



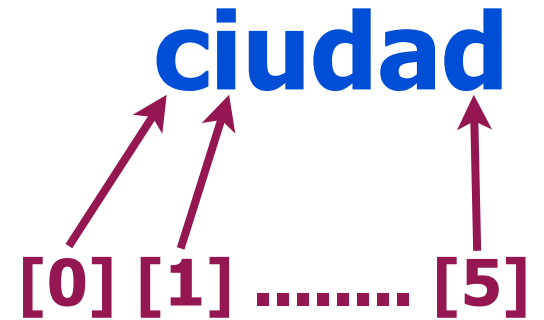
Strings de JavaScript

El tipo string



- ◆ String: texto internacionalizado codificado con el código UNICODE
 - delimitado entre **comillas** o **apóstrofes**
- ◆ Ejemplos
 - **"hola, que tal", 'hola, que tal', 'Γεια σου, ίσως' o '嘿, 也许'**
 - ◆ string "hola, que tal" en varios idiomas
 - String vacío: **""** o **"**
 - **"texto 'entrecomillado' "**
 - ◆ los delimitadores se pueden anidar: **'entrecomillado'** forma parte del texto
- ◆ Operador de concatenación de strings: **+**
 - **"Hola" + " " + "Pepe" => "Hola Pepe"**

String: un array de caracteres



◆ Un string es un **array de caracteres**

- cada caracter se referencia por un índice entre 0 y número_de_caracteres-1
- El string se procesa con métodos y propiedades de objetos

◆ Propiedad: **'ciudad'.length** ==> 6

- devuelve número de caracteres del string

◆ Acceso como array: **'ciudad'[2]** ==> 'u'

◆ Método: **'ciudad'.indexOf('da')** ==> 3

- devuelve posición de substring

◆ Método: **'ciudad'.substring(2,5)** ==> 'uda'

- devuelve substring entre ambos índices

Script ilustrativo

- ◆ El script genera un texto enmarcado con marcas HTML `<pre>`
 - Las marcas `<pre>` mantienen el formato del texto
- ◆ Cada línea se inserta con `document.writeln(..)` que añade `\n` al final
 - En el lado derecho de cada línea se describe la expresión
 - En el lado derecho se evalúa la expresión
 - ◆ El resultado de la evaluación se transforma a string antes de concatenarlo

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo string</title>
<meta charset="UTF-8">
</head>
<body>
<pre> <!-- marca <pre> mantiene el formato -->
<script type="text/javascript">
document.writeln(" 'Madrid!'.indexOf('id') => " + 'Madrid!'.indexOf('id'));
document.writeln(" 'Madrid!'.substring(2,5) => " + 'Madrid!'.substring(2,5));
document.writeln(" 'Madrid!'.length => " + 'Madrid!'.length);
</script>
</pre>
</body>
</html>
```

```
'Madrid!'.indexOf('id') => 4
'Madrid!'.substring(2,5) => dri
'Madrid!'.length => 7
```

```
" 'Madrid!'.indexOf('id') => " + 'Madrid!'.indexOf('id');
" 'Madrid!'.substring(2,5) => " + 'Madrid!'.substring(2,5);
" 'Madrid!'.length => " + 'Madrid!'.length;
```

Internacionalización: UNICODE



Teclado arabe

- ◆ JavaScript está internacionalizado
 - Puede representar textos de muchas lenguas diferentes
- ◆ Los strings JavaScript utilizan el **Basic Multilingual Plane** de **UNICODE**
 - Codificados en UTF-16
- ◆ Limitación: **teclados y editores** de los ordenadores de cada país
 - Los teclados y editores españoles soportan solo las lenguas oficiales



Teclados
chinos





Caracteres escapados

氷	𠩺	𠩺	𠩺	𠩺
2EA2	2EB2	2EC2	2ED2	2EE2

◆ Los **caracteres escapados**

- son caracteres no representables dentro de un string
 - ◆ comienzan por la barra inclinada (\) y estan representados en la tabla

◆ Ademas podemos representar cualquier caracter UNICODE o ISO-LATIN-1:

- **\uXXXX** caracter UNICODE de código hexadecimal **XXXX** 
- **\xXX** caracter ISO-LATIN-1 de código hexadecimal **XX** 

◆ Algunos ejemplos

- "Comillas dentro de \"comillas\""
 - ◆ " debe ir escapado dentro del string
- "Dos \n lineas"
 - ◆ retorno de línea delimita sentencias
- "Dos \u000A lineas"

CARACTERES ESCAPADOS

NUL (nulo):	\0, \x00, \u0000
Backspace:	\b, \x08, \u0008
Horizontal tab:	\t, \x09, \u0009
Newline:	\n, \x0A, \u000A
Vertical tab:	\t, \x0B, \u000B
Form feed:	\f, \x0C, \u000C
Carriage return:	\r, \x0D, \u000D
Comillas (dobles):	\", \x22, \u0022
Apóstrofe :	\', \x27, \u0027
Backslash:	\\, \x5C, \u005C



Boolean, igualdad y otros operadores lógicos de JavaScript

Tipo boolean

FALSE
true

- ◆ El tipo **boolean** solo tiene 2 valores
 - **true**: verdadero
 - **false**: falso
- ◆ Operador negación (negation): **!**
 - Convierte al valor lógico opuesto
- ◆ Conversión a boolean
 - **false**: 0, -0, NaN, null, undefined, "", "
 - **true**: resto de valores

!false	=> true
!true	=> false
!4	=> false
!"4"	=> false
!null	=> true
!0	=> true
!!""	=> false
!!4	=> true



Igualdad e identidad

- ◆ Igualdad estricta (identidad)
 - igualdad de tipo y valor: **===**
 - ◆ **funciona bien con tipos básicos!**
 - Objetos: identidad de referencias
 - negación de igualdad estricta: **!==**
- ◆ Igualdad y desigualdad débil: **==** y **!=**
 - Realiza conversiones impredecibles
 - ◆ **¡NO UTILIZAR!**

// Identidad de tipos básicos

