se le puede asociar un manejador de evento que se ejecute al cargarse el documento HTML y su DOM, pero antes de que se carguen otros elementos tales como imágenes, al contrario de lo que ocurre con el window.onload.

```
$(document).ready( function({
                alert("documento cargado");
    }
);
window.onload = function ({
    alert("todo cargado");
};
```

Asignación de eventos

JQuery permite agregar manejadores de eventos para los elementos HTML. Entre otras ventajas permite manejar de una forma uniforme determinadas características dependientes del navegador.

on(String evento, function manejador): Asigna un manejador al evento del elemento.

El manejador de eventos puede recibir un parámetro event que representa al evento producido y que proporciona información sobre este.

Adicionalmente, se dispone de una variable *this* (no pasada como parámetro en el manejador) que representa el elemento DOM sobre el cual se está procesando el evento. No hay que olvidar que los eventos pueden transmitirse en forma de burbuja desde el elemento más interior hasta el elemento contenedor más exterior.

Eventos posibles

blur, change, click, dblclick, error, focus, keydown, keypress, keyup, load, mousedown, mousemove, mouseout, mouseover, mouseup, resize, scroll, select, submit, unload

JQuery event

Las funciones manejadoras de eventos reciben un parámetro ***event*** que representa al evento producido y que proporciona información sobre este.

atributos del objeto event

pageX, pageY: contiene la coordenada x o y de la posición del ratón respecto a la esquina superior izquierda de la página al producirse el evento.

preventDefault(): Impide el comportamiento por defecto del evento. Por ejemplo en un hipervínculo, evita que se navegue a la dirección de destino.

stopPropagation(): Evita la propagación del evento hacia los elementos contenedores.

Si el manejador del evento retorna *false*, equivale a la ejecución de preventDefault y stopPropagation.

target: elemento DOM que recibió el evento.

currentTarget: elemento DOM que tiene asignado el manejador de evento.

JQuery métodos para eventos: off, one, trigger

JQuery permite agregar manejadores de eventos para los elementos HTML. Entre otras ventajas permite manejar de una forma uniforme determinadas características dependientes del navegador.

off(String evento): Desasigna el manejador de evento asociado al elemento.

one(String evento, function manejador): Asigna el evento de la misma forma que ***on***, pero una vez ejecutado el evento, se desasignará automáticamente.

trigger(String evento): Dispara el evento indicado asociado al elemento.

JQuery métodos para eventos: click, toggle, dblclick

JQuery permite agregar manejadores de eventos para los elementos HTML. Entre otras ventajas permite manejar de una forma uniforme determinadas características dependientes del navegador.

click(function manejador): Asigna el manejador para la acción de hacer click sobre el elemento asociado.

dblclick(function manejador): Asigna el manejador para la acción de hacer doble click sobre el elemento asociado.

~~**toggle(function manejadorImpar, function manejadorPar)**~~**:** Asigna los manejadores alternativos al hacer click. Uno se ejectua una vez y la siguiente el otro. ***Deprecated*** en la versión 1.8

JQuery: animate()

Permite crear transiciones de múltiples propiedades css simultáneamente.

.animate(css_json_properties, tiempo, easingType, callback)

Anima todas las propiedades simultáneamente. Al terminar puede llamar a la función de callback especificada.

EasingType indica la función empleada para calcular el grado de evolución a lo largo de la aplicación de la animación. Por defecto, el único valor posible es linear. Si se quiere definir otro, hay que aplicar la función JQuery como se indica.

```
jQuery.extend({
        'easing': {'linear': function(x, t, attrStart, attrDelta,T) {
        return x * attrDelta + attrStart;}     }});
```

Donde

x: valor entre 0 y 1 que indica el grado de avance de la animación.

t y T: tiempo, en ms, transcurrido desde el inicio de la animación y tiempo total.

attStart y attDelta: Valor inicial del atributo e incremento que sufrirá al final.

JQuery: hide(), show(), fadeXX() y slideXX()

Los objetos JQuery disponen de dos métodos hide y show que ocultan (display="none") y muestran (display="valor inicial") el objeto subyacente.

hide y show temporizados

Adicionalmente, se puede suministrar un parámetro numérico para indicar los milisegundos que tarda en aplicarse el efecto y mientras tanto se va modificando la opacidad, el alto y ancho del objeto.

