

Sintaxis de JavaScript

- Javascript distingue entre mayúsculas y minúsculas
 - **while** (correcto)
 - While, WHILE (incorrectos)
- Javascript ignora espacios en blanco, tabuladores y saltos de línea entre *tokens*
 - token: palabra reservada, número, string, nombre de función, ...
 - Conviene utilizar sangrado para que los scripts sean más legibles
- El uso de **;** al final de cada instrucción es opcional (aunque recomendable)

return
true

➔

return;
true;
- Comentarios como en Java
 - Comentario de varias líneas entre **/*** y ***/**
 - Comentario hasta el final de la línea con **//** comentario

Identificadores

- Similar a C/Java:
 - Deben comenzar por una letra o por **'_'**
 - Pueden contener letras, dígitos y **'_'**
 - No pueden coincidir con las palabras reservadas
- Palabras reservadas de JavaScript
 - break**
 - case, catch, continue**
 - default, delete, do**
 - else**
 - finally, for, function**
 - if, in, instanceof**
 - new**
 - return**
 - switch**
 - this, throw, try, typeof**
 - var, void**
 - while, with**

Literales

■ Números

- Internamente las operaciones se realizan en punto flotante
- Representación:
 - Enteros: 0, -1, 44, ...
 - Decimales (*float*): 0.20, 3.1415, -3.23e+6
 - Hexadecimal, empiezan por 0x: 0xFF, 0x1A

■ Valores lógicos (**Booleanos**)

- true y false

■ Strings

- Secuencia de caracteres entre comillas dobles " o simples '
 - "Esto es un String"
 - ...
- Secuencias de escape, para representar caracteres especiales:
 - \ ' Comilla simple \" Comilla doble
 - \b Retroceso \f Salto de página
 - \n Salto de línea \t Tabulación
 - \\ Barra inclinada \

Variables

- JavaScript es un lenguaje débilmente tipado
 - No se especifica el tipo de las variables
 - Se deduce por el contenido de la variable y el contexto
- Para declarar una variable se usa la palabra reservada **var** seguida por una lista de nombres de variables a declarar separadas por ,
 - El nombre de una variable puede estar formado por letras, números y los símbolos \$ (dólar) y _ (guión bajo)
 - El primer carácter no puede ser un número

```
var variable1, $variable, _tmp;
var x = 1;
var y = 2;
z = x + y; // z no está declarada pero al usarla por primera vez
           // se crea como variable global
```

- Para dejar una variable indefinida se le asigna el valor null
 - indefinida = null;

Expresiones

- **Asignación**
 - Guarda un valor específico en una variable
`var x = 0;`
- **Expresiones numéricas**
 - Operadores aritméticos:
 - `+`, `++`, `-`, `--`, `*`, `/`, `%` (módulo)
 - `+=`, `-=`, `*=`, `/=`, `^=` (exponenciación), `%=`
- **Expresiones lógicas**
 - Operadores lógicos: `&&` (and), `||` (or), `!` (not)

Expresiones

- **Expresiones de comparación**
 - Operadores relacionales: `==`, `!=`, `>`, `<`, `>=`, `<=`, `===`, `!==`
 - Conversión automática de tipos en las comparaciones
 - JavaScript realiza conversiones automáticas entre tipos para llevar a cabo la comparación cuando sea necesario
 - Si un operando es una cadena y el otro un número, se intenta convertir la cadena a número. Si no se puede convertir la comparación devuelve `false`
 - Si uno de los operandos es un booleano y el otro un número se convierte el booleano a número (`true` 1, `false` 0)
 - Comparación estricta (`===`, `!==`): no se realiza conversión alguna
- **Reglas de precedencia de operadores**
 - `() [] .` (el operador punto sirve para los objetos)
 - `! - ++ --`
 - `*` / `%`
 - `+-`
 - `<< >> >>>` (desplazamientos a nivel de bit)
 - `< <= > >=`
 - `== !=`
 - `& ^ |` (lógicos a nivel de bit)
 - `&& ||` (lógicos booleanos)
 - `= += -= *= /= %= <<= >>= >>>= &= ^= !=` (asignación)

Control de flujo

■ Instrucciones condicionales

■ **if**

```
if ( condición ) {           // 0, "" y null equivalen a false
    // Instrucciones
}
else {
    // Instrucciones
}
```

■ **switch**

```
switch ( expresión ) { // La expresión devuelve un numero,
                        // un valor lógico o un string
case valor1:
    // Instrucciones caso 1
    break; // para acabar el switch
case valor2:
    // Instrucciones caso 2
    break;
default: // opcional
    // Instrucciones si no se diera ningún caso
}
```

Ejercicio

■ Realizar una página con un script que calcule el valor de la letra de un número de identificación fiscal (NIF)

■ El algoritmo es el siguiente:

- Comprobar que el número está entre 0 y 999999999
- Calcula el resto de la división entera del número de DNI y el número 23
- Selecciona la letra dentro del array de letras siguiente:
var letras = ['T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X', 'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E', 'T'];
 - Esto es, si el resto de la división es 0, la letra del DNI es la T, si el resto es 1 la letra es la R, etc.

