



FENW	miw.etsisi.upm.es • 2
<h2>Índice</h2>	
<ul style="list-style-type: none"> <li>• Introducción             <ul style="list-style-type: none"> <li>◦ Conceptos básicos</li> <li>◦ Incorporación de JavaScript</li> </ul> </li> <li>• Lenguaje Javascript             <ul style="list-style-type: none"> <li>◦ Variables y operadores</li> <li>◦ Sentencias de control</li> </ul> </li> <li>• Objetos del Lenguaje             <ul style="list-style-type: none"> <li>◦ Arrays</li> <li>◦ Number, Boolean y Math</li> <li>◦ Date y String</li> </ul> </li> <li>• Eventos             <ul style="list-style-type: none"> <li>◦ Manejadores básicos de eventos</li> <li>◦ Tratamiento avanzado: event</li> </ul> </li> <li>• Objetos del Navegador             <ul style="list-style-type: none"> <li>◦ Elementos de formularios</li> <li>◦ Documento</li> <li>◦ Ventanas</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• Manipulación del DOM</li> <li>• Objetos de usuario             <ul style="list-style-type: none"> <li>◦ Definición: propiedades y métodos</li> <li>◦ Prototipos</li> <li>◦ Ampliación y reducción de los objetos</li> </ul> </li> <li>• Manipulación de las propiedades CSS con JS: DHTML             <ul style="list-style-type: none"> <li>◦ Acceso a las propiedades</li> <li>◦ Efectos visuales</li> </ul> </li> </ul>

## Introducción a JavaScript

- Permite introducir dinamismo en las páginas html aumentando sus capacidades.
  - Realizar cálculos aritméticos.
  - Mover, ocultar y mostrar elementos visuales.
  - Realizar validaciones de la información contenida en formularios.
  - Generar dinámicamente páginas html.
  - Interaccionar con objetos Active-X, Java,...
  - Mantenimiento del estado de las aplicaciones

## Introducción a JavaScript

- Integrado en HTML.
- Basado en objetos, que no orientado a objetos.
- Es un lenguaje levemente tipado.
- Permite el tratamiento de eventos.
- Interpretado por el navegador.
- Mejora el rendimiento de las aplicaciones web al descargar parte del proceso en el cliente.

# Lenguaje Javascript

- Documentación manejada:
  - European Computer Manufacturers Association(ECMA)  
<http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>
  - Tutoriales de HTML, CSS, JavaScript  
<http://www.w3schools.com/>
  - Libros gratuitos sobre WEB <http://www.librosweb.es/>
  - *Learning JavaScript* Shelley Powers O'Reilly 2009
  - *JavaScript: The Good Parts* Douglas Crockford O'Reilly 2008

## JavaScript embebido en HTML

- El código JavaScript puede aparecer en una página HTML en 4 formas distintas:
  - Entre las etiquetas `<script>` y `</script>`

```
<script type="text/javascript">
  Código JavaScript
</script>
```
  - Importando código javascript de un fichero externo.

```
<script src="libreria.js"></script>
```

## JavaScript embebido en HTML

- El código JavaScript puede aparecer en una página HTML en 4 formas distintas:

- Como respuesta o tratamiento de un evento

```
<input type="button" value="botón"
      onclick="Código JavaScript">
```

- Como recurso a obtener al pulsar un enlace. Es un nuevo URL llamado URL javascript

```
<a href="javascript:Código JavaScript">
  enlace</a>
```

## Ocultación de JavaScript

- Los navegadores muy antiguos no incluyen JavaScript.
- Para evitar que se muestre el código JavaScript en los navegadores antiguos se pone entre comentarios html

```
<script type="text/javascript">
<!-- Ocultación en navegadores antiguos
  Código javascript
-->
</script>
```

- La ejecución de JavaScript puede ser deshabilitada por el usuario en su navegador

## Comentarios de código JavaScript

- Comentarios hasta final de línea `// ...`  
`// Esto es un comentario hasta final de línea`
- Comentarios de varias líneas `/*...*/`  
`/* Esto es un comentario de  
varias líneas */`

## Secuencia de instrucciones

- Las instrucciones de una secuencia se separan por `;` ó **retornos de carro** y se ejecutan de arriba a abajo.

```
instrucción 1;  
instrucción 2;  
.....  
instrucción N;
```

- Es recomendable utilizar siempre `;` por analogía con otros lenguajes y permitir que el código se puede comprimir para que ocupe menos espacio.

## ECMA 6: Constantes

- Para definir una constante basta con anteponer la palabra reservada `const` a una asignación de un valor a una variable.

```
const PI = 3.14;  
const SEPARADOR_FECHAS = "-";
```

- No pueden ser re-declaradas ni cambiar su valor por reasignación
- Siguen las mismas reglas de ámbito que las variables
- Atención: si el valor asignado es un objeto, el objeto sí puede cambiar, aunque la constante no

## Declaración de variables

- Se declaran utilizando la palabra `var`.
- Antes de utilizar una variable en JavaScript no es necesario declararla.
- En la declaración las variables pueden ser inicializadas asignándolas un valor por medio del operador `=` (`nombredevariable = valor`).
- Javascript tiene un tipado dinámico en función del contenido almacenado en la variables

## Ejemplos de variables

### ▪ Ejemplos:

```
variable = "hola";
var cinco = 5;
cuatro = 4;
var nombre = "Daniela";
```

### ▪ Identificadores en JavaScript:

- Primer carácter debe ser (a-z ó A\_Z) o guión bajo ( \_ ) o dólar (\$)
- Resto caracteres (a-z o A-Z o 0-9 o \_)
- No pueden tener espacios en blanco
- No pueden coincidir con las palabras reservadas (`var`, `function`, `for`, `if`,...)
- Es **sensible** a las mayúsculas y minúsculas

## Tipos de las variables

### ▪ JavaScript tiene los siguientes tipos de variables:

- Numéricas: enteras ó reales.
- Cadenas de caracteres: contienen secuencias de caracteres.
- Booleanas: sólo dos valores `true` ó `false`.
- null: variable a la que se le ha asignado el valor `null`
- undefined: valor de variable declarada pero haber recibido un valor por medio de una asignación
- Objetos: contienen referencias a un objeto.

### ▪ JavaScript también tiene objetos propios del lenguaje para números, cadena de caracteres y booleanos

- Una variable que contenga un número, cadena de caracteres o booleano puede utilizar los métodos de los objetos correspondientes Ej: `"Hola".toUpperCase()` ;

## Valores de tipo numérico

- Variables numéricas:

- Enteras:

- Decimal: 2, -20, 10
- Octal: 02, -024, 012
- Hexadecimal: 0x2, -0x14, 0xA

- Reales:

- Notación decimal: 3.11, -0.0011, 45.000
- Notación científica: 311E-2, -11E-4, 45E3

## Valores de tipo string

- Su valor van con entrecomillado simple

`'Julio delegado', '12', 'En un lugar de la...'`

- Su valor también puede ir con entrecomillado doble

`"Daniela", "12", "En un lugar de la..."`

- Algunos caracteres aparecen escapados adoptando un significado especial

<code>\b</code> Espacio hacia atrás	<code>\f</code> Alimentación de línea
<code>\n</code> Nueva línea	<code>\r</code> Retorno de carro
<code>\t</code> Tabulación	<code>\\</code> Backslash
<code>\'</code> Comilla simple	<code>\"</code> Comilla doble



## Tipado de los datos

- JavaScript no es un lenguaje fuertemente tipado. El tipo de valores que una variable puede albergar puede cambiar durante la ejecución, es dinámico.

```
a = 2;           // variable numérica
a = "dos";       // variable cadena
a = true;        // variable booleana
```

- Existe conversión automática de tipos según las operaciones empleadas para manejar los datos

## Operadores Aritméticos

- Se utilizan para hacer cálculos aritméticos básicos.

- Suma +
- Resta -
- Multiplicación \*
- División /
- Resto de la división entera %
- Incremento de una variable ++
- Decremento de una variable --

```
variable = A + 3 * 2;
variable ++;
variable= -- B % 2
```

## Funciones numéricas

- **isNaN(valor)** función que comprueba si el valor es NaN. Devuelve **true** si no es un número y **false** cuando lo es
- **parseInt(cadena, base)**, examina la cadena y trata de convertirla en un número en la base especificada como parámetro. Esta función devuelve NaN (Not a Number) si no puede obtener un número
- **parseFloat(cadena)** igual que **parseInt** pero con números reales
- **isFinite()** devuelve **true** si el valor no es infinito o NaN

## Operadores de cadenas

- Se emplea en la manipulación de cadenas.

- concatenación de cadena **+**

```
saludo = "Buenas " + 'Tardes';
Variable = saludo + ", alumnos";
```

Observación: El significado del operador **+** estará determinado por su contexto:

```
a = 1 + 2      // a contiene 3
a = "1" + 2    // a contiene 12
a = 1 + "2"    // a contiene 12
a = "1" + "2"  // a contiene 12
```

## Operadores de relación

- Comparan dos valores devolviendo **true** o **false** en función de que se cumpla la relación indicada
    - Operador igual-que ==
    - Operador no-igual !=
    - Operador mayor-que >
    - Operador mayor-o-igual-que >=
    - Operador menor-que <
    - Operador menor-o-igual-que <=
    - Operador identidad ===
    - Operador no-identidad !==
- ```
menor_de_edad = edad < 18;
encontrado = clave == 'PeTeTe';
```
- Si ambos operadores son **objetos**, los operadores comparan las referencias de los objetos.

## Operadores lógicos

- Se aplican sobre operandos booleanos.
  - Operador not !
  - Operador and &&
  - Operador or ||

```
joven = (edad >= 18) && (edad <= 30);
correcto = !(a < b) || (c == d);
```

## Operadores de asignación y aritméticos

- $x = x + y$
- $x = x - y$
- $x = x * y$
- $x = x / y$
- $x = x \% y$

- $x += y$
- $x -= y$
- $x *= y$
- $x /= y$
- $x \% = y$

## Precedencia de Operadores

- La precedencia de operadores de menor a mayor valor es:

- Asignación

$=$   $+=$   $-=$   $*=$   $/=$   $\%=$   
 $<<=$   $>>=$   $>>>=$   $\&=$   $\^=$   $|=$

- Condicional

$?:$

- "o" lógica

$||$

- "y" lógica

$\&\&$

- "o" nivel bit

$|$

- "xor" nivel bit

$\^$

- "y" nivel bit

$\&$

## Precedencia de Operadores

- La precedencia de operadores de menor a mayor valor es:

|                            |             |
|----------------------------|-------------|
| ▫ Igualdad                 | == !=       |
| ▫ Comparación              | < <= > >=   |
| ▫ Desplazamiento           | << >> >>>   |
| ▫ Suma/Resta               | + -         |
| ▫ Multiplicación/ División | * /         |
| ▫ Negación/ Incremento     | ! ~ - ++ -- |

Con `()` se puede cambiar la precedencia.