.hide (tiempo_en_ms)

.show(tiempo_en_ms)

También existen tres constantes predefinidas para el tiempo slow (600 ms), normal (400 ms), fast (200 ms)

Los métodos **.slideUp(t)** y **.slideDown(t)**, hacen que desaparezca y aparezca el objeto modificando su altura hasta que aparezca o desaparezca.

Los siguientes métodos hace que aparezca o desaparezca el objeto sin redimensionarlo, solamente actuando sobre su opacidad.

.fadeIn (tiempo_en_ms)

.fadeOut(tiempo_en_ms)

jQuery

Profesor Vladimir Bataller

JQuery Traversing

El objeto jQuery, también dispone de métodos para manipular el dom de los elementos asociados a el.

append(contenido): Agrega un hijo a los elementos asociados al objeto jQuery. Ese hijo se pasa como parámetro bien como un String con el HTML o un objeto Dom o un objeto jQuery.

clone(boolean): retorna un objeto jquery con una copia de todos los nodos del objeto jQuery original. Si se pasa true como parámetro además se copian también los manejadores de eventos de cada elemento.

empty(): Elimina todos los nodos hijos del elemento asociado al jQuery.

remove(): Elimina del DOM los elementos asociados.

appendTo(selector de destino): Añade el objeto representado por el objeto jQuery actual al objeto indicado por el selector de destino.

Para crear un nuevo elemento, se ejecuta la función $ indicándole en su interior la etiqueta HTML a crear, esta etiqueta se pone con los signos de mayor y menor (<>).

var nuevo = **$(" ")**;

var cajaNueva = $("**<input type="text" />**");//En los input hay que poner el tipo

desde el principio sino en IE, luego ya no se puede cambiar.

JQuery Drag (I)

Para hacer que un elemento sea arrastrable, basta con ejecutar sobre un objeto jQuery asociado a él el método: draggable(options)

Propiedades del options de draggable

revert: Si vale true, al soltar el objeto, volverá a su posición inicial.

opacity: Valor de opacidad que tendrá el objeto al desplazarse.

start: Callback que se ejecuta al iniciarse el arrastre.

stop: Callback que se ejecuta al terminar el arrastre.

drag: Función de callback que se ejecuta continuamente durante el arrastre.

handle: Esta propiedad indica el elemento interior al dragable sobre el cual hay que pinchar para arrastrarlo. Es el elemento o objeto jQuery.

helper: Indica que es realmente lo que se está moviendo. Puede ser un String con el valor "original" que es el valor por defecto o puede ser "clone" indicando que lo que se mueve es una copia del objeto original. También se puede pasar una función que recibe el elemento del dom original y retorna el elemento del dom a mover.

JQuery Drag (II)

Parámetros recibidos por las funciones de callback start, drag y stop

event: Este es el primer parámetro recibido y representa el evento. En concreto su propiedad target es el elemento del dom que está siendo arrastrado.

propiedades: Es el segundo parámetro recibido y tiene las siguientes propiedades:

- **position:** tiene las propiedades top y left en el caso de "start" no está definido. En el caso de "drag" y "stop" tiene las coordenadas de la esquina del objeto.

Dentro de estas funciones "this" representa el elemento del dom que está siendo arrastrado.

JQuery Drop (I)

Para hacer que un elemento pueda interactuar con los objetos que se arrastren sobre él, debemos ejecutar sobre su objeto JQuery, el método:

.droppable(options)

Propiedades del objeto options de droppable

accept: String con los selecctores JQuery que identifican los objetos arrastrables sobre el droppable (p.e.: "div.movible" -> todos los div de con class="movible").

activeClass: clases de estilo que se aplicará al droppable cuando un objeto draggable, que se puede soltar sobre él, empieza a moverse.

hoverClass: Clase de estilo que se aplicará al droppable cuando un objeto draggable, que se puede soltar sobre él, pasa por encima.