■ El script tiene que solicitar un número al usuario

- Para solicitar el número se puede usar la función **prompt()**:
var numero = prompt("Introduce tu número de DNI (sin la letra)");
- Si el número no es correcto, indicar un mensaje de error
- Si fuera correcto, calcular y mostrar el NIF completo

Control de flujo

■ Bucles

■ **for**

```
for ( inicialización; condición; actualización) {  
    // Instrucciones  
}
```

• Ejemplo:

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

■ **for in**

```
for ( indice in array ) {  
    // Instrucciones  
}
```

• Ejemplo:

```
for (i in array) {  
    procesa(array[i]);  
}
```

Control de flujo

■ Bucles

■ **while**

```
while( condición ){  
    // Instrucciones  
}
```

• Ejemplo:

```
while( true ) {  
    bucle_infinito();  
}
```

■ **do while**

```
do {  
    // Instrucciones  
} while( condición )
```

■ Sentencias para control de bucles

■ Salir del bucle **break**

■ Saltar a la siguiente iteración **continue**

Control de flujo

- Excepciones: **try..catch**

```
try {  
    // Código a ejecutar  
}  
catch(err) {  
    // Gestión de errores  
}
```

- Se puede lanzar una excepción con **throw**

- `throw excepcion`

Funciones

- **function ()**

```
function nombre_funcion ( arg1, arg2, ...){  
    // instrucciones  
    return; // o return resultado;  
}
```

- Entre paréntesis la lista de parámetros, sin tipo, separados por comas
- El tipo de resultado no se declara, y se devuelve con **return**

- Se pueden definir funciones anidadas

```
function hipotenusa(a, b) {  
    function cuadrado(x) { return x*x; }  
    return Math.sqrt(cuadrado(a) + cuadrado(b));  
}
```

Funciones

■ arguments

- El objeto *arguments* permite acceder a los argumentos de una función como un array
 - Los argumentos se acceden con `arguments[i]`
 - El número de argumentos se accede con la propiedad `length`

```
function max( ){  
    var m = Number.NEGATIVE_INFINITY;  
    for(var i = 0; i < arguments.length; i++)  
        if (arguments[i] > m) m = arguments[i];  
    return m;  
}
```

Ámbito de las variables

- Locales
 - Se definen dentro de una función con `var`
- Globales
 - Se definen fuera de cualquier función
 - O dentro de una función sin especificar `var`
- Dentro de una función una variable local prevalece sobre la global
- **Ejercicio:** ¿Qué mensajes se muestran con el siguiente código?

```
var m = "global";  
function muestraMensaje() {  
    m = "local";  
    alert(m);  
}  
  
alert(m);  
muestraMensaje();  
alert(m);
```

Objetos

- Se dice que JavaScript es un lenguaje basado en objetos
 - En JavaScript no se definen clases, solo objetos
 - Es un lenguaje basado en prototipos (no basado en clases)
 - Se pueden crear objetos copiando prototipos de otros objetos
- Un objeto en JavaScript es un conjunto de variables con un nombre
 - Las variables del objeto se denominan **propiedades**
 - Las propiedades pueden ser valores de cualquier tipo de datos: arrays, funciones y otros objetos
 - Las propiedades que son funciones se llaman **métodos**

Objetos

- Se puede crear un objeto directamente indicando sus propiedades:

```
persona=new Object();
persona.nombre="Juan";
persona.id= 12893;
```
- O en una sola instrucción, indicando las propiedades entre llaves:

```
var persona = { nombre: "Juan", id: 12893 }
```
- O definir un **constructor**

```
function persona(nombre, id) {
    this.nombre=nombre;
    this.id=id;
}
```

 - Y así crear varios objetos:

```
var juan=new persona("Juan", 12893);
var adela=new persona("Adela", 23782);
```
- Acceso a sus propiedades:

```
nombre = persona.nombre;      // dos formas de acceder a una
nombre = persona["nombre"];    // propiedad del objeto
```


Objetos

- Se pueden definir métodos para un objeto dentro del constructor

```
function persona(nombre, id) {  
    this.nombre=nombre;  
    this.id=id;  
  
    function renombra(nombre) {  
        this.nombre=nombre;  
    }  
}
```

- Y se invoca sobre el objeto:

```
var juan=new persona("Juan", 12893);  
juan.renombra("Juanjo");
```

Tipos de objetos

- Objetos del lenguaje
 - Object, Boolean, Number, Math, Date, String, Array, RegExp
- Objetos del navegador
 - Window, Navigator, Screen, Location, History, Timing, Cookies
- Objetos DOM
 - Core DOM
 - Node, NodeList, NameNodeMap, Document, Element, Attr
 - HTML DOM
 - Document, Events, Elements
 - Anchor, Area, Base, Body, Button, Form, Frame, Frameset, Image, Input, Link, Meta, Object, Option, Select, Style, Table, Textarea
- Objetos definidos por el usuario

JavaScript

Objetos del lenguaje

Objetos del lenguaje

- Dan soporte para manejar tipos básicos
- Todos los objetos del lenguaje tienen las propiedades
 - **constructor**
 - Devuelve la función que crea el objeto

```
objeto.constructor
```
 - Devolverá algo así

```
function Boolean() { [native code] }
```
 - **prototype**
 - Es un constructor que permite añadir propiedades y métodos al objeto
 - Se aplicará a todos los objetos de ese tipo

```
Boolean.prototype.nuevaFuncion=function() {  
    // código  
}
```
- Y los métodos
 - **toString()**: Devuelve una representación como string del objeto
 - **valueOf()**: Devuelve el valor primitivo (true/false, un número, etc.) del objeto

Objeto Boolean

- Permite convertir objetos no booleanos a booleanos
- Creación de un objeto booleano:

```
var unBooleano=new Boolean(otro);
```
- El valor será **false** si se crea con uno de los siguientes valores
 - **0**
 - **-0**
 - **null**
 - **""**
 - **false**
 - **undefined**
 - **NaN**
- En el resto de los casos el valor será **true**