```
0 === 0           => true
0 === 0.0         => true
0 === 1           => false
0 === false       => false

"" === ""         => true
```

// Identidad de referencias
// a objetos

```
var x = {}, y = x, z = {};
x === y           => true
x === {}          => false
x === z           => false
```

Operadores: &&, || y ?:

◆ Operador lógico Y (AND): **a && b**

- si **a** evalúa a **false**
 - ◆ devuelve **a**, sino devuelve **b**

◆ Operador lógico O (OR): **a || b**

- si **a** evalúa a **true**
 - ◆ devuelve **a**, sino devuelve **b**

◆ Operador condicional: **(c) ? a : b;**

- si **c** evalúa a **true**
 - ◆ devuelve **a**, sino devuelve **b**

```
0 && true      => 0
1 && "5"       => "5"
```

```
// Evita error de ejecución
// si x es undefined o null
y = x && x.a;
```

```
undefined || 0    => 0
13 || 0           => 13
```

```
// Valor por defecto
// si x es undefined o null
y = x || 0;
```

```
(7)? 0 : 1        => 0
( "")? 0 : 1      => 1
```

Operadores de comparación

◆ JavaScript tiene 4 operadores de comparación

- Menor: **<**
- Menor o igual: **<=**
- Mayor: **>**
- Mayor o igual: **>=**

◆ Números, booleanos, strings y objetos

- tienen definida una relación de orden
 - ◆ Aunque se utilizan principalmente con números

1.2 < 1.3 => true

1 < 1 => false

1 <= 1 => true

1 > 1 => false

1 >= 1 => true

false < true => true

"a" < "b" => true

"a" < "a" => false

"a" < "aa" => true



La sentencia if/else de JavaScript

Sentencia if/else

- ◆ Ejecución condicional de
 - bloques de instrucciones
- ◆ Condición debe ir entre paréntesis
- ◆ Bloques de 1 sentencia pueden omitir {}
- ◆ La parte **else** es opcional

```
17-if_bloque.js  UNREGISTERED
// Sentencia if/else
//
// -> ejecuta bloque 1
//     si x es true
//
// -> ejecuta bloque 2
//     si x es false

if (x) {
    y = 0;
    z = "hola";
}
else {
    y = 1;
    z = "adios";
}
```

```
18-if_bloque_1_sentencia.js  UNREGISTERED
// La parte else es opcional

if (x) {
    y = 0;
}

// Bloque de 1 sentencia
// puede omitir parentesis

if (x) y = 0;

if (x)
    y = 0;
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if/else </h3>
```

```
<script type="text/javascript">
```

```
    // Math.random() devuelve
    // número aleatorio entre 0 y 1.
```

```
    var numero = Math.random();
```

```
    if (numero <= 0.5){
        document.writeln(numero + ' MENOR que 0,5');
    }
    else {
        document.writeln(numero + ' MAYOR que 0,5');
    }
```

```
</script>
```

```
</body>
```

```
</html>
```

Sentencia if/else

0.5242976508023318 MAYOR que 0,5

Ejemplo con sentencia if/else

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if </h3>

<script type="text/javascript">

    // Math.random() devuelve
    // número aleatorio entre 0 y 1.
    var numero = Math.random();
    var str = ' MAYOR que 0,5';

    if (numero <= 0.5){
        str = ' MENOR que 0,5';
    }

    document.writeln(numero + str);
</script>
</body>
</html>
```

Sentencia if/else

0.5242976508023318 MAYOR que 0,5

Ejemplo con sentencia if

Funciones alert(), confirm() y prompt()

◆ Interacción sencilla basada en “pop-ups”

◆ alert(msj):