## Ventanas de alerta

- JavaScript nos ofrece la posibilidad de comunicarnos sucesos en la aplicación Web por medio de ventanas de alerta

```
alert('Esta es una ventana de Alerta');
```

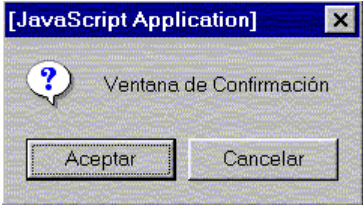


FENW miw.etsisi.upm.es • 27

## Ventanas de confirmación

- Las ventanas de confirmación devuelven un valor booleano según la elección realizada por el usuario

```
conf = confirm("Ventana de Confirmación");
```



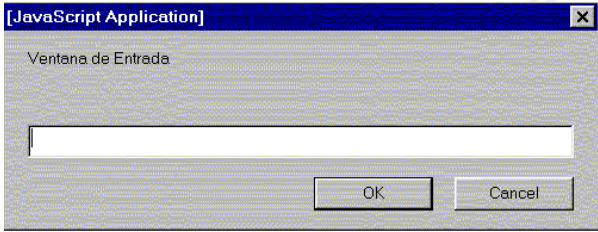
The image shows a standard JavaScript confirm dialog box. The title bar reads '[JavaScript Application]'. The main content area has a question mark icon and the text 'Ventana de Confirmación'. At the bottom, there are two buttons: 'Aceptar' (Accept) and 'Cancelar' (Cancel).

FENW miw.etsisi.upm.es • 28

## Ventanas de entradas de datos

- Invitan al usuario a introducir una cadena de caracteres

```
entrada = prompt("Ventana de Entrada", "");
```



The image shows a standard JavaScript prompt dialog box. The title bar reads '[JavaScript Application]'. The main content area has the text 'Ventana de Entrada' and a single-line text input field. At the bottom, there are two buttons: 'OK' and 'Cancel'.

FENW

miw.etsisi.upm.es • 29

## Operador Condicional

- La sintaxis del operador condicional es:

```
if (expresión condicional) {  
    Código si se cumple la condición  
}  
else {  
    Código si no se cumple la condición  
}
```

FENW

miw.etsisi.upm.es • 30

## Operador Condicional

- La sentencia *if* puede no tener parte *else*.
- Sentencias *if* anidadas, permiten colocar una sentencia *if* dentro de otra.
- No es necesario escribir las llaves (tanto en la parte *if*, como en la *else*) cuando el código esté formado por una sola instrucción.
- Debe tener especial cuidado en la utilización correcta de las llaves cuando haya varias sentencias anidadas

## Operador condicional terciario

- La expresión condicional siguiente:

```
if (condición)
    variable = Valor1;
else
    variable = Valor2;
```

es equivalente al operador condicional ternario

```
variable = (condición) ? Valor1 : Valor2;
```

## Bucle for

- Bucle for:

```
for (expresión inicial ;
    expresión condicional ;
    expresión de actualización)
{
    Código JavaScript que se repite
}
```

**expresión inicial:** inicialización de las condiciones o variable de gobierno del bucle

**expresión condicional :** condición de continuación en el bucle.

**expresión de actualización:** actualización de las condiciones o variables del bucle.



FENW miw.etsisi.upm.es • 33

## Bucles while y do while

- Bucle **while**:  

```
while (expresión condicional){  
    código JavaScript que se repite  
}
```
- Bucle **do while**:  

```
do {  
    Código JavaScript que se repite  
}while (expresión condicional)
```

FENW miw.etsisi.upm.es • 34

## Interrupción de un bucle

- Un bucle puede detenerse desde dentro con la sentencia **break**. Esta sentencia hace que pase a ejecutarse la siguiente línea de código después del bucle.
- **break** puede utilizarse con etiquetas para indicar la parte del bucle que va a interrumpirse.
- La sentencia **continue** es una forma de controlar un bucle de forma que reinicia el proceso inmediatamente antes del principio de bucle y empieza el siguiente ciclo.
- **continue** vuelve a la evaluación de la condición en un bucle **while** o a expresión de actualización de un bucle **for**

## Sentencia switch

- Su sintaxis es:

```
switch (expresion){  
    case etiqueta1:  
        sentencia1;  
        break;  
    case etiqueta2:  
        sentencia2;  
        break;  
    ...  
    default: sentencia por defecto;  
}
```

## Sentencia switch

- **switch** evalúa la expresión y luego compara el resultado con las etiquetas que hay definidas en la parte **case**.
- Si hay un **break**, éste hace que el programa salte a la siguiente instrucción a continuación del **switch**.
- Si no encuentra ninguna similitud ejecuta la sentencia que exista en la etiqueta **default**.

## ECMA 6: Bucle for ... of

- El bucle for ... of crea un bucle sobre objetos iterables (Array, Map, Set, String, arguments, con una iteración sobre cada uno de sus elementos

```
for (elemento of objetoIterable){
    //tratamiento de elemento
}
```

- Ejemplo:

```
let elArray = [1,3,5,7,9];
let resultado = 0;
for (let elemento of elArray){
    resultado = resultado + elemento;
}
console.log (resultado); // === 25
```

## Objeto Array

- Para utilizarlo sólo hay que crear una instancia del objeto matriz y se le asignan valores. El objeto se llama **Array**.
- Para crear instancias se usa la palabra clave **new**.  
Ej: `var vector = new Array()`
- El objeto **Array** comienza en el índice 0.
- `var info = new Array(3)` crea elementos que tienen por índice los valores 0, 1, 2.
- `var info = new Array("uno", 2)` crea una array con 2 elementos que contiene los valores "uno" y 2.
- También se puede crear un array literal encerrando entre corchetes una lista de valores  
Ej: `var vector = ["uno", 2]`  
`var otrovector = [];`
- La propiedad **length** indica cuántos elementos tiene la instancia del array.
- El array crece automáticamente según se necesiten nuevas posiciones.

FENW

miw.etsisi.upm.es • 39

# Objeto Array

- Script que muestra los elementos de un array.

miw

```
<script type="text/javascript">
  var longitud, i;
  var a = new Array(1,2,3,4,"cinco");
  longitud = a.length;
  document.write("<BR> La longitud es: " +
                  longitud + "<BR>");
  for (i = 0; i < longitud; i++)
    document.write ("El elemento " + i +
                    " es " + a[i] + "<BR>")
</script>
```

FENW

miw.etsisi.upm.es • 40

# Objeto Array

- delete a[i] asigna el valor null al elemento situado en la posición i del vector
- Pueden definirse arrays de más de una dimensión.
  - Para definirlos basta con crear arrays para cada una de las dimensiones del array.

miw

```
var f1 = new Array (00, 01)
var f2 = new Array (10, 11)
var f3 = new Array (20, 21)
var datos = new Array (f1,f2,f3)
```

|    |    |
|----|----|
| 00 | 01 |
| 10 | 11 |
| 20 | 21 |

- El acceso a una matriz bidimensional se hace de la siguiente forma:  
nombrevariable[filas][columna]

## Objeto Array

- `concat(array1, array1,...)` devuelve el resultado de concatenar el array con `array1, array1,...`
- `join(separador)` devuelve un String con todos los elementos del vector separados por el String `separador`
- `pop()` elimina el último elemento del array y lo devuelve
- `push(elem1, elem2,...)` añade los elementos `elem1, elem2,...` por el final del array y devuelve el nuevo tamaño
- `reverse()` invierte el orden de los elementos del array
- `shift()` elimina el primer elemento del array y lo devuelve
- `slice(principio, final)` devuelve la parte del array comprendida entre principio y final. Si principio es negativo se cuenta desde el final del array
- `sort(criterio)` ordena los elementos del array. Opcionalmente se puede indicar un `criterio` de ordenación
- `splice(indice,cantidad, elem1, elem2,...)` Elimina del array tantos elementos como indique `cantidad` desde `indice` y los sustituye por `elem1, elem2,...`
- `unshift((elem1, elem2,...)` añade los elementos `elem1, elem2,...` por el principio del array y devuelve el nuevo tamaño

## Acceso a los elementos de una página (I)

- Para acceder a un elemento hay que obtener una referencia al mismo y después acceder a sus propiedades o métodos:
  - `document.getElementById("identificador")` devuelve una referencia al elemento correspondiente con el elemento HTML que tiene el atributo `id` igual al valor `"identificador"`
- También se pueden obtener arrays de referencias a elementos que tengan algo en común (nombre en el caso de un "radio button" o que tengan la misma etiqueta HTML:
  - `document.getElementsByName("nombre")`, devuelve un array de referencias a los elementos correspondientes con el atributo `name` igual al valor `"nombre"`
  - `document.getElementsByTagName("etiqueta")`, devuelve un array de referencias a los elementos correspondiente con la `"etiqueta"` HTML

## Acceso a los elementos de una página (II)

- Ejemplo: cada una de las imágenes (<img>) de un documento html tienen un objeto Javascript asociado.
- Para cambiar una imagen hay que modificar el valor de la propiedad **src** del objeto imagen.

- Ejemplo:

- Html

```

```

- JavaScript

```
document.getElementById("myimagen").src = "otra.gif";
```

- También se puede utilizar el DOM1:

```
document.miimagen.src = "otra.gif";
```

- Se recomienda la utilización del método **getElementById()**

## Acceso a los elementos de una página (III)

Acceso a los valores de un input de tipo "text" :

- Html

```
<form name="miformulario">
  <input type="Text" name="micaja" id="idcaja">
</form>
```

- Javascript

```
document.getElementById("idcaja").value = "Hola";
alert(document.getElementById("idcaja").value);
```

o

```
var el_input = document.getElementById("idcaja");
el_input.value = "Hola";
alert (el_input.value);
```

## Funciones

- En JavaScript el programador puede definirse funciones propias.
- Una función es un bloque de sentencias en JavaScript que se agrupan bajo un mismo nombre y que permite que puedan ser usadas múltiples veces sin tener que ser reescritas.
- Las funciones en JavaScript son objetos por lo que una función puede tener propiedades y métodos!!