Objeto Number

- Solo hay un tipo de números que se puede escribir con o sin decimales
 - Todos los números se almacenan con 64 bits
- Creación de un objeto Number:

```
var num = new Number(valor);
```
- Propiedades:
 - **MAX_VALUE**: mayor número posible (1.7976931348623157e+308)
 - **MIN_VALUE**: menor número posible (5e-324)
 - **NEGATIVE_INFINITY**: $-\infty$
 - **POSITIVE_INFINITY**: ∞
 - **NaN**: para indicar que el valor no es un número
- Métodos:
 - **toExponential(x)**: pone el número en notación científica (1.23e+3)
 - **toFixed(x)**: formatea el número con x decimales
 - **toPrecision(x)**: formatea el número con longitud x

Objeto Math

- Ofrece varias operaciones matemáticas
 - Constantes matemáticas
 - **Math.E**
 - **Math.PI**
 - Math.SQRT2: raíz cuadrada de 2
 - Math.SQRT1_2: raíz cuadrada de 1/2
 - Math.LN2
 - Math.LN10
 - Math.LOG2E
 - Math.LOG10E
 - Métodos
 - **round**(decimal): redondeo
 - **random**(): devuelve un número aleatorio entre 0 y 1
 - **max**(x, y)
 - **min**(x, y)

Objetos String

- Métodos sobre strings
 - **length**: número de caracteres de un string: `s.length`
 - Concatenación de strings: operador **+**
 - Al igual que en Java, si el primer operando es un string, los demás operandos se convertirán a strings para concatenarse
`var cad = "2"+2+2 → "222"`
 - **toUpperCase(), toLowerCase()**
`var m = "Juan";`
`var m2= m.toUpperCase(); // m2 = "JUAN"`
 - **charAt(posicion)**
 - **indexOf(caracter), lastIndexOf(caracter)**
 - Cuenta desde 0. Si no estuviera el carácter devuelven -1
`var posicion = m.indexOf('a'); // posicion = 2`
 - **substring(inicio, final)**
`var resto = m.substring(1); // resto = "uan"`
 - **split(separador)**
`var letras = m.split(""); // letras = ["J", "u", "a", "n"]`
`m="Hola Juan"; palabras=m.split(" ") // palabras=["Hola","Juan"]`

Objeto Date

- Proporciona la fecha y hora

```
new Date() // fecha y hora actual
new Date(milisegundos) //milisegundos desde 1 de enero 1970
new Date(string)
new Date(anno, mes, dia, horas, minutos, segundos, milisegundos)
```
- Métodos:
 - **getTime()**: devuelve el número de milisegundos desde 01.01.1970
 - **getFullYear()**: devuelve el año (cuatro dígitos)
 - **getDate()**: devuelve el día del mes (1..31)
 - **getDay()**: devuelve el día de la semana (0..6)
 - **getHours()**: devuelve la hora (0..23)
 - **getMinutes()**: devuelve los minutos (0..59)
 - Los equivalentes **setDate**, **setHours**, etc.
 - **setFullYear()**: cambia la fecha

```
d.setFullYear(2020,10,3);
```
 - **toUTCString()**: convierte la fecha a un string con formato de fecha de tiempo universal (Wed, 30 Jan 2013 07:03:25 GMT)

Arrays

- Colección de variables
 - Pueden ser todas del mismo tipo o cada una de un tipo diferente

```
var nombre_array = [valor1, valor2, ..., valorN];
var sin_inicializar = new Array(5);
```
 - Se accede a los elementos con `nombre_array[índice]`
 - índice es un valor entre 0 y N-1
- Propiedades y métodos
 - **length**: número de elementos de un array
 - **concat()**: concatenar los elementos de varios arrays

```
a1 = [1, 2, 3];
a2 = a1.concat(4, 5, 6); // a2 = [1, 2, 3, 4, 5, 6]
a3 = a1.concat([4, 5, 6]); // a3 = [1, 2, 3, 4, 5, 6]
```
 - **pop()**: elimina y devuelve el último elemento del array
 - **push(elemento)**: añade un elemento al final del array
 - **shift()**: elimina y devuelve el primer elemento del array
 - **unshift(elemento)**: añade un elemento al principio del array
 - **reverse()**: coloca los elementos del array en el orden inverso a su posición original

```
a1.reverse(); // a1 = [3, 2, 1]
```

Ejercicios

- Crea un script para visualizar un reloj en una página
- Crea un script que visualice el día de la semana
 - usa la función `getDay()` y un array con los nombres de los días de la semana

Objeto RegExp

- Para trabajar con expresiones regulares
 - Permite expresar patrones de caracteres y buscar correspondencias (*matching*) en un string
 - Una expresión regular consta de un patrón y modificadores:
`var er=new RegExp(patron,modificadores);`
 - Se puede crear también con la notación más sencilla:
`var expresion_regular=/patron/modificadores;`
- Los modificadores pueden ser
 - **i**: no diferencia mayúsculas de minúsculas
 - **g**: encuentra todas las correspondencias
- Operaciones sobre una expresión:
 - **test**(string) – devuelve true si se cumple el patrón en el string
 - **exec**(string) – devuelve el texto del texto correspondiente
 - O null si no hay ninguna correspondencia
- Ejercicio: ¿Qué devolverá el siguiente código?

```
var patron=new RegExp("e");
document.write(patron.test("JavaScript no es difícil"));
```

Objeto Global

- Hay un conjunto de propiedades y métodos que pueden accederse directamente
- Propiedades globales
 - **Infinity** – valor numérico que representa el infinito
 - **NaN** – valor que no es un número (*"Not a Number"*)
`Number.NaN`
 - **undefined** – una variable que no tiene asignado un valor

```
var variable;
if (variable===undefined) {
    // en este caso variable está indefinida
}
```