JQuery Drop (II)

Propiedades del objeto options de droppable

activate: función que se ejecuta cada vez que un objeto que puede arrastrarse sobre el droppable empieza a ser arrastrado.

deactivate: función cuando un objeto arrastrable sobre este deja de ser arrastrado.

over: Cuando un objeto arrastrable empieza a estar encima del droppable.

out: Cuando un objeto arrastrable termina de estar encima del droppable.

drop: Función que se ejecuta al soltar el objeto draggable sobre el droppable.

JQuery Drop (III)

Parámetros pasados a las funciones de callback de un droppable

event: Contiene los datos del evento.

propiedades: Contiene los siguientes atributos:

- **draggable:** Objeto jQuery asociado al elemento del dom que inicialmente está siendo arrastrado.
- **droppable:** Objeto jQuery que recibe el arrastrado.
- **options:** opciones de configuración del droppable.
- **helper:** Objeto jQuery asociado al elemento del dom que realmente está siendo arrastrado.

JQuery Ajax (I)

La función $.ajax(Object opciones) realiza una petición asíncrona de acuerdo con las opciones suministradas en el objeto JSON pasado como parámetro.

Atributos de objeto opciones

url: url de destino.

type: GET o POST (el valor por defecto es GET).

dataType: Tipo de datos que se espera recibir del servidor (xml, html, json, script o text).

timeout: Tiempo máximo antes de considerar un fallo.

global: true (valor por defecto) indica que los manejadores Ajax por defecto serán empleados.

beforeSend: callback ejecutada antes del envío.

error: callback si se produce un error.

success: callback si se recibe respuesta correctamente, el manejador recibe como parámetro los datos recibidos.

complete: callback cuando ha terminado la comunicación correcta o incorrectamente.

JQuery Ajax (II)

data: objeto json o string con los datos a enviar.

processData: true (valor por defecto) indica que los datos enviados se deben convertir de un form a una querystring.

contentType: tipo mime de los datos enviados (por defecto application/x-www-form-urlencoded).

async: true (valor por defecto) indica que la petición es asíncrona.

La función **$.ajax** retorna el objeto *XMLHttpRequest* asociado a la petición.

JQuery Ajax funciones de callback

Parámetros de las funciones de callback.

A continuación se muestra qué argumentos recibe cada una de las funciones de callback:

complete(datos, statusText, request)

success(datos,okText,request)

error(request,msgError,exceptionText)

Donde:

datos: Datos recibidos.

statusText: "success" o "error"

request: Objeto XMLHttpRequest asociado a la petición.

exceptionText: Si el error ha sido producido por una excepción

JQuery Ajax métodos derivados

jQuery proporciona los siguientes métodos de utilidad para simplificar su uso en casos específicos.

Para respuestas xml o texto

$.get(String url, JsonObj datos,function success): petición Ajax get. La función de callback recibe como parámetro los datos recibidos.

$.post(String url, JsonObj datos,function success): petición Ajax post. La función de callback recibe como parámetro los datos recibidos.

Para respuestas html o texto

.load(String url, JsonObj datos,function complete): carga el HTML recibido en el elemento asociado al objeto jQuery sobre el que se ejecuta el método.

Para respuestas JSON y javascript

$.getJSON**(String url, JsonObj datos,function success):** petición ajax get. La función de callback recibe un objeto json con la respuesta.

$.getScript(String url, function success): ejecuta el script recibido.

JQuery Ajax funciones globales

En jQuery-Ajax se pueden definir las funciones manejadoras por defecto para success, complete y error.

Función global de error

$(document).ajaxError(*function manejadorDeErrorPorDefecto***);**

Función global de success

$(document).ajaxSuccess(*function manejadorDeSuccessPorDefecto***);**

Función global de complete

$(document).ajaxComplete(*function manejadorDeCompletePorDefecto***);**

JQuery Form.serialize

Cuando tenemos formularios muy voluminosos, serialize nos facilita el trabajo de componer la querystring.

Método .serialize()

Los objetos jQuery asociados a un *form*, tienen el método ***.serialize()*** que retorna todos los campos del formulario en formato querystring (*clave=valor&clave=valor*). Para eso se toma la propiedad ***name*** de cada uno de los campos del formulario.