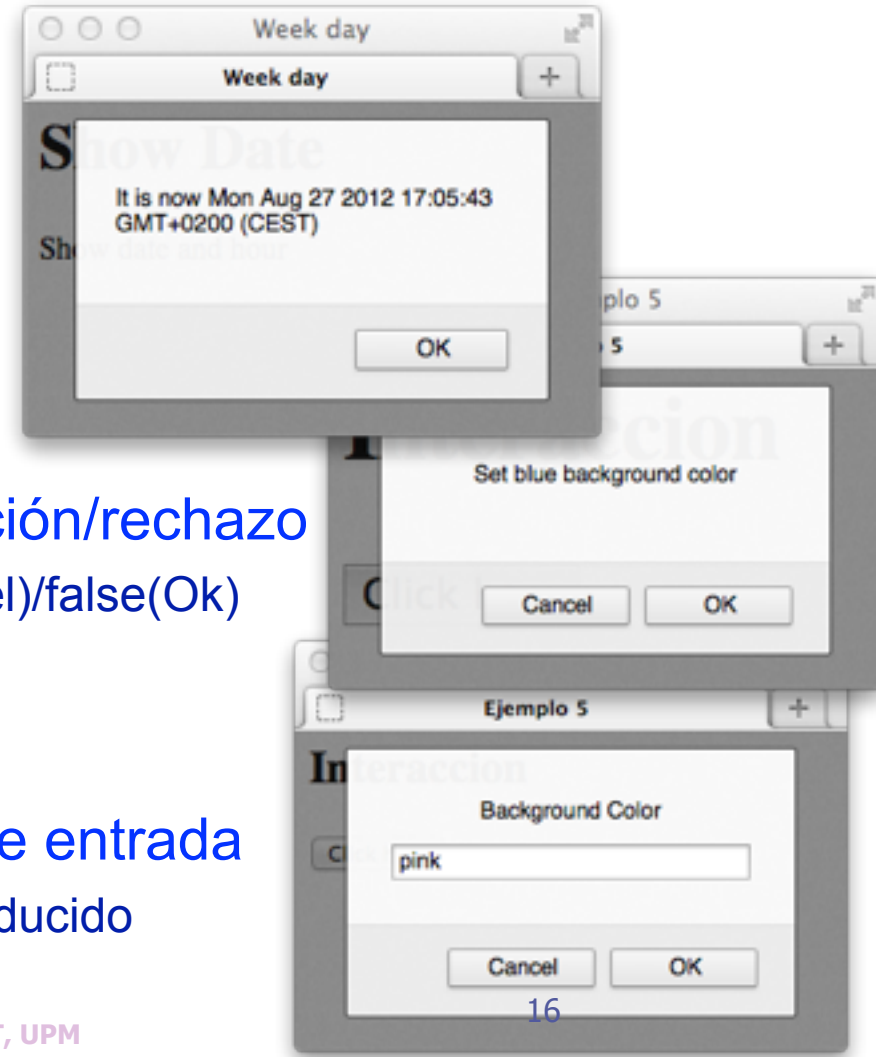
- presenta un mensaje al usuario
 - ◆ Retorna al pulsar OK

◆ confirm(msj):

- Presenta mensaje y pide confirmación/rechazo
 - ◆ Retorna al pulsar, devuelve true(Cancel)/false(Ok)

◆ prompt(msj):

- Presenta mensaje y pide un dato de entrada
 - ◆ Retorna al pulsar, devuelve string introducido



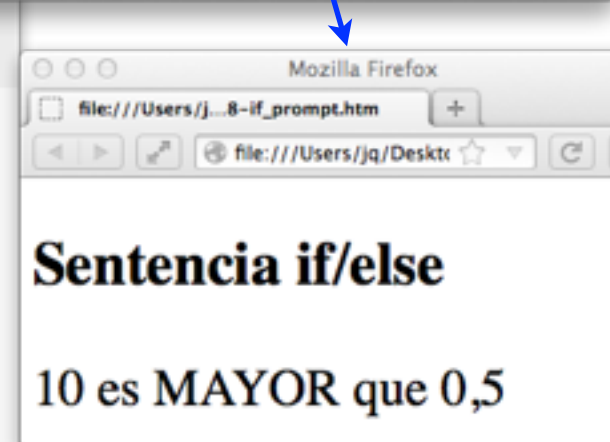
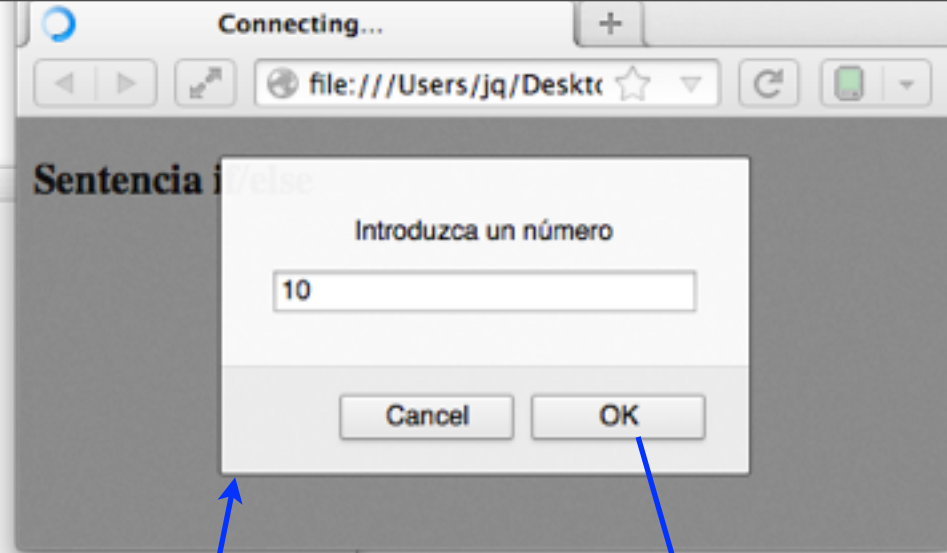
Ejemplo de prompt()