Objeto Global

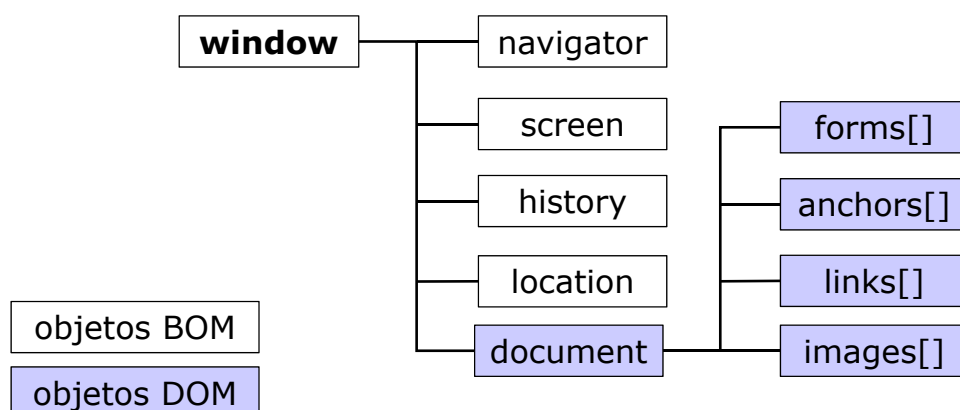
- Métodos globales
 - **eval**(string) – Evalúa una cadena de texto como si fuera un programa JavaScript
 - **parseInt**(string, base) – Convierte una cadena de texto a un número entero
 - base indica el sistema de numeración (2..36) (si no se indica se puede derivar del inicio del string ("0x" hex, "0" octal, o decimal)
 - Si no puede hacer la conversión devuelve `Number.NaN`
 - **parseFloat**(string) – Convierte una cadena de texto a un float
 - **isNaN**(valor) – Devuelve true si valor no es un número, false si lo es
 - **isFinite**(valor) – Devuelve true si su argumento no es NaN o Infinity
 - **encodeURIComponent**(uri) – Codifica los caracteres especiales de una URI excepto `, / ? : @ & = + $ #`
 - Para codificar también estos se usa `encodeURIComponent()`
 - **decodeURI**(uri_codificada) – Descodifica una URI codificada

JavaScript

Objetos del navegador

Objetos del navegador

- Browser Object Model (BOM)
 - JavaScript proporciona objetos que permiten interactuar con el navegador
 - No son parte del estándar pero casi todos los navegadores los implementan
- Jerarquía de objetos:



Objeto window

- Objeto que contiene todos los demás
 - Por ejemplo, el más usado: **window.document**
 - Aunque algunos navegadores permiten omitir en ocasiones la referencia a *window* por ser un objeto tan común que se sobreentiende
- Propiedades
 - **innerHeight** – altura interior de la ventana del navegador
 - **innerWidth** – anchura interior de la ventana del navegador
 - Para que funcione en versiones de IE 8 y anteriores, se puede poner:

```
var w=window.innerWidth || document.documentElement.clientWidth  
|| document.body.clientWidth;
```
- Métodos
 - **resizeTo**(ancho, alto) – cambia el tamaño de la ventana
 - **moveTo**(x, y) – mueve la ventana actual

Objeto window

- Aparte de la ventana del navegador se pueden crear otras ventanas
 - **open**(URL, nombre, parámetros)
 - Los parámetros son una lista de algunos de los siguientes elementos:
 - toolbar[=yes|no]
 - location[=yes|no]
 - directories[=yes|no]
 - status[=yes|no]
 - menubar[=yes|no]
 - scrollbars[=yes|no]
 - resizable[=yes|no]
 - width=pixels
 - height=pixels

```
var nuevaVentana=window.open("http://www.ucm.es");  
var otraVentana=window.open("", "", "width=200,height=100");  
otraVentana.document.write("<p>Esta es otra ventana</p>");  
otraVentana.focus();
```

- Y destruirlas
 - **ventana.close()**

Objeto window

- Ventanas de diálogo
 - **alert**(mensaje) – Muestra una ventana de alerta con un mensaje
 - **confirm**(mensaje) – Muestra una ventana de confirmación con los botones OK y Cancel
 - **prompt**(mensaje, valorPredeterminado) – Muestra una ventana de diálogo para solicitar una información
 - Se indica un mensaje
 - Se puede indicar un valor por defecto para el área de la respuesta
 - Como resultado se espera recibir un string
- Estos métodos se pueden invocar sobre el objeto window o directamente:

```
var valor=window.prompt("Introduzca el valor: ", "");  
var valor=prompt("Introduzca el valor: ", "");
```

Ejercicios

- Utiliza el objeto screen para obtener la dimensión de la pantalla y abre una nueva ventana en el centro de la pantalla
- Escribir un script con un botón para crear una nueva ventana. La nueva ventana tendrá a su vez un botón para cerrarla
 - En una ventana creada se puede escribir sobre el objeto document que contiene:

```
var ventana=open();  
ventana.document.write("<p>Un texto</p>");  
ventana.document.write("<p>Otro párrafo</p>");
```
 - Para asociar una función JavaScript al evento de pulsar un botón, se especifica al declarar el botón:

```
<FORM>  
  <INPUT TYPE="button" VALUE="Abrir Ventana" onClick="nueva();">  
</FORM>
```

 - En este ejemplo, nueva() es una función JavaScript declarada dentro de un elemento <script>

Objeto screen

- Contiene propiedades con la información de la pantalla del usuario
 - No es necesario indicar window.screen, basta con screen

```
<p>Propiedades de la pantalla:
<script>
  document.write("Anchura y altura total: ");
  document.write(screen.width + "x" + screen.height);
  document.write("<br>");
  document.write(" Anchura y altura disponible: ");
  document.write(screen.availWidth + "x" + screen.availHeight);
  document.write("<br>");
  document.write("Profundidad del color: ");
  document.write(screen.colorDepth);
  document.write("<br>");
  document.write("Resolución del color: ");
  document.write(screen.pixelDepth);
</script>
</p>
```