POO

Programación orientada a objetos con JavaScript

Creación de objetos

En javascript se pueden crear clases de objetos propios para usar en nuestra aplicación.

Creación de clases y objetos

1.- Creación del Contructor de la clase (función-constructor).

2.- Declaración de propiedades.

3.- Declaración de métodos y asignación a una clase.

4.- Creación de objetos de la clase.

Constructor

El constructor es una función con el nombre que se le va a asignar a la clase. Puede recibir los parámetros para la inicialización de atributos.

La nomenclatura JavaScript recomienda poner en mayúscula la primera letra del nombre de las clases.

En este ejemplo, el constructor recibe dos parámetros (t y a) que se emplean para inicializar los atributos titulo y autor a los cuales se accede mediante la variable this.

```
function Libro(t, a)
{       this.titulo=t;
        this.autor=a;
}
```

Asignación de métodos

Los métodos son funciones JavaScript que se asocian a una clase, para ello se emplea la variable prototype de la clase que hace referencia a su estructura.

Primero creamos la función (para acceder a los atributos se emplea this):

```
function Libro_imprimir(){
        return this.titulo + "(" + this.autor + ")";
}
```

Posteriormente asignamos la función a la clase con el nombre que queramos que tenga el método.

```
Libro.prototype.imprimir = Libro_imprimir;
```

JavaScript - POO

Profesor Vladimir Bataller

Uso de objetos de clases propias

Crear objetos de una clase, primero deberemos importar el archivo donde esté definida y posteriormente mediante new crear objetos de la clase y acceder a sus atributos y métodos.

```
<html><head><title>Ejemplo de uso de libros</title>
<script type="text/javascript" src="Libro.js" ></script>
</head><body>   Se van a mostrar dos libros ... <br />
        <script type="text/javascript">
                var libro1 = new Libro("Quijote", "Cervantes");
                libro1.titulo = "Don Quijote de la Mancha";
                var libro2 = new Libro("El buscón", "Quevedo");
                alert(libro1.imprimir());     alert(libro2.imprimir());
        </script>
        fin </body></html>
```

Acceso a métodos y a atributos en forma matricial

Además de mediante un punto, se puede acceder a las propiedades y métodos de un objeto como si fueran elementos de una matriz.

```
var libro1 = new Libro("Quijote", "Cervantes");
alert(libro1["titulo"]);        alert(libro1["autor"]);
alert(libro1["imprimir"]()); alert(libro2["imprimir"]());
document.write("===== Propiedades del libro1 =====<br />");
// Bucle for para recorrer todas las propiedades del objeto
for(nombrePropiedad in libro1)
{           document.write("---- "+nombrePropiedad + " ----<br />");
            document.write(libro1[nombrePropiedad] + "<br />");
}
```

metodo toString de Objetos

Cuando un objeto se ha de convertir a String, javascript busca el método toString del objeto y el String que este retorna será la reperesentación textual del objeto.

```
function Persona_toString()
{
        return "id:" + this.id + "<br />"
        + "nombre:" + this.nombre + "<br />"
        + "telefono:" + this.telefono;
}

Persona.prototype.toString = Persona_toString;
var p = new Persona(4,"Pepe","67273737")
document.getElementById("detallepersona").innerHTML = p;
```

mejorar los objetos implicitos de JS

A los objetos implicitos de Javascript se les pueden agregar nuevas propiedades y métodos para mejorar su funcionalidad o adaptarla a las necesidades de nuestra aplicacion.

```
var p = new Persona(4,"Pepe","67273737")
var opt = new Option(p.nombre,this.p);//Clase de Javascript
opt.persona = p; // Le agregamos una nueva propiedad
var lista = document.getElementById("personas");
lista.options[1]= opt;


var lista = document.getElementById("personas");
var opt = lista.options[lista.selectedIndex];
document.getElementById("detallepersona").innerHTML = opt.persona; // usamos el atributo persona del option
```

Expresiones Regulares

Expresiones regulares con JavaScript

Expresiones Regulares

Profesor Vladimir Bataller

Expresiones Regulares (RE)

Las expresiones regulares son un estandar para crear patrones de cadenas de texto. Se emplea para realizar busquedas y filtrados

En las expresiones regulares, los caracteres siguientes tienen un significado especial

^ $. * + ? = ! : | \ / ( ) [ ] { }

Cuando se desee representar cualquiera de estos caracteres de forma literal y no por su significado especifico, deberán ir precedidos de una contrabarra o backslash "\" .