## Definición de funciones

- **Función declarativa.** La función es una sentencia por sí misma que comienza con la palabra reserva `function` seguida del nombre de la función, de los parámetros entre paréntesis y de código que implementa la función. Estas definiciones se parsean una sola vez.
 

```
function nombre(parámetros) {
    Código de la función
}
```
- **Función anónima.** Se parsean una única vez. Se asigna a una variable y no tiene nombre. El formato de la definición es:
 

```
var func = function (parámetros) {
    Código de la función
}
```
- **Literal Función o expresión función.** Se crean por medio del constructor `Function`. Se parsean cada vez que se accede a ellas y no tienen un nombre específico
 

```
var variable = new Function ("param1",... , "paramN",
                              "Código de la función");
```

## Devolución de valores en funciones

- Una función devuelve el valor que se especifica en la sentencia **return** (cadena, número, booleano u objeto).
- Una función puede utilizar varias sentencias **return** si lo exige el programa, aunque SÓLO debe existir uno por cada grupo de sentencias.

## Invocación de funciones

- Las funciones se invocan por su nombre seguido de la lista de parámetros entre paréntesis.  
`nombre (parámetros)`
- Las funciones pueden declararse en cualquier parte de un documento HTML.
- Antes de utilizar una función ha de ser declarada.
- Un script se carga por completo en memoria antes de ejecutarse.



## Parámetros pasados a las funciones

- Los parámetros se pasan:
  - Por **valor**: Se pasa copia de los tipos de datos cadenas, números y booleanos.
  - Por **referencia**: De los objetos se pasa una referencia no una copia.
- El array **arguments** contiene todos los parámetros recibidos por la función

```
function sumaGenerica() {
    var acumulador = 0;
    for (var i=0; i<arguments.length;i++)
        acumulador += arguments[i]
    return acumulador;
}
```

## Parámetros pasados a las funciones

- Una función invocada con mas parámetros de los definidos ignora los parámetros sobrantes
- Una función invocada con menos parámetros de los definidos trabaja con valores indefinidos para los parámetros faltantes.
- ⦿ Como las funciones son objetos, una función se puede pasar como parámetro a otra

```
function parametro(x, y){return (x*y)}

function procesa(vector, valor, funcion){
    for (var i=0; i < vector.length; i++)
        vector[i] = funcion(vector[i],valor);
}

vector = new Array(1, 2, 3);
procesa(vector, 10, parametro);
```

## Ámbito de las variables

- Todas las declaradas dentro del cuerpo de la función por medio de la palabra reservada **var** son locales
- Si no declaran las variables por medio de la palabra reservada **var** dentro de la función son globales
- Las funciones se pueden anidar. La función interior tiene acceso a las variables locales de la función exterior

```
function exterior(p1){
  var local = 10;
  function interior(p2){
    return local + p1 + p2;
  }
  alert(interior(20))
}
```

```
exterior(30) // ventana de alerta con el valor 60
```

## ECMA 6: Variables de ámbito de bloque

- Se pueden definir variables de ámbito de bloque, que evitan el denominado efecto “hoisting” mediante la palabra reservada **let**

```
for (let i = 0; i < 5; i++){
  let x = 2 * i;
  console.log(i);
  console.log (x);
}
console.log(x); // Error: x is not defined
```

- También se pueden definir funciones dentro de un bloque y su ámbito será ese bloque

## ECMA 6: Funciones “flecha” (arrow)

- Se puede usar una nueva sintaxis para la definición de funciones denominadas funciones “arrow”

```
var profesores = ['Santiago', 'Pedro', 'Manuel'];

profesores.forEach(function(valor) {
    console.log (valor + " es profesor");
});
```

Es equivalente a:

```
profesores.forEach(valor => {
    console.log (valor + " es profesor");
});
```

- Son siempre funciones anónimas

## ECMA 6: Funciones “flecha” (arrow) (ii)

- Si se necesita más de un parámetro:

```
var profesores = ['Santiago', 'Pedro', 'Manuel'];

profesores.forEach(function(valor, indice){
    console.log ("Posicion %s : %s", indice, valor);
} );

profesores.forEach( (valor,indice) => {
    console.log ("Posicion %s : %s", indice, valor);
} );
```

- No se pueden usar si se necesita acceso al objeto *arguments*

## Programación en modo estricto

- “*use strict*”:
- Afecta al ámbito en que haya sido definido, pero hay que ponerlo al inicio
- Lo soportan los nuevos navegadores (en los antiguos es una cadena de texto que se evalúa y no genera errores)
 

```
“use strict”
X = 10;
→ ERROR
```
- No se permite:
  - Utilizar variables no declaradas previamente
  - No se pueden eliminar variables o propiedades con *delete*
  - No se pueden repetir identificadores de parámetros *func (x,x){}*
  - No se puede utilizar el método *eval()* para la creación de variables
  - No se permite octal ni la “barra de escape”
  - No se puede utilizar *with()*
  - No se puede usar el identificador *arguments* ni *eval*
  - No se pueden utilizar las palabras reservadas de “futuro”
  - El valor de *this* no referencia al objeto global si está sin definir

## Herramientas de depuración

- Utilización “obligada” de herramientas de depuración como *Firebug* o las herramientas de desarrolladores de los navegadores (F12). – Utilización de la consola y de la visualización de valores en tiempo de ejecución
- Se puede utilizar *console.log(mensaje)* para realizar trazas de depuración en la consola
- También *console.info()*, *console.error()*, *console.debug()*
- Utilización de herramientas como *JsLint*:
- Existen editores que integran *JsLint* (ej. brackets)
- Comprueba la calidad del código generado, ayudando a evitar errores de programación
  - <http://www.jshint.com/>
  - Trabaja con muchas de las recomendaciones aquí expuestas

## Objetos del lenguaje

- Los tipos primitivos `string`, `number` y `boolean` tienen su correspondencia con objetos. Cuando sobre un tipo primitivo se invoca un método, JavaScript lo trata como si fuese un objeto
- Objetos Incorporados
  - `Array()`
  - `Boolean()`
  - `Date()`
  - `Math()`
  - `Number()`
  - `String()`
  - `RegExp()`
  - `Set()`
  - `Map()`

ECMA 6

## Objeto Boolean

- ⦿ Se usa para convertir valores no booleanos en booleanos
- ⦿ Valores:
  - `var variable= new Boolean(cadena)`
  - `Boolean()` -> devuelve `false`
  - `Boolean(cadena)` si `cadena` es : `0`, `nulls`, `false` o texto sin comillas o comillas vacías -> devuelve `false`
  - `Boolean(cadena)` si `cadena` es : `true` o texto con comillas -> devuelve `true`

## Objeto Number

### ● Propiedades estáticas:

- `Number.MAX_VALUE` número positivo mayor en Javascript
- `Number.MIN_VALUE` número positivo menor en Javascript
- `Number.NaN` Representa **Not-a-Number**
- `Number.NEGATIVE_INFINITY` Representa infinito negativo
- `Number.POSITIVE_INFINITY` Representa infinito

### ● Métodos:

- `toExponential(num)` devuelve una cadena con la representación del número en notación exponencial
- `toFixed(num)` devuelve una cadena con la representación del número en notación exponencial
- `toPrecision(num)` devuelve una cadena con la representación del número con una precisión determinada

## Objeto Date

- Permite trabajar con fechas y horas. Debe ser creada una instancia para poder trabajar con este objeto.

### ▪ Formas de crear un objeto **Date**

- `fecha = new Date(milisegundos)`
- `fecha = new Date()`
- `fecha = new Date("mes_en_inglés día, año  
horas:minutos:segundos")`
- `fecha = new Date(aa,mm,dd)`
- `fecha = new Date(aa,mm,dd,hh,mm,ss)`

## Objetos Date

### ▪ Métodos del objeto Date

- `getDate()` devuelve el día del mes (1-31)
- `getDay()` devuelve el día de la semana (0 domingo, 1 lunes,...,6 sábado)
- `getHours()` devuelve la hora (0-23)
- `getMinutes()` devuelve los minutos (0-59)
- `getMonth()` devuelve el mes (0 -11)
- `getSeconds()` devuelve los segundos (0-59)
- `getTime()` devuelve los milisegundos transcurridos desde la hora 0 del 1/1/1970)
- `getFullYear()` devuelve el año
- `toLocaleString()` devuelve la fecha como una cadena con el formato particular del ordenador

## Objeto Date

### ▪ Métodos del objeto Date

- `setDate(día)` cambia el día del mes.
- `setHours(hora)` cambia las horas
- `setMinutes(minutos)` cambia los minutos.
- `setMonth(mes)` cambia el mes.
- `setSeconds(segundos)` cambia los segundos.
- `setTime(milisegundos)` cambia la fecha a la correspondiente a los milisegundos transcurridos desde el 1/1/1970 00:00:00.
- `setYear(año)` cambia el año.

## Temporizadores

- JavaScript incorpora varios tipos de temporizadores que permiten ejecutar acciones después de transcurrido un cierto intervalo de tiempo.
- `setInterval()` ejecuta periódicamente una acción después de que transcurran los milisegundos que recibe como parámetro.

```
setInterval(Nombre_funcion,milisegundos)
```

## Objeto Math

- Proporciona métodos para hacer cálculos matemáticos.
- Contiene constantes matemáticas: número e, pi, logaritmo en base 10, raíz cuadrada de 2, etc.
- Todas las propiedades y métodos son estáticos

- Propiedades del objeto `Math`

- `E` valor del número e
- `LN2` valor del Logaritmo neperiano de 2
- `LN10` valor del Logaritmo neperiano de 10
- `LOG2E` valor del Logaritmo en base 2 de E
- `LOG10E` valor del Logaritmo en base 10 de 2
- `PI` valor del número PI
- `SQRT1_2` valor de la raíz cuadrada de 0,5
- `SQRT2` valor de la raíz cuadrada de 2

Ejemplo:

```
numeroEpornumeroPI = Math.E * Math.PI
```



## Objeto Math

### ▪ Métodos del objeto `Math`

- `abs(X)` devuelve el valor absoluto de X.
- `acos(X)` devuelve el arcocoseno de X.
- `asin(X)` devuelve el arcoseno de X.
- `atan(X)` devuelve el arcotangente de X.
- `ceil(X)` redondea el número al inmediatamente superior.
- `cos(X)` devuelve el coseno de X.
- `exp(X)` devuelve E elevado al numero X.
- `floor(X)` redondea el número al inmediatamente inferior.
- `log(X)` devuelve el logaritmo neperiano de X.