Objeto navigator

- No es muy útil porque se implementa de maneras bastante diferentes
 - Además es posible que el usuario configure el navegador para que declare que es otro, luego la información no es fiable

```
<div id="navegador"></div>
<script>
txt = "<h3>Propiedades del navegador:</h3>";
txt+= "<p>CodeName: " + navigator.appCodeName + "</p>";
txt+= "<p>Name: " + navigator.appName + "</p>";
txt+= "<p>Versión: " + navigator.appVersion + "</p>";
txt+= "<p>Cookies permitidos: " + navigator.cookieEnabled + "</p>";
txt+= "<p>Plataforma: " + navigator.platform + "</p>";
txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";
txt+= "<p>Lenguaje del sistema: " + navigator.systemLanguage + "</p>";
document.getElementById("navegador").innerHTML=txt;
</script>
```

Objeto history

- Para proteger la privacidad, este objeto tiene una funcionalidad bastante limitada, básicamente para avanzar o retroceder páginas
 - **history.length** – indica cuántas páginas están registradas (realmente sirve para saber si hay alguna anterior)
 - **history.back()** – carga la página precedente (si la hubiera)
 - **history.forward()** – carga la página siguiente (si la hubiera)
 - **history.go(número)** – carga la página de la lista hacia delante o atrás indicada por el número, según sea positivo o negativo

```
<script>
function atras(){
    window.history.back()
}
</script>
```

Objeto location

- Facilita la manipulación del URL actual y la posibilidad de recargar la página o redireccionar a otra página
- Propiedades
 - **location.href** – devuelve el URL
`document.write(location.href);`
 - **location.hostname** – devuelve el dominio del host web
 - **location.path** – devuelve el camino del fichero de la página actual
 - **location.port** – devuelve el puerto (p.ej., 80 o 443)
 - **location.protocol** – devuelve el protocolo usado (http:// o https://)
 - **location.search** – devuelve la parte del URL tras ? (incluido)
 - **location.hash** – devuelve el anchor (la parte del URL tras #, incluido)
- Métodos
 - **location.assign(url)** – carga la página indicada

```
function nuevoDoc() {
    window.location.assign("http://www.ucm.com")
}
```
 - **reload()** – recarga la página
 - Normalmente desde la caché, pero se puede forzar con `reload(true)`

Ejercicio

- Crea un script para que al pulsar a una referencia a una página aparezca una ventana de confirmación. En caso afirmativo se cargará la nueva página, y en caso negativo se mantendrá la actual
 - `window.confirm` permite sacar la ventana de diálogo
 - El objeto `location` permite cambiar de página
 - Habrá que asociar una función de script al evento de seleccionar una nueva página

Cookies

- Un cookie es una variable que se almacena en el navegador, que la enviará al servidor en las siguientes invocaciones que le envíe
 - Los cookies tienen una fecha de expiración
- Creación de un cookie

```
function creaCookie (nombre, valor, dias) {  
    var fechaExpiracion=new Date();  
    fechaExpiracion.setDate(fechaExpiracion.getDate() + dias);  
    var valorCookie=escape(valor) + ((dias==null) ? "" :  
        "; expira="+fechaExpiracion.toUTCString());  
    document.cookie=nombre + "=" + valorCookie;  
}
```

Cookies

■ Recuperación de un cookie

```
function recuperaCookie(nombre) {
    var i,x,y,cookies=document.cookie.split(";");
    for (i in cookies) {
        x=cookies[i].substr(0, cookies[i].indexOf("="));
        y=cookies[i].substr(cookies[i].indexOf("=")+1);
        x=x.replace(/^\s+|\s+$/g,"");
        if (x==c_name) {
            return unescape(y);
        }
    }
}
```

Cookies

■ Utilización del cookie

- Para invocar el método se puede hacer por ejemplo al cargar la página
<body onload="usaCookie()">

```
function usaCookie() {
    var usuario=getCookie("nombreusuario");
    if (usuario!=null && usuario!="") {
        alert("Bienvenido de nuevo, " + usuario);
    }
    else {
        usuario=prompt("Por favor, introduzca su nombre:", "");
        if (usuario!=null && usuario!="") {
            setCookie("nombreusuario", usuario,365);
        }
    }
}
```

Ejercicio

- Define una página que la primera vez solicite el nombre del usuario y las siguientes le salude sin necesidad de pedir el nombre
 - Para ello se almacena el nombre en un cookie
 - Probarlo en el navegador habitual

JavaScript

DOM

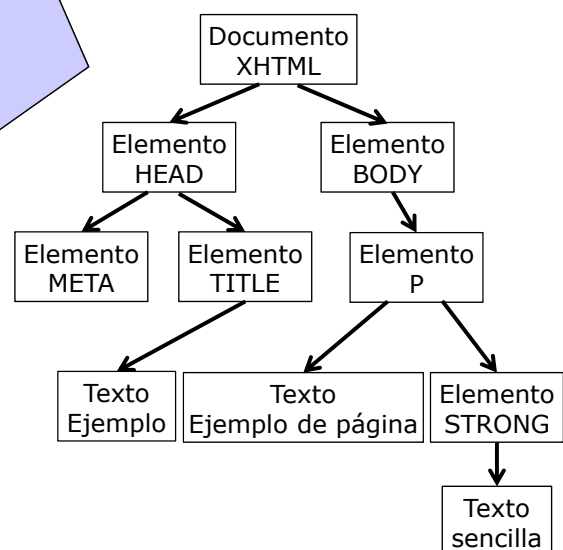
DOM

- Definido por W3C
- Modelo de Objetos del Documento (*Document Object Model*)
 - DOM define objetos y propiedades de los elementos HTML y XML, y los métodos para acceder a ellos
 - Representación de documentos HTML y XML
 - API para consultar y manipular los documentos (contenido, estructura, estilo)
- Los elementos de un documento se organizan en una jerarquía (árbol): jerarquía DOM
 - Los elementos del documento son los nodos del árbol
 - Las relaciones entre los nodos representan las interconexiones de los elementos
- El API DOM proporciona operaciones para poder acceder a estos objetos y manipularlos