Sin embargo, los siguientes caracteres no tienen significado especial y se representan a ellos mismos.

" ' @

Expresiones Regulares

Profesor Vladimir Bataller

Clases de caracteres

Las clases de caracteres representan un caracter que debe encontrarse entre un conjunto de caracteres especificados.

Especificación de Clases de caracteres mediante corchetes "[]"

1.- Se puede indicar una lista de caracteres (no se separan, ni por comas ni por espacios, van juntos uno detrás de otro). p.e.: "[aeiou]" el carácter que cumpla este patrón será una vocal.

2.- Se puede indicar un rango de valores. Se emplea el guion separando el valor inicial del valor final. p.e.: "[\dA-Fa-f]" expresa un digito hexadecimal (un número o una letra de la "a" a la "f" en mayúsculas o minúsculas).

Otro ejemplo: "diversi[óo]n" permite detectar tanto si lleva acento como si no lo lleva. Pero solamente con una "o".

Clases de caracteres negadas

Las clases de caracteres representan un carácter que debe encontrarse entre un conjunto de caracteres especificados.

Clases de caracteres negadas mediante corchetes "[]"

Permite indicar el conjunto de caracteres que NO debe encontrarse. Se indica con el carácter ^ después de abrir el corchete

"[^abc]" representa cualquier carácter que no sea a, b o c.

Expresiones Regulares

Profesor Vladimir Bataller

Caracteres no imprimibles

\t — Tabulador.

\r — Retorno de carro.

\n — Nueva linea.

\x*nn* — Código ASCII o ANSI. p.e: copyright en Latin-1 es "\xA9".

\u*xxxx* — Codigo Unicode.

Expresiones Regulares

Profesor Vladimir Bataller

Caracteres imprimibles

Punto ".": Representa a cualquier caracter excepto saltos de línea.

\d — Un dígito del 0 al 9.(digit)

\w — Un carácter alfanumérico(letra,numero o guión bajo).(word)

\s — Un espacio en blanco.(space)

\D — Cualquier carácter que no sea un dígito del 0 al 9.

\W — Cualquier carácter no alfanumérico.

\S — Cualquier carácter que no sea un espacio en blanco.

\b — Representa la posición de inicio o final de una palabra. es decir cuando se cambia de \w a \W o viceversa.

\B — Marca la posición entre dos caracteres alfanuméricos o dos no-alfanuméricos. Es decir las posiciones en las que no cambia la palabra o nose cambia de no-palabra.

Expresiones Regulares

Profesor Vladimir Bataller

Cuantificadores

Los cuantificadores, ubicados detrás de un carácter o clase de caracteres indican cuantas veces puede aparecer el carácter en las cadenas que sigan el patrón.

Cuantificadores

+ Una o más veces. Por ejemplo, "ho+la" describe el conjunto infinito hola, hoola, hooola, hoooola, etc.

? Cero o una vez. Por ejemplo, "ob?scuro" casa con oscuro y obscuro.

***** Cero, una, o más veces. Por ejemplo, "0*42" casa con 42, 042, 0042, 00042, etc

Expresiones Regulares

Profesor Vladimir Bataller

Inicio y final de cadena

"$" y "^" detectan el final y principio de línea. "^" no puede estar dentro de corchetes pues tendría otro significado.

Final de cadena con "$"

Representa el final de una cadena de caracteres (o el final de una línea si se trabaja en modo multilínea).

Ejemplos:

"ion$" -> detecta cadenas o líneas terminadas en ion.

"\.$"-> detecta cadenas o líneas terminadas en punto.

Principio de cadena con "^"

Representa el principio de una cadena de caracteres. Por ejemplo: "^[A-Z]" Representa las cadenas que empiezan en letra mayúscula.