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3> Sentencia if/else </h3>

<script type="text/javascript">

    // Prompt pide un dato con un desplegable
    var numero = prompt("Introduzca un número");

    if (numero <= 0.5){
        document.writeln(numero + ' es MENOR que 0,5');
    }
    else {
        document.writeln(numero + ' es MAYOR que 0,5');
    }
</script>
</body>
</html>
```



Ejemplo de else-if

```
<!DOCTYPE html>
<html><head>
<meta charset="UTF-8">
</head><body>
<h3> Sentencia if/else </h3>

<script type="text/javascript">

    // Prompt pide un dato con un desplegable
    var numero = prompt("Introduzca un número");

    // isNaN(..) determina si es un número
    if (isNaN(numero)) {
        document.write(numero + ' no es un número');
    }
    else if (numero <= 0.5) {
        document.write(numero + ' es MENOR que 0,5');
    }
    else
        document.write(numero + ' es MAYOR que 0,5');
    }
</script>
</body>
</html>
```

Sentencia if/else

Introduzca un número

xx

Cancel

OK

Sentencia if/else

xx no es un número



Características avanzadas de los objetos de JavaScript

Propiedades dinámicas

◆ Las propiedades de objetos

- Pueden **crearse**
- Pueden **destruirse**

◆ Operaciones sobre **propiedades**

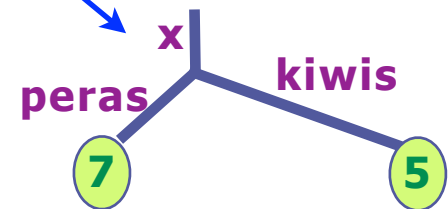
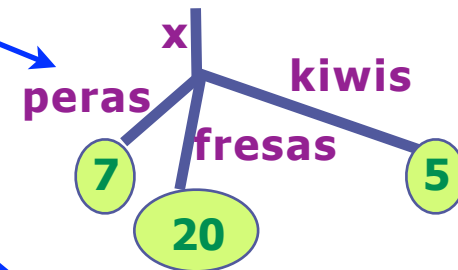
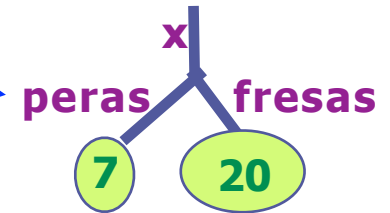
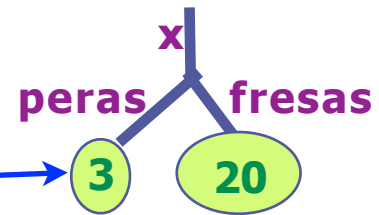
- **x.c = 4** ¡¡OJO: sentencia compleja!!
 - ◆ si propiedad **x.c** existe, le asigna **4**;
si **x.c** no existe, crea **x.c** y le asigna **4**
- **delete x.c**
 - ◆ si existe **y.c**, la elimina; si no existe, no hace nada
- **"c" in x**
 - ◆ si **x.c** existe, devuelve **true**, sino devuelve, **false**

```
var x = { peras:3, fresas:20};
```

```
x.peras = 7;
```

```
x.kiwis = 5;
```

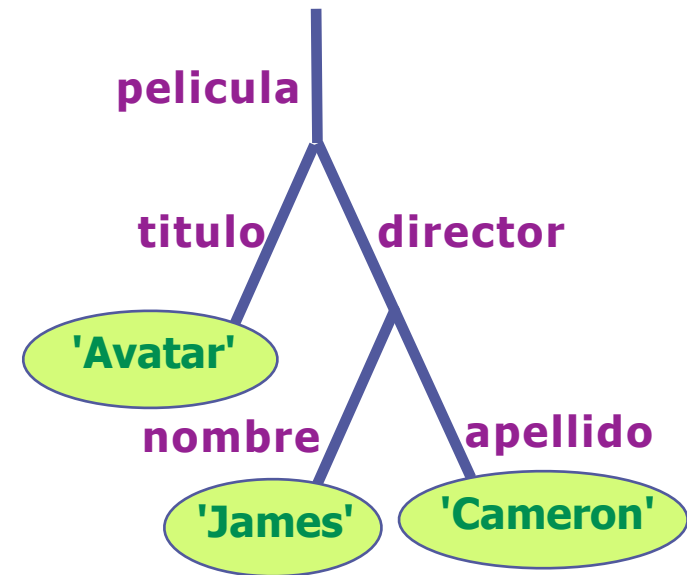
```
delete x.fresas;
```



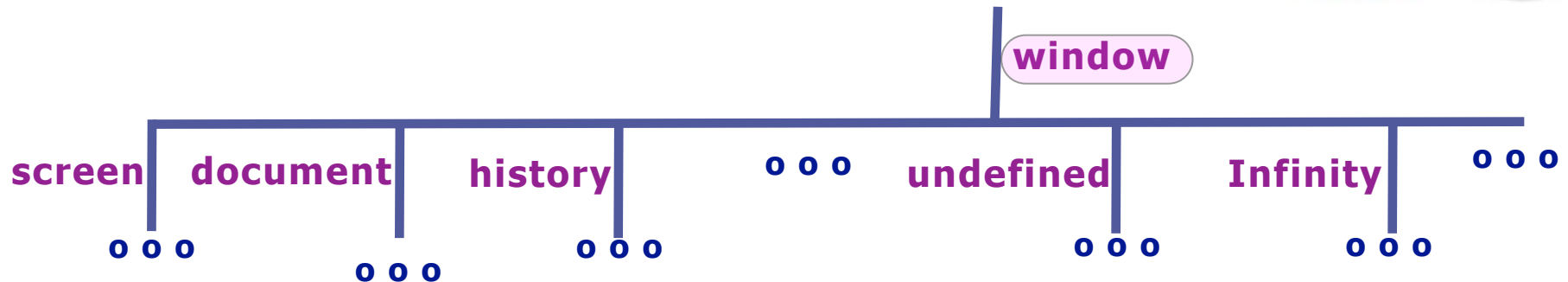
Objetos anidados: árboles