## Objeto Math

### ▪ Métodos del objeto `Math`

- `max(X,Y)` devuelve el mayor de dos números.
- `min(X,Y)` devuelve el menor de dos números.
- `pow(X,Y)` devuelve X elevado a Y.
- `random()` devuelve un número aleatorio entre 0 y 1.
- `round(X)` redondea el número X al entero más cercano.
- `sin(X)` devuelve el seno de X.
- `sqrt(X)` devuelve la raíz cuadrada de X.
- `tan(X)` devuelve la tangente de X.

## Objeto String

- Se crean por medio del constructor `String("literal")`  
Ej: `cadena = new String("mi cadena");`
- Tanto los `String` literales como los objetos tienen la propiedad `length` que determina el número de caracteres del mismo.
- `String` tiene el método estático `fromCharCode(n1,n2,...)` que crea una cadena a partir de los caracteres asociados a los códigos Unicode `n1, n2`,
- Los `String` literales como objetos tienen los siguientes métodos:
  - `big()`, `blink()`, `italics()`, `small()`, `sub()`, `sup()`, `strike()`, `link(url)`, `anchor(nombre)`, `fontcolor(color)` y devuelven una cadena con el elemento HTML correspondiente al `String`  
Ej: `"hola".italics()` es la cadena `"<i>hola</i>"`

## Objeto String

- Otros métodos de `String`:
  - `charAt(índice)` Devuelve el carácter situado en la posición especificada por `índice`.
  - `charCodeAt(índice)` Devuelve el número de la codificación del carácter situado en la posición especificada por `índice`.
  - `indexOf(cadena_buscada, índice)` Devuelve la posición, dentro de la cadena actual, de la primera ocurrencia de `cadena_buscada`, a partir de la posición indicada por `índice`. El argumento `índice` es opcional, si no se proporciona, la búsqueda comienza por el primer carácter de la cadena.
  - `lastIndexOf(cadena_buscada, índice)` Devuelve la posición, dentro de la cadena actual, de la última ocurrencia de `cadena_buscada` a partir de la posición indicada por `índice` y buscando hacia atrás. El argumento `índice` es opcional, si no se proporciona, la búsqueda comienza por el último carácter de la cadena.

## Objeto String

- Otros métodos de **String**:
  - **toLowerCase()** Devuelve la cadena de caracteres en minúsculas.
  - **toUpperCase()** Devuelve la cadena de caracteres en mayúsculas.
  - **substr(inicio,longitud)** Devuelve la subcadena que comienza en la posición indicada por *inicio* y acaba en la posición *inicio* + *longitud* - 1.
  - **substring(primerIndice,segundoIndice)** Devuelve la subcadena que comienza en la posición indicada por *primerIndice* y que finaliza en la posición anterior a la indicada por *segundoIndice*. Si *primerIndice* es menor que *segundoIndice* se intercambian los valores
  - **slice(primerIndice,segundoIndice)** Devuelve la subcadena que comienza en la posición indicada por *primerIndice* y que finaliza en la posición anterior a la indicada por *segundoIndice*. Si el *segundoIndice* es negativo se empieza a contar desde el final del String

## Objeto String

- Otros métodos de **String**:
  - **concat(cadena1, cadena2...)** concatena el string con *cadena1*, *cadena2*,...
  - **split(separador)** Parte la cadena en un array de cadenas de caracteres. Si el carácter o expresión regular *separador* no se encuentra, devuelve un array con un solo elemento que coincide con la cadena original.
  - **search(ExpReg)** devuelve la posición del string en donde concuerda la *ExpReg*. Si no hay concordancia devuelve -1
  - **match(ExpReg)** devuelve un array con las coincidencias entre encontradas en el patrón
  - **replace(ExpReg/substring,nuevaCadena)** reemplaza las apariciones de la *ExpReg* o *substring* por la *nuevaCadena*

## Objeto RegExp

- Permite la definición y manipulación de expresiones regulares al estilo de PERL
- Una expresión regular es una cadena de caracteres que representa un patrón que puede ser utilizado para realizar búsquedas y sustituciones de Strings
- Para crear una expresión regular se utiliza el constructor **RegExp**

```
expresion = new RegExp("\\d+");
expresion = new RegExp("[aeiou]+", "gi");
```
- Se puede crear una expresión regular literal

```
expresion = /[aeiou]+/gi;
```

## Objeto RegExp

- Métodos de **RegExp**
  - **exec(cadena)** Si existe concordancia entre el patrón y la cadena devuelve la primera concordancia. Si se invoca otra vez devolverá la segunda concordancia...
  - **test(cadena)** Devuelve true o false en función de si existe o no concordancia entre el patrón y la cadena
- Propiedades **RegExp**
  - **globalSpecifies**, **matchmultiline**, **ignoreCaseSpecifies** contendrán el valor **true** si se han activado los correspondientes flags en la expresión regular
  - **lastIndex** Es un índice que contiene la posición del String por donde continuará la búsqueda del patrón
  - **source** contiene el texto del patrón

## ECMA 6: Conjuntos: Set

- Nueva estructura “conjunto” de valores **únicos de cualquier tipo**. Si se pasa un elemento iterable como parámetro, sus elementos se añaden al conjunto

```
let conjunto = new Set();
```

- **size** propiedad que devuelve el número de elementos del conjunto

```
let elconjunto = new Set();
elconjunto.add(8);
elconjunto.add(4);
elconjunto.add(8);
console.log (elconjunto.size); // 2
```

## ECMA 6: Conjuntos: Set

- **Métodos:**
- **add(nuevoelemento)** - Añade el nuevo elemento al conjunto
- **delete (elemento)** - Elimina el elemento del conjunto, devolviendo true o false dependiendo de si estaba contenido o no.
- **clear()** - Elimina todos los elementos del conjunto
- **has(elemento)** - Devuelve true o false dependiendo de si el elemento pertenece o no al conjunto
- **entries()** - Devuelve un elemento iterable que contiene un conjunto de pares [clave, valor] en el que las claves son iguales que los valores en el orden de inserción
- **values()** - Devuelve un elemento iterable que contiene los valores de los elementos en el orden de inserción.
- **keys()** - Devuelve un elemento iterable que contiene los valores de los elementos en el orden de inserción (igual que values()).

## ECMA 6: Objeto Map

- Nueva estructura iterable de pares de **clave/valor**  

```
let mapa = new Map();
```
- **size** propiedad que devuelve el número de elementos del "mapa"
- Métodos:
  - **set(clave, valor)** - Establece el valor de una determinada clave
  - **get (clave)** - Recupera el valor de una determinada clave

```
let carrito = new Map();
carrito.set("Prod1", 5);
carrito.set("Prod2", 2);
carrito.set("Prod3", 2);
console.log (carrito.size);           // 3
console.log (carrito.get("Prod2"));   // 2
console.log (carrito.get("Prod1"));   // 5
```

## ECMA 6: Objeto Map

- Métodos:
  - **delete (clave)** - Elimina el elemento asociado a la clave y devuelve su valor
  - **clear()** - Elimina todos los pares clave/valor
  - **has(clave)** - Devuelve true o false dependiendo de si existe un valor asociado a la clave
  - **entries()** - Devuelve un elemento iterable que contiene el conjunto de pares [clave, valor]
  - **values()** - Devuelve un elemento iterable que contiene los valores de los elementos.
  - **keys()** - Devuelve un elemento iterable que contiene las claves de los elementos.

```
for (let [clave, valor] of carrito.entries()) {
  console.log ("%s tiene como valor: %s", clave, valor);
}
```

## Reescritura de contenidos

- Se puede **conocer y modificar** el contenido de un elemento con la propiedad **innerHTML**.
- La propiedad **innerHTML** incluye las etiquetas **HTML**.
- Es muy frecuente su utilización en **AJAX**.

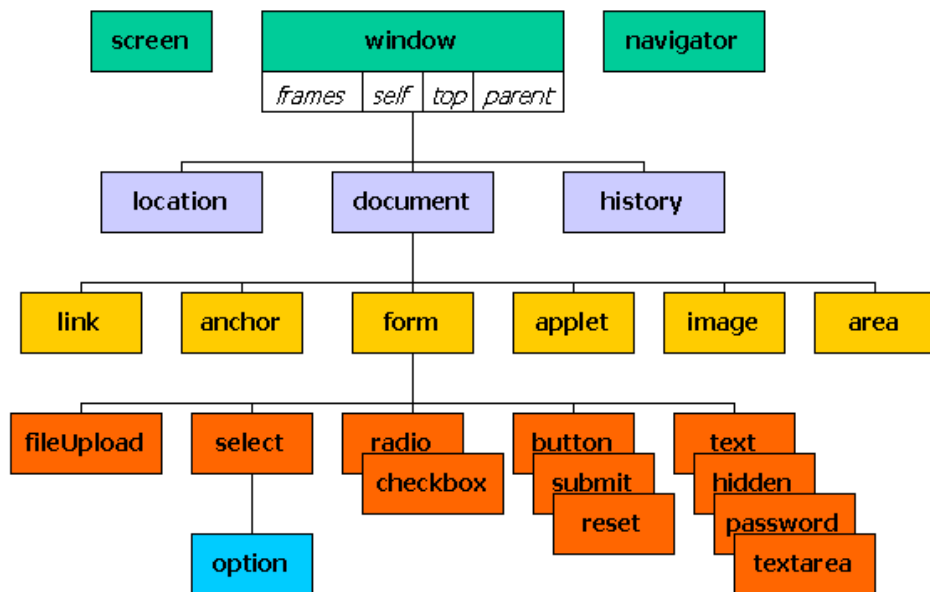
```
<div id= "micapa" style='border:solid 1px black;color:red'>
  Este es el contenido<br>
  <b> de la capa que estoy viendo </b>
</div>
```

```
alert (document.getElementById("micapa").innerHTML;

var cadena = "Este es el <i>nuevo contenido</i> de la capa";

document.getElementById("micapa").innerHTML =  cadena;
```

## BOM(Browser Object Model)



## BOM: objeto screen

- El objeto **screen** tiene las siguientes propiedades:
  - **width** contiene la anchura en pixels de la pantalla.
  - **height** contiene la altura en pixels de la pantalla.
  - **left** contiene la posición izquierda donde se puede situar la ventana.
  - **top** contiene la posición superior donde se puede situar la ventana.
  - **colorDepth** Contiene el número de bits de la paleta que se está utilizando en la pantalla.