DOM

- El navegador transforma el código del documento en un árbol DOM

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html
xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1" />
<title>Ejemplo</title>
</head>
<body>
<p>Ejemplo de página <em>sencilla</em></p>
</body>
</html>
```



Tipos de nodos

- Los nodos básicos de un documento HTML son:
 - **Document** – nodo raíz del que derivan todos los demás nodos del árbol
 - **Element** – representa cada una de las etiquetas
 - Es el único nodo que puede contener atributos
 - **Attr** –representa cada uno de los atributos de las etiquetas
 - Hay uno por cada par atributo=valor
 - **Text** – contiene el texto encerrado por una etiqueta XHTML
 - **Comment** – representa los comentarios incluidos en la página
- Otros tipos de nodos menos utilizados:
 - **DocumentType** – interfaz a las entidades definidas para el documento
 - **CDataSection** –una sección CDATA (que no analiza el parser)
 - **DocumentFragment** – un documento "ligero" (una parte)
 - **Entity** –una entidad
 - **EntityReference** –una referencia a una entidad
 - **ProcessingInstruction** –una instrucción de procesado
 - **Notation** –una notación declarada en la DTD

Acceso a nodos

- Acceso directo
 - **Acceso por ID:** `document.getElementById(id)`
 - Devuelve el objeto correspondiente al elemento que tenga el id especificado (solo puede haber uno) o *null* si no lo hubiera
 - Este es un método del objeto *document*

```
var x=document.getElementById("principal");
```

 - Devuelve el elemento con id="principal" dentro del documento
 - **Acceso por etiqueta:** `getElementsByTagName(etiqueta)`
 - Devuelve un array de nodos que tienen la etiqueta especificada
 - Este método se puede aplicar a cualquier nodo

```
var enlaces=x.getElementsByTagName("a");
```

 - Devuelve todos los enlaces <a> dentro del elemento x (id="principal")

Acceso a nodos

- Desde el nodo padre
 - Acceder al nodo raíz (document), y navegar por los hijos hasta el nodo deseado
 - Se utilizan propiedades y métodos de los nodos
 - Propiedades
 - **childNodes** – NodeList (array) que contiene todos los hijos del nodo
 - **firstChild** y **lastChild** –primer y último hijo de un nodo
 - **parentNode** – el padre del nodo
 - **nextSibling** y **previousSibling** –nodos siguiente y anterior en el mismo nivel
 - Métodos
 - **hasChildNodes()**

Métodos de los nodos

- Todos los nodos tienen un conjunto de métodos para manipular los hijos:
 - **appendChild()**
 - **insertBefore()**
 - **isSameNode()**
 - **removeChild()**
 - **replaceChild()**
- Y otros métodos para acceder a los atributos del nodo
 - **getAttribute("atributo")** – devuelve el valor del atributo
 - **setAttribute("atributo", "valor")** – permite modificar el valor del atributo, o añadir un nuevo atributo

Cambios en el documento y en elementos

- **document.write**(string) – para escribir en el documento:
`document.write("texto");`
`document.write(Date());`
`document.writeln("Salta a la siguiente línea");`
- Propiedad **innerHTML** – para cambiar el contenido de un elemento
`document.getElementById(id).innerHTML="nuevo código HTML"`
- Otra posibilidad, más elaborada:
`var padre=document.getElementById("elPadre");`
`padre.removeChild(padre.firstChild);`
- Cambio de un atributo de un elemento
`document.getElementById(id).atributo="nuevo valor"`
- Cambio del estilo
`document.getElementById(id).style.property="nuevo valor"`

Creación de un nuevo elemento

- Para crear un nuevo elemento hay que:
 - Crear el elemento: **createElement**(etiqueta) y **createTextNode**(string)
 - Añadirlo a un elemento existente (padre) con la operación **appendChild**(hijo)

```
<div id="seccion">
<p id="p1">Primer párrafo.</p>
<p id="p2">Segundo párrafo.</p>
</div>

<script>
/* Se crea el nuevo párrafo, incluyendo un nodo para el texto */
var nuevop=document.createElement("p");
var nodo=document.createTextNode("Un nuevo párrafo.");
nuevop.appendChild(nodo);          // el nuevo párrafo con su texto

/* Se añade el párrafo al final de la sección correspondiente */
var elemento=document.getElementById("seccion");
elemento.appendChild(nuevop);
</script>
```

Eliminación de un elemento

- Para eliminar un nuevo elemento hay que:
 - Localizar el padre del elemento
 - Eliminar el nodo hijo que corresponde al elemento con la operación **removeChild**(hijo)

```
<div id="seccion">  
<p id="p1">Primer párrafo.</p>  
<p id="p2">Segundo párrafo.</p>  
</div>
```

```
<script>  
var padre=document.getElementById("seccion");  
var hijo=document.getElementById("p1");  
padre.removeChild(hijo);  
</script>
```