```
var pelicula = {  
  titulo: 'Avatar',  
  director: {  
    nombre: 'James',  
    apellido: 'Cameron'  
  }  
};
```

- ◆ Los objetos pueden **anidarse** entre si
 - Los objetos anidados representan **árboles**
- ◆ La notación punto o array puede **encadenarse**
 - Representando un **camino en el árbol**
 - ♦ Las siguientes expresiones se evalúan así:
 - `pelicula.director.nombre` \Rightarrow `'James'`
 - `pelicula['director']['nombre']` \Rightarrow `'James'`
 - `pelicula['director'].apellido` \Rightarrow `'Cameron'`
 - `pelicula.apellido` \Rightarrow `undefined`
 - `pelicula.apellido.director` \Rightarrow `Error_de_ejecución`

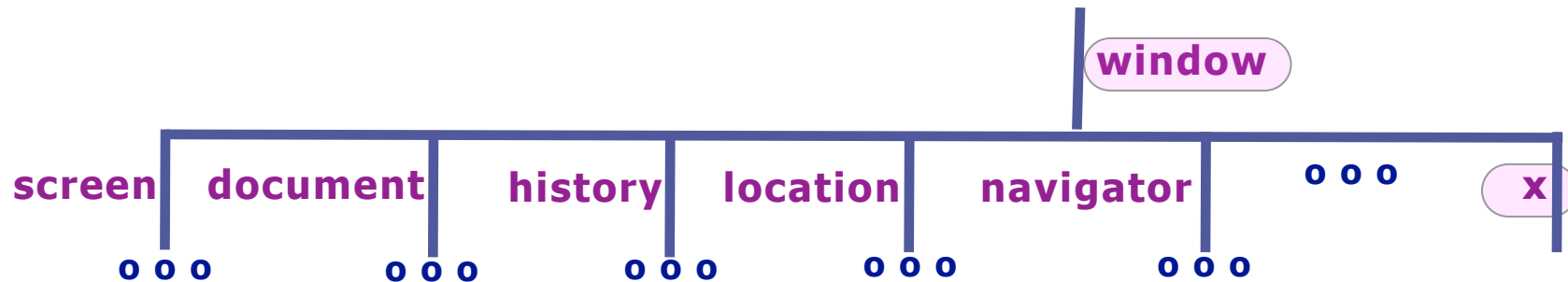


Objeto window: this



- ◆ JavaScript se ejecuta dentro del **objeto global window**
 - El **objeto global window** tiene propiedades con información sobre
 - ◆ Objetos predefinidos de JavaScript, el navegador, el documento HTML,
- ◆ JavaScript permite referenciar **window** como **this** u omitirse
 - La **propiedad document** de **window** se puede referenciar como
 - ◆ **window.document**, **this.document** o **document**
- ◆ Documentación: http://www.w3schools.com/jsref/obj_window.asp

Variables globales y el entorno de ejecución

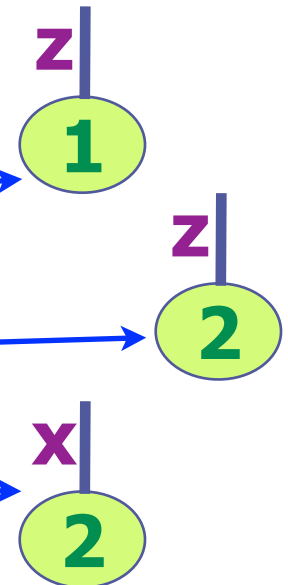


- ◆ Un programa JavaScript se ejecuta con el objeto **window** como entorno
 - las propiedades de window son visibles en todo el programa
- ◆ Las **variables globales** de JavaScript son **propiedades de window**
 - **x = 1;** es equivalente a **this.x = 1;** o a **window.x = 1;**
 - ◆ **x = 1;** si x no existe, crea una nueva propiedad (variable global) de window

Definición de variables globales

- ◆ Una variable global es visible en cualquier parte del programa
 - Una variable global se define con una sentencia de asignación
 - ◆ Crea una propiedad en el objeto global window
 - **OJO!** muy peligroso porque **puede modificar otras propiedades de window**
- ◆ La definición de variables globales es confusa
 - Se considera una parte mal diseñada de JavaScript
 - ◆ Se recomienda usarlas solo si es absolutamente necesario

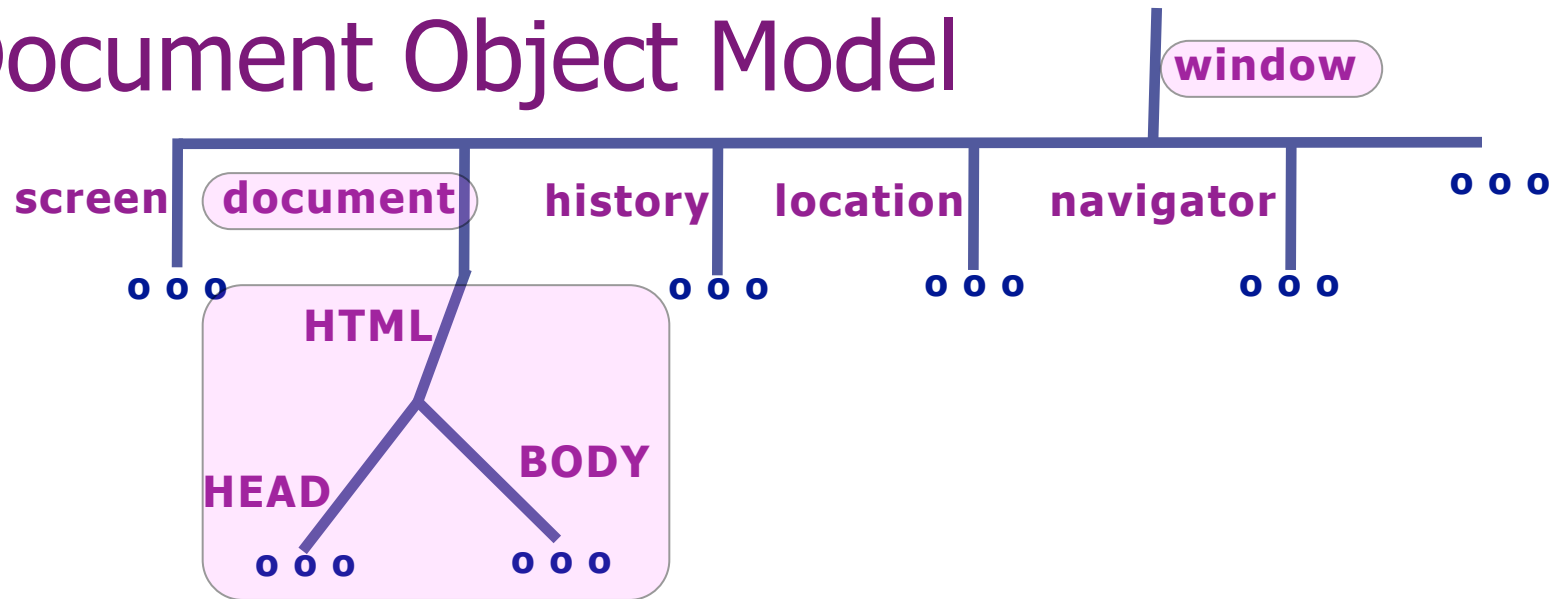
```
var z = 1;    // crea la variable local z inicializada a 1  
  