## BOM: objeto navigator

- Propiedades del objeto **navigator**:
  - **appName**, nombre del navegador
  - **appCodeName**, nombre del código base del navegador
  - **appVersion**, versión del navegador
  - **userAgent**, información enviada por el navegador al servidor cuando se pide una página Web
  - **mimeTypes[i].type**, elemento i-ésimo de la matriz de tipos mime reconocidos por el navegador
  - **plugins[i].name**, elemento i-ésimo de la matriz de plugins reconocidos por el navegador
  - **userLanguage**, versión del lenguaje del navegador
  - **platform**, sistema operativo en el que se ejecuta
  - **online**, booleano indicando si el navegador está en línea
  - **cookiesEnabled**, booleano indicando si las cookies están habilitadas



## BOM: objeto window

- ◎ Propiedades del objeto **window**
  - **closed**, booleano indicando si la ventana está cerrada.
  - **defaultStatus**, mensaje por defecto en la barra de estado.
  - **frames**, array que representa los objetos frame del objeto window.
  - **history**, array que representa las URL almacenadas en el historial del objeto window.
  - **length**, número de frames
  - **name**, nombre de la ventana o del frame actual.
  - **opener**, referencia al objeto window que abrió el objeto window actual (si se utilizó para ello el método open()).
  - **parent**, referencia al objeto window que contiene frameset.
  - **self**, Nombre alternativo del objeto window actual.
  - **status**, mensaje en la barra de estado en un instante.
  - **top**, Nombre alternativo para la ventana de nivel más superior.
  - **window**, Nombre alternativo del objeto window actual.

## BOM: objeto window

- ◎ Métodos del objeto **window**
  - **alert(mensaje)**, muestra el mensaje en una ventana de diálogo.
  - **confirm(mensaje)**, muestra el mensaje en una ventana de diálogo con las opciones de "**Aceptar**" y "**Cancelar**".
  - **prompt(mensaje, respuesta)**, muestra el mensaje en una ventana de diálogo y se captura la respuesta del usuario.
  - **blur()**, elimina el foco del objeto **window**.
  - **focus()**, captura el foco sobre el objeto **window**.
  - **setInterval("expresión", tiempo)**, Evalúa la **expresión** cada vez que pasen los milisegundos especificados en **tiempo**. Devuelve un ID
  - **setTimeout("expresión", tiempo)**, Evalúa la expresión después de que hayan pasado los milisegundos especificados en tiempo. Devuelve un ID.
  - **clearInterval(ID)**, Elimina el timer referenciado por ID.
  - **clearTimeout(ID)**, Cancela el timer especificado por ID.

## BOM: objeto window

### ● Métodos del objeto `window`

- `moveBy(x,y)`, mueve el objeto `window` actual el número de pixels especificados por x e y.
- `moveTo(x,y)`, mueve el objeto `window` actual a las coordenadas especificadas por x e y.
- `resizeBy(x,y)`, ajusta el tamaño del objeto `window` actual moviendo su esquina inferior derecha, el número de pixels especificados por x e y.
- `resizeTo(ancho,alto)`, ajusta el tamaño del objeto `window` actual cambiando los valores `outerWidth` y `outerHeight` a los valores especificados en ancho y alto.
- `scroll(x,y)`, desplaza el objeto `window` actual a las coordenadas especificadas por x e y.
- `scrollBy(x,y)`, desplaza el objeto `window` actual el número de pixels especificados por x e y.
- `scrollTo(x,y)`, desplaza el objeto `window` actual a las coordenadas especificadas por x e y.
- `close()`, cierra el objeto `window` actual.
- `print()`, abre la ventana de dialogo para la impresión del documento

## BOM: objeto window

- `open(url, nombre, características)`, método que carga la `url` en la ventana llamada `nombre` con las `características` especificadas. Las `características` opcionales son:
  - `toolbar = [yes,no,1,0]`: la ventana tiene barra de herramientas.
  - `location = [yes,no,1,0]`: la ventana tiene campo de localización.
  - `directories = [yes,no,1,0]`: la ventana tiene botones de dirección.
  - `status = [yes,no,1,0]`: la ventana tiene barra de estado.
  - `menubar = [yes,no,1,0]`: la ventana tiene barra de menús.
  - `scrollbars = [yes,no,1,0]`: la ventana tiene barras de desplazamiento.
  - `resizable = [yes,no,1,0]`: la ventana podrá cambiar de tamaño.
  - `width = pixels`: Indica el ancho de la ventana cliente en pixels.
  - `height = pixels`: Indica el alto de la ventana cliente en pixels.
  - `outerWidth = pixels`: Indica el ancho total de la ventana en pixels.
  - `outerHeight = pixels`: Indica el alto total de la ventana en pixels.
  - `left = pixels`: distancia en pixels desde el lado izquierdo de la pantalla a la que se deberá colocar la ventana.
  - `top = pixels`: distancia en pixels desde el lado superior de la pantalla a la que se deberá colocar la ventana.

## BOM: Objeto location

### • Propiedades del objeto `location`:

- `hash`, nombre del enlace interno (`#enlace`) dentro de la URL.
- `host`, nombre del servidor y el número de puerto de la URL.
- `hostname`, Es una cadena que contiene el nombre de dominio del servidor (o la dirección IP), dentro de la URL.
- `href`, Es una cadena que contiene la URL completa.
- `pathname`, El camino al recurso, dentro de la URL.
- `port`, el número de puerto del servidor, dentro de la URL.
- `protocol`, el protocolo utilizado dentro de la URL.
- `search`, query-string dentro de la URL.

## BOM: Objeto location

### • Métodos del objeto `location`:

- `reload()`, recarga la URL especificada en la propiedad `href` del objeto `location`.
- `replace(cadenaURL)`, reemplaza el historial actual mientras carga la URL especificada en `cadenaURL`.

- Cambiando el valor de `location` de una ventana o frame se carga otra página en ellos.

```

window.location="url"
window.nombreframe.location="url"

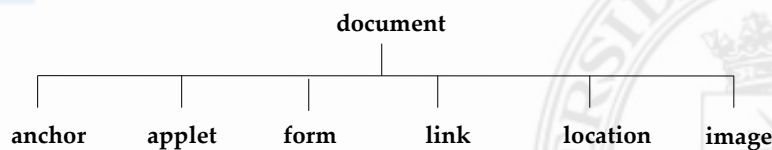
```

## BOM: Objeto history

- El objeto **history** almacena la lista de sitios por los que se ha estado "navegando".
- Se utiliza este objeto para manejar dicha lista, pudiendo realizar movimientos hacia delante y hacia atrás en la misma.
- Métodos objeto **history**:
  - **back()**, recarga la URL del documento anterior dentro del historial.
  - **forward()**, recarga la URL del documento siguiente dentro del historial.
  - **go(posición)**, recarga la URL del documento especificado por posición dentro del historial.

## BOM: objeto document

- El objeto **document** representa cualquier documento cargado en una ventana o frame.
- Jerarquía del objeto **document**



## BOM: objeto document

### ■ Propiedades del objeto `document` (I)

- `alinkColor`, Representa el color de los enlaces activos.
- `anchors`, array conteniendo enlaces internos del documento.
- `applets`, array conteniendo todos los applets del documento.
- `bgColor`, Representa el color de fondo del documento.
- `cookie`, cadena con los valores de las cookies del documento actual.
- `domain`, es el nombre del servidor que ha servido el documento.
- `embeds`, array conteniendo los elementos `<EMBED>` del documento.
- `fgColor`, Representa el color de primer plano del documento.
- `forms`, array conteniendo todos los objetos form del documento.

## BOM: objeto document

### ■ Propiedades del objeto `document` (II)

- `Images`, array conteniendo todas las imágenes del documento.
- `lastModified`, fecha de la última modificación del documento.
- `linkColor`, representa el color de los enlaces del documento.
- `links`, array conteniendo los enlaces externos del documento.
- `location`: Es una cadena que contiene la URL del documento actual.
- `referrer`, cadena conteniendo la URL del documento que llamó al actual, si el usuario utilizó un enlace.
- `title`, cadena conteniendo el título del documento actual.
- `vlinkColor`, fepresenta el color de los enlaces visitados.

## BOM: objeto document

### ▪ Métodos de **document**

- **clear()**, limpia la ventana del documento.
- **close()**, cierra la escritura sobre el documento actual.
- **open(*mime*, "replace")**, abre la escritura sobre un documento. El parámetro *mime* especifica el tipo de documento soportado por el navegador. Si la cadena "replace" se pasa como segundo parámetro, se reutiliza el documento anterior dentro del historial.
- **write()**, escribe texto y HTML sobre el documento actual.
- **writeln()**, escribe texto y HTML sobre el documento actual, seguido de una nueva línea.
- **getElementById("identificador")** accede al elemento HTML que tenga el atributo id con el valor de "identificador"
- **getElementsByName("nombre")** Accede a todos los elementos HTML que tenga en el atributo name el valor "nombre"
- **getElementsByTagName("etiqueta")** Accede a todos los elementos HTML de tipo especificado por "etiqueta"

## BOM: array de objetos links

- **links** es un array de objetos que contiene las urls de los enlace contenidos en un documento.
- Las propiedades de cada item de **links** son las mismas que las que tiene el objeto **location**

## BOM: array de objetos anchors

- **anchors** es un array de objetos que contiene información sobre los enlaces internos del documento.
- Propiedades de los elementos de **anchor**
  - **name**: Contiene el valor de este atributo en el **anchor**.
  - **text**: Contiene el texto comprendido entre las etiquetas de apertura y cierre del **anchor**.

## BOM: array de objetos images

- **images** es un array de objetos que contiene información sobre una imagen de un documento.
- Propiedades de los items de **images**
  - **border**, contiene el valor del atributo **border**.
  - **complete**, booleano que indica si la imagen se descargó totalmente.
  - **height**, contiene el valor del atributo **HEIGHT**.
  - **hspace**, contiene el valor del atributo **HSPACE**.
  - **lowsrc**, contiene el valor del atributo **LOWSRC**.
  - **name**, Contiene el valor del atributo **NAME**.
  - **src**, contiene el valor del atributo **SRC**.
  - **vspace**, contiene el valor del atributo **VSPACE**.
  - **width**, contiene el valor del atributo **WIDTH**.