Cuantificadores mediante llaves

Las llaves permiten determinar el número de repeticiones del carácter o bloque al que siguen.

Cuantificador por llaves "{}"

Si a continuación de "\d" o "\s" o "\w" o cualquier clase de caracteres se incluyen las llaves, se indica mediante un número o rango de números cuantas de repeticiones de la clase se espera.

sintaxis:

{n}	n repeticiones
{n,m}	entre n y m repeticiones
{n, }	n o más repeticiones

Ejemplos:

"[a-z]{4}\s\d{2,4}" -> 4 minúsculas, espacio y entre 2 y 4 dígitos.

"^\d{2}/\d{2}/\d{4}$"-> Fecha en formato dd/mm/yyyy aunque no valida los valores.

Bloqueo de repeticiones

"*" representa cero, una o más veces el carácter que le precede, así como "+" representa una o más veces.

Bloqueo de repeticiones de "*" y "+" mediante "?"

Por defecto, estos elementos tratan de abarcar el mayor numero de caracteres en su búsqueda. Por ejemplo, si se va a analizar una cadena "(lunes) y (martes)" y se le aplica una búsqueda con el patrón "\(.*\)", en lugar de retornar "(lunes)" y "(martes)", retorna "(lunes) y (martes)" porque intenta meter entre el primer y último paréntesis, todos los que se puedan. Para limitar ese comportamiento, se emplea el ? detrás del * que localiza la coincidencia más pequeña. Así pues el patrón quedaría: "\(.*?\)".

Lo expuesto para "*" es igualmente válido para "+".

Expresiones Regulares

Profesor Vladimir Bataller

Alternación

Se expresa mediante el operador "|" (tubería o pipeline) y permite separar valores alternativos. p.e.:"sabado|Sabado|domingo|Domingo"

Hay que tener en cuenta que las alternativas se evaluan de izquierda a derecha y el motor se quedará con el primer patrón que se cumpla. Por lo tanto un patrón "a|ab" aplicado a "abnegado" concuerdará solamente con el primer caracter.

Expresiones Regulares

Profesor Vladimir Bataller

RE en Javascript: match

La clase RegExp representa a las expresiones regulares. Los literales de expresiones regulares se representan entre barras /. También existe un constructor de RegExp que recibe un String.

Método match de la clase String

String[] match(RegExp)

Si la RegExp tiene el flag g activado, retorna una matriz de String con todas las coincidencia encontradas. Si el flag g no está activado retorna una matriz con un elemento correspondiente a la primera coincidencia.

```
var s = "1 hola 3# 22   , 6";
var m=s.match(/\d+/g);       //m=["1", "3", "22","6"]
```

Nota: El flag g de las expresiones regulares se activa agregando una g detrás de la barra de cierre de su declaración. /\d+/g

Expresiones Regulares

Profesor Vladimir Bataller

RE en Javascript: test

La clase RegExp representa a las expresiones regulares. Los literales de expresiones regulares se representan entre barras /. También existe un constructor de RegExp que recibe un String.

Método test de la clase RegExp

boolean test(String)

Retorna true si el String suministrado coincide con la expresión regular.

La clase RegExp posee un atributo llamado *lastIndex* que indica la posición del final la última coincidencia encontrada, si se repite la ejecución del método, este continuará buscando a partir de dicha posición.

La propiedad lastIndex de los objetos RegExp se pone a cero si no se encuentra coincidencia o mediante asignación directa.

Expresiones Regulares

Profesor Vladimir Bataller

RE en Javascript: exec

La clase RegExp representa a las expresiones regulares. Los literales de expresiones regulares se representan entre barras /. También existe un constructor de RegExp que recibe un String.

Método exec de la clase RegExp

String[] exec(String)

Retorna una matriz con la primera coincidencia encontrada en el el el elemento 0.

Si se quieren buscar más coincidencias, se debe ejecutar de nuevo y leer el elemento 0 otra vez.

Si no se encuentran coincidencias, retorna null.

la propiedad index de la matriz indica la posición donde se ha encontrado la coincidencia.

La propiedad lastIndex también es válida para este método, si se pone a cero se volverá a buscar desde el principio del string.