z = 2;       // asigna 2 a la variable local z  
             // porque la variable z ya existe  
  
x = 2;       // crea la variable global x inicializada a 2  
             // porque todavía no existe ninguna variable x.  
             // En realidad crea la propiedad x de window
```





Objetos DOM de JavaScript

DOM: Document Object Model



- ◆ **Objeto DOM:** objeto js asociado al documento HTML visualizado en el navegador
 - El navegador lo almacena en la propiedad **document** de **window**
 - ◆ Que contiene el árbol de objetos DOM de la página HTML
 - **Documentación:** http://www.w3schools.com/jsref/dom_obj_all.asp
- ◆ Los objetos DOM pueden buscarse por **atributos** (“id”, “class”, ..) de **HTML**
 - Por ej., el método **document.getElementById("idx")** devuelve el objeto DOM
 - ◆ asociado al elemento de la página HTML con **identificador** “idx”
- ◆ **window** tiene además **propiedades con el nombre de los identificadores**
 - **¡OJO! peligro:** Si el nombre no coincide con el de otra propiedad de **window**

- ◆ `getElementById ("fecha")` devuelve el objeto DOM de `<div id="fecha"></div>`
- ◆ Propiedad **innerHTML** de un objeto DOM
 - Permite extraer/insertar HTML en el elemento del documento

The image shows a web browser window titled "Ejemplo fecha y hora" displaying the text "La fecha y hora son:" followed by "Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)". Below the browser window, the source code of the HTML file is shown. The code includes a script that uses `document.getElementById("fecha")` to retrieve a DOM element and `innerHTML` to insert the current date and time. Dashed blue arrows indicate the flow of data: one arrow points from the `getElementById` call in the script to the `<div id="fecha"></div>` element in the HTML, and another arrow points from the `innerHTML` property access to the rendered date and time in the browser window.

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo fecha y hora</title>
<meta charset="UTF-8">
</head>

<body>
<h2>La fecha y hora son:</h2>

<div id="fecha"></div>

<script type="text/javascript">
  var cl = document.getElementById("fecha");
  cl.innerHTML = new Date( );
  // Insertar Date en elem con id="fecha"
</script>
</body>
</html>
```