## BOM : objeto form

- El objeto `form` contiene toda la información sobre un formulario de un documento.
- Propiedades del objeto `form`
  - `action`, cadena que especifica la URL donde la información del formulario debe ser procesada/enviada.
  - `elements`: array que contiene cada uno de los objetos que aparecen en el formulario, y en el mismo orden en que aparecen en el código HTML.
  - `encoding`, cadena que contiene la codificación MIME especificada en el atributo `ENCTYPE`.
  - `method`, cadena que contiene el nombre del método encargado de recibir/procesar la información del formulario.
  - `target`, cadena que contiene el nombre de la ventana a la que se deben enviar las respuestas del formulario.
- Métodos del objeto `form`
  - `reset()`, resetea el formulario. Produce el mismo resultado que pulsar sobre un botón de tipo `RESET`.
  - `submit()`, envía el formulario. Produce el mismo resultado que pulsar sobre un botón de tipo `SUBMIT`.

## BOM: objetos text, password y textarea

- Propiedades de los objetos `text`, `password` y `textarea`
  - `name`, cadena que contiene el nombre del objeto `text`, `password` o `textarea`, es decir, contiene el valor del atributo `NAME`.
  - `defaultValue`, cadena que contiene el valor por defecto del objeto
  - `value`, cadena que contiene el valor del objeto `text`, `password` o `textarea`, en cada momento. Esta propiedad también la tienen los objetos `hidden` y `file`
  - `disabled`, booleano que indica si el elemento está deshabilitado
  - `readonly`, booleano que indica si el elemento es solo de lectura
  - `type`, tipo de control
  - `form`, referencia al formulario donde está incluido el objeto (también lo tienen `hidden` y `file`)
- Métodos de los objetos `text`, `password` y `textarea`
  - `blur()`, dispara el evento de eliminar el foco del objeto dado
  - `click()`, dispara el evento de eliminar el foco del objeto dado.
  - `focus()`, dispara el evento de obtener el foco sobre el objeto dado.
  - `select()`, dispara el evento de seleccionar el texto del objeto.
- `textarea` cuenta con propiedades para acceder a `cols` y `rows`
- `text` y `password` cuenta con propiedades para acceder a `size` y `maxlength`



## BOM: objetos submit, reset y button

### ◉ Propiedades de los objetos `submit`, `reset` y `button`

- `name`, cadena que contiene el nombre del objeto es decir, el valor del atributo `NAME`.
- `value`, cadena que contiene el valor del objeto botón, es decir, el valor del atributo `VALUE`.
- `type`, tipo de control
- `form`, referencia al formulario donde está incluido el objeto

### ◉ Métodos de los objetos `submit`, `reset` y `button`

- `click()`, dispara el evento de pulsar sobre el botón

## BOM: objeto checkbox

### ◉ Propiedades del objeto `checkbox`

- `checked`, valor booleano que indica si el objeto `checkbox` está seleccionado o no.
- `defaultChecked`, valor booleano que indica si el objeto `checkbox` debe estar seleccionado por defecto.
- `name`, cadena que contiene el nombre del objeto `checkbox`, es decir, contiene el valor del atributo `NAME`.
- `value`, cadena que contiene el valor del objeto `checkbox`, es decir, contiene el valor del atributo `VALUE`.
- `form`, referencia al formulario donde está incluido el objeto

### ◉ Métodos del objeto `checkbox`

- `click()`, dispara el evento de pulsar sobre el objeto

## BOM: objeto radio

- ⊙ Agrupa a los elementos radio con igual valor del atributo **name**.
- ⊙ Realmente es un array en el que cada ítem se corresponde con uno de los elementos radio agrupado
- ⊙ Propiedades del objeto **radio**
  - **length**, valor numérico que nos indica el número de opciones.
  - **form**, referencia al formulario donde está incluido el objeto
  - **nombreradio[i]** contiene:
    - **checked**, valor booleano que indica si el objeto radio está seleccionado o no.
    - **defaultChecked**, booleano que indica si debe estar seleccionado por defecto.
    - **name**, cadena que contiene el nombre del objeto radio (atributo **NAME**).
    - **value**, cadena que contiene el valor del objeto radio (atributo **VALUE**).
- ⊙ Métodos del objeto **radio**
  - **click()**, dispara el evento de pulsar sobre una acción se debe aplicar a un ítem del radio (**nombreradio[i].click()**).

## BOM: objeto select

- Propiedades del objeto **select**
  - **length**, es un valor numérico que nos indica el número de opciones del objeto select dado.
  - **form**, referencia al formulario donde está incluido el objeto
  - **multiple**, boolean que indica si el select es múltiple
  - **size**, contiene que contiene el valor del atributo SIZE
  - **name**, es una cadena que contiene el nombre del objeto select, es decir, contiene el valor del atributo NAME.
  - **selectedIndex**, Es un valor numérico que nos indica el índice de la opción actualmente seleccionada.
  - **options**, es un array con las opciones del select

## BOM: array options del objeto select

- ◉ **options**: array que contiene cada una de las opciones que aparecen en la lista seleccionable. Tiene las siguientes propiedades:
  - **defaultSelected**, indica si la opción fue seleccionada por defecto.
  - **index**, valor numérico que indica el índice de la opción dentro de la lista.
  - **length**, valor numérico que indica el número de opciones de la selección.
  - **options**, cadena que contiene el código HTML de la lista de selección.
  - **selected**, valor booleano que indica si la opción está seleccionada o no.
  - **selectedIndex**, valor numérico: indica el índice a la opción seleccionada.
  - **text**, cadena que contiene el texto mostrado en la lista para cada opción.
  - **value**, contiene el valor (atributo **VALUE**) de una opción en particular

## Objeto Image

- Permite almacenar imágenes.
- Se utiliza para realizar precargas de imágenes.
- Por medio del constructor **Image(A,B)** se crea un objeto imagen del tamaño **A x B** sin fichero de imagen asociado. El tamaño es optativo
- Por medio de la propiedad **src** se asigna un fichero de imagen al objeto

```
miimagen = new Image(100,100);
miimagen.src= "imagen.gif";
```

FENW
miw.etsisi.upm.es • 103

## Eventos

- Un evento es un suceso que ocurre en un navegador cuando el usuario hace algo o interno del propio navegador (carga o descarga de página Web)
- En el modelo original de JavaScript los eventos pueden ser tratados por manejadores de eventos asignando una serie de sentencias al identificador del evento en una etiqueta cualquiera
 

```
<TAG atributos onevento = "Tratamiento del evento">
```
- Los eventos pueden ser disparados por código javascript
 

```
form.submit()
```

FENW
miw.etsisi.upm.es • 104

## Lista de Eventos

- Lista de manejadores de eventos que pueden ser definidos
 

<ul style="list-style-type: none"> <li>- onabort</li> <li>- onblur</li> <li>- onchange</li> <li>- onclick</li> <li>- ondblclick</li> <li>- onerror</li> <li>- onfocus</li> <li>- onkeydown</li> <li>- onkeypress</li> <li>- onkeyup</li> <li>- onload</li> </ul>	<ul style="list-style-type: none"> <li>- onmousedown</li> <li>- onmousemove</li> <li>- onmouseup</li> <li>- onmouseover</li> <li>- onmouseout</li> <li>- onreset</li> <li>- onresize</li> <li>- onselect</li> <li>- onsubmit</li> <li>- onunload</li> </ul>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Eventos: valor de retorno y this

- El retorno producido por el código JavaScript tiene significado importante para algunos manejadores:
  - Si se desea evitar que un form sea enviado:
 

```
<Form name="miform" onsubmit="return Comprueba()">
```

"Comprueba" debe devolver **true** si todo ha ido bien y **false** si no se debe enviar el formulario
  - En general, si al manejador del evento se le devuelve el valor "false", se anulará el evento por el que se ha disparado dicho manejador
- En el código que procesa un evento la referencia *this* es al elemento HTML que captura el evento

## Ejemplo de cancelación de eventos: return false

```
<html><head><title>Prueba de eventos</title>
</head>
<body>
<script type='text/javascript'>
  function Procesa(ElEvento){
    var evento = ElEvento || window.event;
    var tecla = evento.charCode || evento.keyCode;
    if (tecla <48 || tecla > 57){
      alert ('Te dije solo números');
      return false;
    } // del if tecla esta fuera de rango
    return true;
  } // de la función
</script>
<form name='datos'>
  Campo de números
  <input type='text' name='campo' onkeypress='return Procesa(event) '>
</form>
</body>
</html>
```

## Manejadores de eventos como propiedades

- Los manejadores de eventos se pueden asignar en javascript como si fuesen propiedades:

```
document.getElementById("elem").onclick = laFuncion;
```

**Atención:** el nombre de la función a ejecutar no incorpora los paréntesis ya que si los llevase se asignaría el resultado de su ejecución al evento

Para ejecutar algo en la carga se puede usar:

```
window.onload = laFuncion;
```

Y equivale a: `<body onload = "laFuncion()">`

// o una función anónima:

```
window.onload = function(){
    // código a ejecutar
}
```

## Manejadores de eventos en DOM2

- Registro de manejadores de eventos: Se utiliza el método `addEventListener()` del objeto sobre el que se desea tratar el evento  
`addEventListener(tipoevento, funcion, boolean)`

- `tipoevento` es el tipo de evento sin "on"
- `funcion` es el "manejador" del evento
- `true` o `false` determinan la fase en que se trata (captura o propagación)

Ej: `document.addEventListener("click",comprueba, true)`

- Eliminación de manejadores de eventos : Se quita un manejador con el método `removeEventListener()` del objeto sobre el que se desea tratar el evento

```
document.removeEventListener("click",comprueba,
true)
```

## OBJETO EVENT: Tratamiento avanzado

- Cada vez que ocurre un evento se crea un objeto denominado **event** entre cuyas propiedades se encuentran las principales características del evento en curso. Se destruye al finalizar la ejecución de las funciones que lo tratan.
- En Internet Explorer el objeto existe como **window.event**
- En el resto de los navegadores el objeto aparece de manera automática como parámetro de la función que lo trata:

```
function TrataEvento (objetoevento){
    .....
}
```

## OBJETO EVENT: Tratamiento avanzado

- Tratamiento por todos los navegadores:

```
function TrataEvento (ElEvento){
    var evento = ElEvento || window.event;
    //Aquí se trataría el evento
}
document.onclick = TrataEvento;
```

- Dentro de la función que lo trata se puede acceder a todas las propiedades del objeto

## OBJETO event: Propiedades comunes

- **type**: tipo de evento ocurrido (al igual que el nombre del evento pero sin "on", por ejemplo **mousedown**, **click**, etc)
- **target**: es una referencia al elemento sobre el que ha ocurrido el evento que se ha disparado
- **button**: determina el botón pulsado en el ratón (1-izdo, 2-dcho, 3-izdo y dcho, 4- central, etc)
- **clientX**, **clientY**: coordenadas en las que ocurre el evento respecto de la ventana del navegador. No tiene en cuenta el scroll efectuado
- **screenX**, **screenY**: coordenadas en las que ocurre el evento respecto de la pantalla completa
- **altKey**: valor booleano que determina si en el momento de ocurrir el evento estaba pulsada la tecla **Alt**
- **ctrlKey**: valor booleano que determina si en el momento de ocurrir el evento estaba pulsada la tecla **Control**
- **shiftKey**: valor booleano que determina si en el momento de ocurrir el evento estaba pulsada la tecla **Shift**

## OBJETO event: Tecla Pulsada

- En IE:
  - **keyCode**:
    - keypress – devuelve el código Unicode del carácter de la tecla pulsada
    - keydown, keyup – devuelve el código numérico de la tecla pulsada
- En DOM2:
  - **keyCode**: devuelve el código numérico de la tecla pulsada. Es cero para keypress
  - **charCode**: devuelve el código Unicode del carácter de la tecla pulsada. No definido para keyup y keydown.