```
// Más corto:  
var hijo=document.getElementById("p1");  
hijo.parentNode.removeChild(hijo);
```

JavaScript

Eventos

Eventos

- HTML DOM permite asociar código JavaScript a los eventos
- Cada evento tiene una propiedad (*event handler*) a la que como valor se le puede asignar la función que se invocará cuando se produzca el evento
 - Esta propiedad suele llevar el prefijo *on* seguido del nombre de evento

```
<html>
<head><title>Gestión de eventos</title>
<script type="text/javascript"><!--
function paginaCargada(){
    alert("La pagina ha sido cargada");
}
window.onload=paginaCargada;
//--></script>
</head>
<body>
<form>
<input type="button" value="Hola" onClick="window.alert('Hola')" />
</form>
</body>
</html>
```

Eventos

- El código a ejecutar por un evento se puede declarar en varios lugares
 - En el propio elemento (como un atributo)
`<h1 onclick="this.innerHTML='¡01e!'">Haz click en este texto</h1>`
 - En una función llamada desde el manejador del evento

```
<script>
    function cambiaElTexto(id) { id.innerHTML="¡01e!"; }
</script>
// ...
<h1 onclick="cambiaElTexto(this)">Haz click en este texto</h1>
```

 - La variable **this** se refiere al elemento que ha provocado el evento
 - Fuera del elemento
`document.getElementById("primero").onclick=function(){ cambiaElTexto(this) };`
 - Nota: Si se invoca `getElementById` en un script dentro del `<head>`, normalmente devolverá `null` porque no se habría construido aún el árbol DOM
 - La solución consiste en incluir ese código dentro de `onLoad` (ver ejemplo en la siguiente página)

Eventos

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Gestión de eventos</title>
<script type="text/javascript"><!--
window.onload = function() {
    document.getElementById("primero").onclick=function(){cambiaTexto(this)};
}

function cambiaTexto(id) { id.innerHTML="¡Ole!"; }
//--></script>
</head>
<body>
<h1 id="primero">Haz click en este texto</h1>
</body>
</html>
```

Eventos

- Sobre la página (se refieren al elemento <body>)
 - **onload** – Al cargar una página o una imagen
 - **onunload** – Cuando se abandona la página
 - **onresize** – Al modificar el tamaño de la ventana del navegador
- Sobre elementos
 - **onfocus** – Cuando el foco se dirige a un objeto
 - **onblur** – Cuando se cambia el foco a otro objeto
 - **onclick** – Cuando se hace click sobre un objeto
 - **ondblclick** – Cuando se hace doble click sobre un objeto
- Sobre formularios
 - **onsubmit** – Al pulsar el botón de envío de un formulario
 - **onchange** – Al cambiar el valor de un campo de un formulario
 - **onreset** – Al inicializar el formulario (borra todos los datos)

Eventos

- Teclado
 - **onkeydown** – Cuando se pulsa un tecla
 - **onkeyup** – Cuando se suelta una tecla
 - **onkeypress** – Cuando se pulsa y suelta una tecla
- Ratón
 - **onmousedown** – Cuando se pulsa un botón del ratón
 - **onmouseup** – Cuando se suelta el botón del ratón
 - **onmousemove** – Al mover el ratón
 - **onmouseover** – Cuando se mueve el puntero del ratón sobre un elemento (cuando entra al elemento)
 - **onmouseout** – Cuando el puntero del ratón abandona un elemento

Eventos

- Algunos eventos (onclick, onkeydown, onkeypress, onreset, onsubmit) ya tienen una acción por defecto que se puede modificar al definir un manejador de evento
- Algunas acciones pueden dar lugar a una sucesión de eventos
 - Por ejemplo, al pulsar sobre un botón de tipo "submit" se desencadenan los eventos onmousedown, onclick, onmouseup y onsubmit de forma consecutiva

Ejercicios

- Realiza un script que muestre las teclas que va pulsando el usuario
 - Se explica cómo hacerlo en http://librosweb.es/javascript/capitulo_6/obteniendo_informacion_del_evento_objeto_event.html
- Haz un script que vaya mostrando los elementos donde se hace click con el ratón

JavaScript

Formularios

Forms

- Al cargar la página el navegador crea automáticamente un array llamado **forms** que contiene la referencia a todos los formularios de la página
`document.forms`
- Dentro de cada form hay un array de elementos
`document.forms[i].elements[j] // i:0..n, j:0..m`
- Como la estructura de los formularios puede cambiar es más eficaz utilizar el atributo **id** en cada elemento del formulario que se quiere tratar

Forms

- Se pueden utilizar las siguientes propiedades de cada elemento:
 - **type**
 - Para los elementos de tipo `<input>` (text, button, checkbox, etc.) coincide con el valor de su atributo type
 - Para las listas desplegables normales (elemento `<select>`) su valor es select-one
 - Para las listas que permiten seleccionar varios elementos a la vez su valor es select-multiple
 - Para los elementos de tipo `<textarea>`, el valor de type es textarea
 - **form**: referencia directa al formulario al que pertenece el elemento
`document.getElementById("id_del_elemento").form`
 - **name**: valor del atributo name de XHTML (no se puede modificar)
 - **value**: valor del atributo value de XHTML
 - Para los campos de texto (`<input type="text">` y `<textarea>`) proporciona el texto que ha escrito el usuario
`<input type="text" id="linea" />`
`var valor = document.getElementById("linea").value;`
 - Para los botones se trata de saber el texto del botón seleccionado:

Forms

- Eventos más utilizados en el manejo de los formularios
 - **onclick** – cuando se pincha con el ratón sobre un elemento
 - Generalmente con los botones (button, submit, image)
 - **onchange** – cuando el usuario cambia el valor de un elemento de texto
 - Generalmente con entrada de tipo text o textarea
 - También se produce cuando el usuario selecciona una opción en una lista desplegable (<select>), al pasar el usuario al siguiente campo del formulario
 - **onfocus** – cuando el usuario selecciona un elemento del formulario
 - **onblur** – cuando el usuario pasa a otro elemento del formulario