© Juan Quemada, DIT, UPM

Acceso a
DOM

- ◆ La propiedad **fecha** de **window** contiene el objeto DOM de `<div id="fecha"></div>`
 - No está recomendado usar estas propiedades de window
 - ◆ Si hay colisión de nombres con otras propiedades y puede haber problemas
- ◆ Propiedad **innerHTML** de un objeto DOM
 - Permite extraer/insertar HTML en el elemento del documento

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo fecha y hora</title>
<meta charset="UTF-8">
</head>

<body>
<h2>La fecha y hora son:</h2>

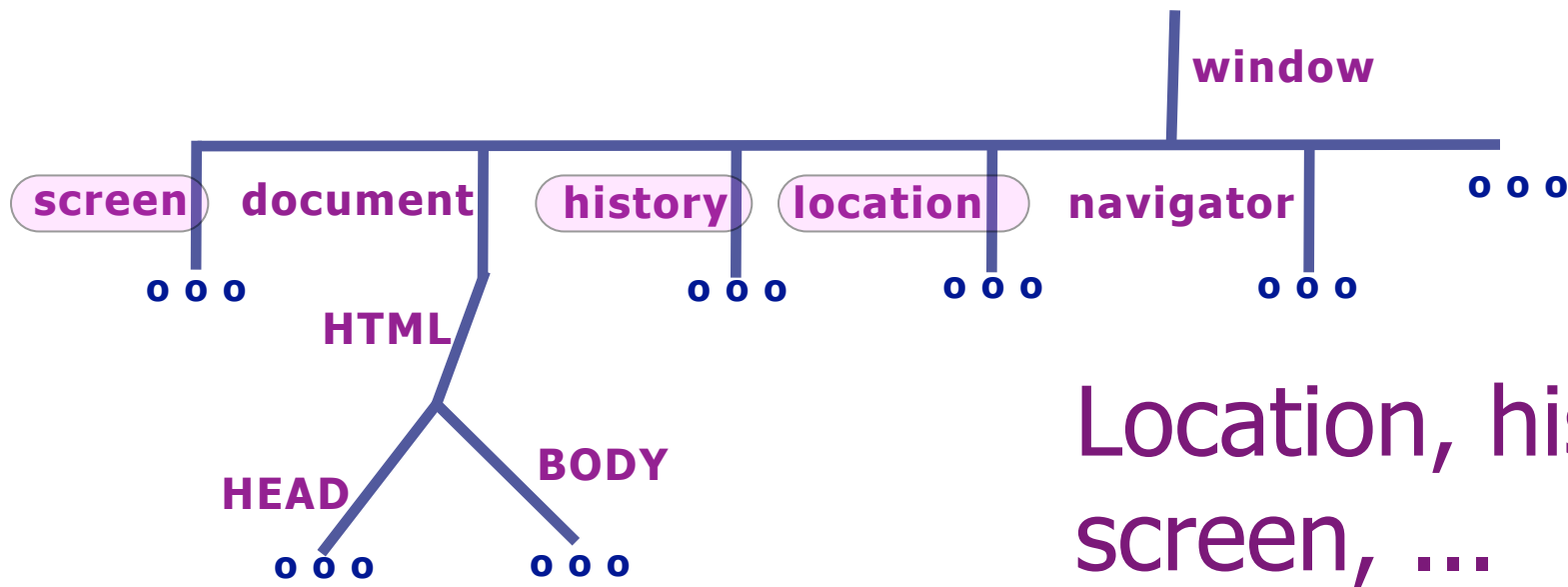
<div id="fecha"></div>

<script type="text/javascript">
  fecha.innerHTML = new Date( );
  // Insertar Date en elem con id="fecha"
</script>
</body>
</html>
```

La fecha y hora son:
Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)

Acceso a DOM

© Juan Quemada, DIT, UPM



Location, history,
screen, ...

- ◆ **location:** propiedad que contiene el URL a la página en curso
 - `location = "http://www.upm.es"` // Carga la página en el navegador
 - ◆ **location.reload()** re-carga la página en curso
 - propiedades: **href** (url), **protocol**, **hostname**, **port**, **pathname**, **search** (query), ...
- ◆ **history:** propiedad con la historia de navegación
 - Métodos para navegar por la historia: **history.back()**, **history.forward()**, ...
- ◆ **screen:** dimensiones de la pantalla
 - **width**, **height**, **availWidth**, **availHeight**: para adaptar apps a pantallas móviles
- ◆ Documentacion: <http://www.w3schools.com/jsref/>

window.screen

```
<!DOCTYPE html>
<html><head>
<title>DOM</title>
<meta charset="UTF-8">
<style>
  span {font-weight: bold;}
</style>
</head><body>
<h1>Propiedades de window</h1>
```

La propiedad location.href contiene el URL:

```
<br>
<span id="i1"></span>
<p>
```

Los pixels de mi pantalla (screen.width y screen.height) son:

```
<span id="i2"></span>
```

```
<script>
```

```
document.getElementById("i1").innerHTML = location.href;
```

```
var p = document.getElementById("i2")
p.innerHTML = screen.width + " x " + screen.height;
```