## OBJETO event: Tecla Pulsada

- Para el tratamiento con keypress hay que igualar el tratamiento en todos los navegadores:

```
function daTeclapulsada (ElEvento){
    var evento = ElEvento || window.event;
    var codigo = evento.charCode || evento.keyCode;
    caracter = String.fromCharCode(codigo);
    alert (caracter)
}

document.onkeypress = daTeclaPulsada;
```

## Objetos definidos por el programador

- Un objeto es una abstracción del mundo real caracterizada por las propiedades lo que identifican y el comportamiento que tiene.
- Un objeto queda determinado por sus propiedades y métodos. Las **propiedades** son las características del objeto y los **métodos** son las funciones definidas para manipular las propiedades del objeto.
- Todos los objetos derivan su funcionalidad de objeto estándar **Object**. El mecanismo para adquirir funcionalidad de otros objetos no es la herencia clásica de los lenguajes de programación sino que se realiza por medio del **prototipado**

## Creación de un objeto: literal

- Se utiliza para crear un único ejemplar de objeto. Se definen la propiedades dentro de un unas llaves de apertura y cierre como una lista, separadas por comas, de pares formados por el nombre de la propiedad y su valor.

```
var unaPersona = {  
    nombre:"Pepe",  
    edad:"33",  
    sexo:"hombre"  
}
```

Los identificadores de las propiedades pueden ir entre comillas

## Creación de un objeto: dinámica

- Se utiliza para crear un único ejemplar de objeto.
- Se crea un objeto vacío y se añaden propiedades y métodos según se necesita

```
var unaPersona = {};  
unaPersona.nombre = "Pepe";  
unaPersona.edad = "33";  
unaPersona.sexo = "hombre";
```

## Creación de un objeto: Constructor (i)

- Se definen las propiedades del objeto, con notación ".", por medio de un constructor. **this** representa al ejemplar del objeto que se va a crear

```
function Personal(nombre, edad, sexo){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
}
```

- Se crea un ejemplar por medio del operador **new**

```
var unaPersona = new Personal("Lucía", 27, "mujer");
var otraPersona = new Personal("Olga", "", "mujer");
```

## Creación de un objeto: Constructor (ii)

- Se definen las propiedades del objeto ,con notación de array asociativo por medio de un constructor

```
function Persona2(nombre, edad, sexo){
    this["nombre"] = nombre;
    this["edad"] = edad;
    this["sexo"] = sexo;
}
```

- Se crea un ejemplar por medio del operador **new**

```
var unaPersona = new Personal("Lucía", 27, "mujer");
var otraPersona = new Personal("Olga", "", "mujer");
```

## Propiedades públicas y privadas

- El acceso a las propiedades públicas se hace con notación "." o notación de array asociativo:

```
Pepe.edad = 34;
Pepe["nombre"] = "José";
Lucia["edad"] = 29;
```

- Para definir una variable privada se declara como **var** en el constructor:

```
function Personal(nombre, edad, sexo){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
    var categoria = "amigo";
}
```

## Definición de métodos públicos

- Los métodos se definen asignando a un elemento del objeto la descripción de una función en cualquiera de los formatos posibles, teniendo en cuenta que para referirse a las propiedades públicas es necesario utilizar el prefijo **this**.

```
function getNombre(){return this.nombre;}
function Personal(nombre, edad, sexo){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
    var categoria = "amigo";
    this.getNombre = getNombre;
    this.getEdad = function(){return this.edad}
    this.getSexo = function(){return this.sexo}
    this.getCategoria= function(){return categoria}
}
```

## Utilización de métodos del objeto

- Los métodos públicos se invocan con la notación "." o de array asociativo

```
var nombre = Lucia.getNombre();
var edad = Lucia["getEdad"]();
```

- Los métodos privados se definen por medio de funciones internas del constructor que serán invocadas en su interior

```
function Personal(nombre, edad, sexo){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
    function incrementa(edad)
        {return edad + 1}
    this.edad = incrementa(this.edad);
}
```

## Métodos y propiedades estáticas

- Se pueden tener métodos y propiedades estáticas definiéndolos para el constructor de objetos

```
// Métodos y propiedades estáticos
Personal.especie="Homo Sapiens";
```

```
Personal.escribirEspecie=
    function() {
        alert('Las personas son de la especie' +
            Personal.especie)
    }
```

```
Personal.escribirEspecie();
```

## Extensión de objetos

- Se puedan asignar nuevas propiedades y métodos a todos los objetos creados con el mismo constructor definiéndolos en el objeto **prototype** del constructor.

```
var unapersona = new Personal("Lucía","30","mujer");
var otraspers = new Personal("Concepcion","70","mujer");
Personal.prototype.telefono="91";
Personal.prototype.setTelefono =
    function(telefono){this.telefono=telefono};
unapersona.setTelefono("888888888888");
alert(unapersona.telefono+" "+ otraspers.telefono)
```

## Uso de prototype

- Si se utiliza el patrón de creación de objetos mediante constructor se pueden crear los métodos en el prototipo del objeto para que no se repita en cada una de las instancias.
- El intérprete acude al prototipo cuando no encuentra un método o propiedad en un objeto

```
function Personal(nombre, edad, sexo){
    this.nombre = nombre;
    this.edad = edad;
    this.sexo = sexo;
};
Personal.prototype.anioNacimiento = function(){
    var hoy = new Date(),
        anio;
    anio = hoy.getFullYear();
    return anio - this.edad;
};

var unapersona = new Personal("Santiago", 35, "H");
console.log("El año de nacimiento es: " +
    unapersona.anioNacimiento());
```

FENW

## ECMA6: Clases

miw.etsisi.upm.es • 125

- ECMAScript 6 permite la definición de clases con una sintaxis intuitiva
- Se definen mediante la palabra reservada `class` y el cuerpo de la clase se encierra entre llaves `{}`
- Dentro del cuerpo se definen los métodos y constructores
- El constructor se define mediante la palabra clave `constructor` y solo puede haber uno

```
class Producto {
  constructor (nombre, precio, stock) {
    this.nombre = nombre;
    this.precio = precio;
    this.stock = stock;
  }
  setVenta(cantidad){
    this.stock = this.stock - cantidad;
    return this.precio * cantidad;
  }
}

const unProducto = new Producto ('memoria', 100, 25),
      vendidos = 2;
console.info ("Precio venta: "+ unProducto.setVenta(vendidos));
console.info ("Quedan "+ unProducto.stock);
```

FENW

## ECMA6: Clases (ii)

miw.etsisi.upm.es • 126

- Se pueden definir métodos estáticos mediante la palabra reservada `static`

```
class Producto {
  constructor (nombre, precio, stock) {
    this.nombre = nombre;
    this.precio = precio;
    this.stock = stock;
  }
  setVenta(cantidad){
    this.stock = this.stock - cantidad;
    return this.precio * cantidad;
  }
  static getIVA(producto, cantidad){
    const iva = 0.21;
    const ivaUnitario = producto.precio * iva;
    return ivaUnitario * cantidad;
  }
}

const unProducto = new Producto ('memoria', 100, 25),
      vendidos = 2;
console.info ("Precio venta: "+ unProducto.setVenta(vendidos));
console.info ("IVA: " + Producto.getIVA(unProducto, vendidos));
console.info ("Quedan "+ unProducto.stock);
```

FENW

ECMA6: Clases (iii) miw.etsisi.upm.es • 127

- Se definen subclases que heredan mediante la palabra reservada *extends* y dentro de una subclase se puede hacer referencia a la superclase a través de la palabra reservada *super*

```
class Producto {
  constructor (nombre, precio, stock) {
    this.nombre = nombre;
    this.precio = precio;
    this.stock = stock;
  }
  getIva(){
    const IVA = 0.21;
    var eliva = this.precio * IVA;
    return eliva;
  }
  setStock(cantidad){
    this.stock = this.stock - cantidad;
  }
  setVenta(cantidad){
    var cantTotal = 0;
    this.setStock(cantidad);
    cantTotal = (this.precio +
      this.getIva()) * cantidad;
    return cantTotal;
  }
}

class Libro extends Producto{
  constructor (nombre, precio, stock, titulo){
    super(nombre, precio, stock);
    this.titulo = titulo;
  }
  getIva(){
    const IVA = 0.04;
    var eliva = this.precio * IVA;
    return eliva;
  }
}
```

FENW

ECMA6: Clases (iv) miw.etsisi.upm.es • 128

```
class Empleado {
  constructor (nombre, sueldo){
    this.nombre = nombre;
    this.sueldo = sueldo;
  }
  getImpuestos(){
    return this.sueldo * 0.15;
  }
  static diferenciaSueldo (unEmp, otroEmp){
    var diferencia = Math.abs(unEmp.sueldo - otroEmp.sueldo);
    return diferencia;
  }
}

class Ingeniero extends Empleado {
  constructor(nombre, sueldo, categoria){
    super (nombre, sueldo);
    this.categoria = categoria;
  }
  getImpuestos(){
    return super.getImpuestos() + this.sueldo * 0.05;
  }
}
```