Ejercicio

- Crea una función **validar()** para validar la entrada de datos de un formulario
 - En http://librosweb.es/javascript/capitulo_7/validacion.html hay varios ejemplos de cómo hacerlo
- Para evitar que se envíe un formulario varias veces conviene deshabilitar el botón de envío tras enviarlo una vez. Escribe un script para gestionar el envío del formulario:
 - Deshabilita el botón "Enviar"
 - Cambia el mensaje que muestra el botón de "Enviar" a "Enviando..."
 - Cuando se haya enviado genera una página nueva indicando que se ha enviado correctamente y muestra la información que se ha enviado

JavaScript

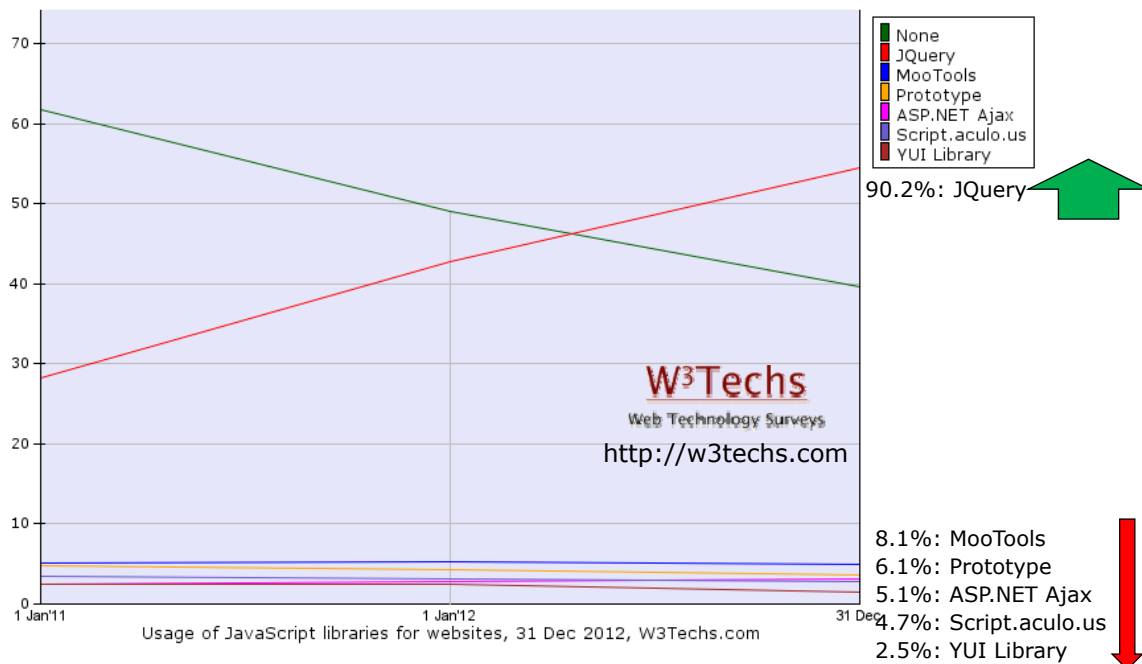
Librerías y frameworks

Librerías JavaScript

- jQuery
 - Usa selectores CSS para acceder y manipular los elementos HTML de una página web, lo que simplifica mucho el manejo del árbol DOM
 - Un framework de interfaz de usuario para gestionar los eventos
 - Múltiples plugins que facilitan el desarrollo de las aplicaciones
- Prototype
 - Manejo simple del árbol DOM
 - Incluye el uso de clases y herencia en JavaScript
- Moo Tools (My Object Oriented Tools)
 - Implementa el uso de clases
 - Proporciona facilidades de animación
- YUI - The Yahoo! User Interface Framework
- script.aculo.us – Efectos visuales

- Una buena lista: <http://javascriptlibraries.com/>

Librerías JavaScript



Tecnologías de script

- **AJAX** (*Asynchronous JavaScript And XML*)
 - Creación de aplicaciones Web interactivas
 - Las aplicaciones se ejecutan en el cliente
 - Pueden comunicarse asíncronamente con el servidor
 - El contenido de las páginas se actualiza sin necesidad de volver a cargarlas => Mayor dinamismo e interactividad
 - Comprende varias tecnologías:
 - **XHTML** y **CSS**: Presentación basada en estándares
 - **DOM**: Interacción y manipulación dinámica de la presentación
 - **XML**, **XSLT** y **JSON**: Intercambio y manipulación de información
 - **XMLHttpRequest**: Intercambio asíncrono de información
 - **JavaScript**: Unión del resto de tecnologías
 - En AJAX el cliente hace una petición al servidor por medio del objeto XMLHttpRequest
 - El servidor procesa la petición y devuelve una respuesta en XML en lugar de una página (X)HTML
 - El propio objeto XMLHttpRequest procesa dicha respuesta y actualiza únicamente las secciones necesarias de la página, evitando tener que recargarla por completo

Depuración de JavaScript

- Para depurar JavaScript
 - Con Firefox: consola JavaScript
 - Web Developer -> Error console (o con ctrl-mayusculas-J)
 - Con Aptana Studio
 - <https://wiki.appcelerator.org/display/tis/About+the+Debug+perspective>

Bibliografía

- <http://librosweb.es/javascript/>
- <http://www.w3schools.com/js/default.asp>
 - Una buena colección de ejemplos:
http://www.w3schools.com/js/js_ex_dom.asp
- Artículos varios: <http://javascript.about.com/>