```
</script>
```

```
</body>
```

```
</html>
```

Propiedades de window

La propiedad location.href contiene el URL:

file:///Users/jq/Desktop/MOOC_FirefoxOS/s3/09-window_table.htm

Las dimensiones de mi pantalla (screen.width y screen.height) son: 2560 x 1440

Funciones de selección de elementos

◆ getElementById("my_id")

- Es el mas sencillo de utilizar porque devuelve un objeto o null
 - ◆ El objeto DOM con el identificador buscado
 - ¡Un identificador solo puede estar en un objeto de una página HTML!

◆ getElementByName("my_name"), getElementsByTagName("my_tag"), getElementsByClassName("my_class"), querySelectorAll("CSS selector"),...

- Devuelven una matriz de objetos
 - ◆ Por ejemplo: getElementByName("my_name")[0]
 - referencia el primer elemento con atributo **name="my_name"**

La fecha y hora son:

Thu Jan 30 2014 08:51:47 GMT+0100 (CET)

08:51:47.247 Use of Mutation Events is deprecated. Use MutationObserver instead. operator.js:1429

>> fecha.innerHTML = "hola"

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo fecha y hora</title>
<meta charset="UTF-8">
</head>

<body>
<h2>La fecha y hora son:</h2>

<div id="fecha"></div>

<script type="text/javascript">
  var cl = document.getElementById("fecha");
  cl.innerHTML = new Date( );
  // Insertar Date en elem con id="fecha"
</script>
</body>
</html>
```

Al ejecutar la instrucción en la consola Web de Firefox, cambiamos la fecha por "hola"

Ejemplo fecha y hora

Ejemplo fecha y hora

file:///Users/jq/Desktop

La fecha y hora son:

hola

08:52:41.433 MutationObserver instead.

08:52:41.435 fecha.innerHTML = "hola"

08:52:41.435 "hola"

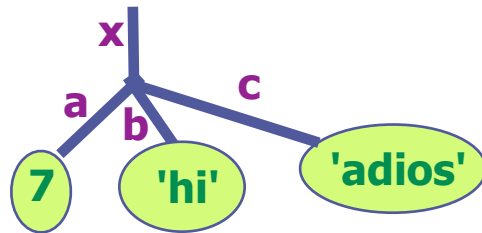
>>



Sentencia for/in de JavaScript

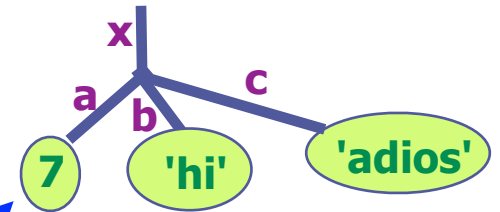
Sentencia for/in

- ◆ **for (i in x) {..bloque de instrucciones..}**
 - itera en todas las propiedades del objeto **x**
- ◆ El **nombre** de propiedad y su **contenido** se referencian con **"i"** y **"x[i]"**
 - **"i"** contiene el nombre de la propiedad en cada iteración
 - **"x[i]"** representa el valor de la propiedad **"i"**
 - ◆ Dentro de la sentencia for debe utilizarse la notación array



Sentencia for/in

- ◆ En el ejemplo se utiliza **for (i in x) {...}**
 - para mostrar en una página Web
 - ◆ el contenido de las propiedades de un objeto



```
<!DOCTYPE html><html>
<head><meta charset="UTF-8"></head>
<body>
<h3>Sentencia for/in:</h3>
```

```
<script type="text/javascript">
```

```
var x = {a:7, b:'hi', c:'adios'};
```

```
var i;
for (i in x) {
    document.write("Propiedad " + i + " = " + x[i] + "<br>");
}
```

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

Sentencia for/in:

Propiedad a = 7

Propiedad b = hi

Propiedad c = adios

Sintaxis de la sentencia for/in

- ◆ La sentencia comienza por **for**
- ◆ Sigue la **condición (i in obj)**
 - debe ir entre **paréntesis (...)**
- ◆ Los bloques de más de 1 sentencia
 - deben delimitarse con {...}
- ◆ Bloques de 1 sentencia
 - pueden omitir {...}, pero mejoran la legibilidad delimitándolos con {..}

```
14-for_in_bloque.js  UNREGISTERED

// Utilizar notacion array para
// acceder a propiedades: obj[i]

for (i in obj) {
    z = z + obj[i];
    obj[i] = "inspected";
}

// En bloques de solo 1 sentencia
// {...} es opcional
//     -> pero se recomienda usarlo

for (i in obj) {
    z = z + obj[i];
}

// Estas 2 formas son equivalentes
// pero menos legibles

for (i in obj)    z = z + obj[i];

for (i in obj)
    z = z + obj[i];
```

window.screen

```
<!DOCTYPE html>
<html>
<head>
<title>DOM</title>
<meta charset="UTF-8">
</head>
<body>
```

```
<h2> Screen </h2>
```

```
    <!-- tabla con propiedades de screen -->
<table id="tabla">
  <tr><th> Propiedad </th><th> Valor </th></tr>
</table>
```

```
<script>
var i, tabla = document.getElementById("tabla");
```

```
for (i in screen){ //cada iteración genera una fila de la tabla
  tabla.innerHTML+="|<td>" + i + "</td><td> = " + screen[i] + "</td></tr>";
}
|  |

```

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

Screen

Propiedad	Valor
availWidth	= 2514
availHeight	= 1418
availTop	= 22
availLeft	= 46
pixelDepth	= 24
colorDepth	= 24
width	= 2560
height	= 1440



Final del tema
Muchas gracias!