FENW
miw.etsisi.upm.es • 129

## ECMA6: Clases (v)

```

class Consultor extends Empleado {
  constructor(nombre, sueldo, categoria){
    super (nombre, sueldo);
    this.categoria = categoria;
  }
  getImpuestos(){
    return super.getImpuestos() + this.sueldo * 0.10;
  }
}

var unEmpleado = new Empleado ("Jorge", 100);
var unIngeniero = new Ingeniero ("Alma", 150, "técnico");
var unConsultor = new Consultor ("Esmeralda", 175, "gerente");

console.log(unEmpleado.nombre + " tiene " + unEmpleado.sueldo + " euros de sueldo");
console.log(unIngeniero.nombre + " tiene " + unIngeniero.sueldo + " euros de sueldo");
console.log(unConsultor.nombre + " tiene " + unConsultor.sueldo + " euros de sueldo");
console.log("La diferencia de sueldo entre " + unIngeniero.nombre + " y " + unConsultor.nombre + " es " + Empleado.diferenciaSueldo(unIngeniero, unConsultor) + " euros");
console.log(unEmpleado.nombre + " paga " + unEmpleado.getImpuestos() + " euros en impuestos");
console.log(unIngeniero.nombre + " paga " + unIngeniero.getImpuestos() + " euros en impuestos");
console.log(unConsultor.nombre + " paga " + unConsultor.getImpuestos() + " euros en impuestos");

```

FENW
miw.etsisi.upm.es • 130

## Bucles for .. in

- Un bucle for .. in accede a cada una de las propiedades de un objeto (o a cada uno de los elementos de un array)
 

```

for (propiedad in miObjeto){
  //procesar elementos miObjeto[propiedad]
}

```
- El bucle for in puede acceder a propiedades o métodos que no pertenecen al objeto sino a su prototipo
- Para evitarlo se debe usar el método hasOwnProperty, que será true si la propiedad es del objeto y no de su prototipo
 

```

for (propiedad in miObjeto){
  if (miObjeto.hasOwnProperty(propiedad))
    //procesar elementos miObjeto[propiedad]
}

```

## Manejo de excepciones

- Una excepción es un error en tiempo de ejecución. Se manejan por medio de la sentencia:

```
try{
    sentencias que se ejecutan normalmente. si en una
    se produce una excepción se salta a las sentencias
    de catch
}
catch (e){
    sentencias de tratamiento de la excepción
}
finally{
    sentencias que se ejecutan siempre después de las
    anteriores
}
```

- Se puede lanzar excepciones por medio de la sentencia **throw**

```
throw "excepción"
throw new Error("excepción");
```

## Ejemplo de manejo de excepciones

```
function fact(n){
    if (n < 0) throw "número negativo" //new Error("número
    negativo")
    else
        if (n == 0)
            return 1
        else return n*fact(n-1);
}

x = 3;
try{
    x = fact(x);
}
catch (e) {
    x = Number.NEGATIVE_INFINITY
    alert(e);
}
finally {alert(x);}
```

## DOM2: Introducción

- Cuando se termina de cargar un documento HTML, JavaScript crea un árbol con los elementos HTML
- En general, por cada etiqueta con texto se crea un nodo correspondiente a la etiqueta del que cuelga otro nodo con el texto
- En general, si una etiqueta **A** tiene en su contenido otra etiqueta **B** entonces se crea un nodo con la etiqueta **A** del que cuelgan 3 nodos: uno con el texto antes de **B**, otro con el texto de **B** y finalmente, un tercero con el texto después de **B**
- **Firefox** con la extensión **DOM Inspector** permite ver el árbol creado

## DOM2: acceso a los elementos HTML

- `document.getElementById("identificador")` devuelve una referencia al nodo correspondiente con el elemento HTML con el atributo `id` igual al valor `"identificador"`
- `document.getElementsByName("nombre")`, devuelve un array de referencias a los nodos correspondientes con elementos HTML con el atributo `name` igual al valor `"nombre"`
- `document.getElementsByTagName("etiqueta")`, devuelve un array de referencias a los nodos correspondiente con la `"etiqueta"` HTML
- `refnodo.getElementsByTagName("clase")`, devuelve un array de referencias a los nodos con elementos HTML con el atributo `class` igual al valor `"clase"` que sean descendiente de `refnodo`.
- `refnodo.querySelectorAll("selectorCSS")`, devuelve un array de referencias a los nodos que concuerden con el selector CSS proporcionado como parámetro que sean descendiente de `refnodo`.

## DOM2: Propiedades de los nodos

- **nodeName**, El nombre del nodo, such as HEAD for the head element.
- **nodeValue**, Si no es un nodo correspondiente con un elemento HTML (nodeType = 1), contiene el valor del nodo. Ej: si es un nodo de texto (nodeType = 3) contiene el texto asociado
- **nodeType**, El tipo numérico del nodo. Los mas importantes son: 1 = ELEMENT\_NODE, 2 = ATTRIBUTE\_NODE, 3 = TEXT\_NODE, 8 = COMMENT\_NODE, 9 = DOCUMENT\_NODE y 10 = DOCUMENT\_TYPE\_NODE
- **parentNode**, Referencia al nodo padre del nodo actual.
- **childNodes**, Lista de nodos hijos del nodo principal
- **firstChild**, Referencia al primer nodo hijo del actual.
- **lastChild**, Referencia al último nodo hijo del actual.
- **previousSibling**, Referencia al hermano anterior en una lista de nodos.
- **nextSibling**, Referencia al hermano posterior en una lista de nodos.
- **attributes**, Una lista de pares clave/ valor con los atributos de los nodos correspondientes con elementos HTML

## DOM2: creación, inserción y eliminación de nodos

- **document.createElement(etiqueta)** crea un nodo de tipo elemento HTML correspondiente con la **etiqueta** y devuelve una referencia a él
- **document.createTextNode(contenido)** crea un nodo de tipo texto con el contenido especificado y devuelve una referencia a él
- **insertBefore(nuevoNodo, referencia)**, inserta **nuevoNodo** antes del nodo especificado por **referencia** en el contexto aplicado
- **replaceChild(nuevoNodo, viejoNodo)**, sustituye **viejoNodo** por **nuevoNodo** en el contexto aplicado
- **removeChild(viejoNodo)**, elimina **viejoNodo** en el contexto aplicado
- **appendChild(nuevoNodo)**, añade un nuevo descendiente por el final

## DOM2: atributos de los nodos

- `getAttribute(nombre)` recupera el valor del atributo `nombre`
- `setAttribute(nombre,valor)` asigna el valor al atributo `nombre`
- `removeAttribute(nombre)` elimina el atributo `nombre`
- `hasAttribute(nombre)` devuelve un booleano que indica si el elemento tiene el atributo `nombre`

## DOM2: nuevos métodos

- `elemento.outerHTML(cadena)` reemplaza el `elemento` en el árbol DOM con el resultado de "parsear" la `cadena`
- `elemento.insertAdjacentHTML(lugar, cadena)` permite insertar la cadena HTML en el `lugar` indicado con respecto al `elemento`. Los valores de `lugar` pueden ser:
  - `beforeBegin` : inserción delante del `elemento`
  - `afterBegin` : inserción como primer hijo del `elemento`
  - `beforeEnd` : inserción como último hijo del `elemento`
  - `afterEnd` : inserción detrás del `elemento`

## DHTML: Dynamic HTML

- DHTML o HTML dinámico consiste en la manipulación de los atributos HTML y los valores de las propiedades CSS desde JavaScript
- Aporta dinamismo y vistosidad a las páginas
  - Necesita del acceso a través del DOM
  - Permite modificar estilos y atributos
  - Permite efectos dinámicos a través del posicionamiento y tamaño
  - Permite cambiar el contenido de los elementos
  - No depende de plug-ins

## Modificación del valor de una propiedad CSS

- Se puede modificar el valor de cualquier propiedad CSS de un elemento a través del objeto **style** del elemento, que contiene todas las propiedades CSS:
  - `document.getElementById('unaid').style.color = 'red'`
- Dado que se trata como un objeto, se puede acceder con la notación de array asociativo:
  - `document.getElementById('unaid').style['color'] = 'red'`
- ATENCIÓN: Si se desea manipular una propiedad CSS cuyo nombre esté formado por **más de una palabra** con guiones, **se unen** poniendo la **siguiente palabra en mayúsculas**:
  - **color** se manipula como **color**
  - **font-size** se manipula como **fontSize**
  - **text-align** se manipula como **textAlign**
  - Excepción: propiedad **"float"** es **styleFloat** en IE y **cssFloat** en los demás

## DHTML: cambio de valor de una propiedad CSS

```
<html><head><title>DHTML</title>
<script type="text/javascript">

function CambiaProps(elemento, propiedad, valor){
    document.getElementById(elemento).style[propiedad] = valor;
}

</script> </head> <body>
<div id='capa' STYLE='text-align:left'> Texto ¿alineado a la izquierda? </div>
<form>
    <input type="button" value='Alinear a la derecha'
        onclick="CambiaPropiedad('capa','textAlign','right')">
    <input type="button" value='Cambiar color'
        onclick="CambiaPropiedad('capa','color','red')">
    <input type="button" value='Cambiar tamaño'
        onclick="CambiaPropiedad('capa','fontSize','24pt')">
</form> </body> </html>
```

## DHTML: Acceso a los valores de propiedades CSS

- Solamente se puede acceder a través del objeto **style** a aquellas propiedades establecidas en línea o dinámicamente a través de javascript

```
<div id='capa1' class='un_estilo' style='left:1px;color:orange;'>
    Información en la capa
</div>
...
alert(document.getElementById('capa1').style['left']);
```

- Para acceder a los valores definidos en una hoja de estilo:
  - **window.getComputedStyle(elemento, pseudoelm)** es un objeto que almacena el valor de cada propiedad CSS definida. El parámetro **pseudoelm** se especifica si se quiere acceder a las propiedades asociadas con este selector
  - **elemento.currentStyle[propiedad]** es el equivalente en IE
  - Cuidado con las unidades de medida ya que los valores pueden ser almacenados con otras unidades distintas a las declaradas (Ej: mozilla almacena en px los tamaños de fuentes)

## DHTML: Acceso a los valores de propiedades CSS

```
<style type='text/css'>
.un_estilo{
    position:absolute;
    top:80px;
    font-size:38px;
}
</style>

<div id='capal' class='un_estilo' style='left:150;color:orange;'>
    Información en la capa
</div>
<script language='javascript'>

function devuelveValores(elemento, propiedad){
    var valor;
    elem = document.getElementById(elemento);
    if (elem.currentStyle)
        valor = elem.currentStyle[propiedad];
    else
        valor = window.getComputedStyle(elem,null)[propiedad];

    return valor;
}

...
// onload
elem = document.getElementById('capal');
alert(devuelveValores('capal','fontSize'));
```