



Tema 5.1

Javascript: Tipos, objetos y valores

JavaScript



◆ JavaScript

- Diseñado por Netscape en 1995 para ejecutar en un Navegador
 - ◆ Hoy se ha convertido en el **lenguaje del Web y Internet**

◆ Norma ECMA (European Computer Manufacturers Association)

- Versión soportada en navegadores actuales:
 - ◆ **ES5: ECMAScript v5**, Dic. 2009, (JavaScript 1.5)
- Navegadores antiguos soportan
 - ◆ ES3: ECMAScript v3, Dic. 1999, (JavaScript 1.3)



◆ Tutorial: <http://www.w3schools.com/js/>

◆ Referencia: <http://www.w3schools.com/jsref/>

◆ Libro: “*JavaScript Pocket Reference*”, D. Flanagan, O’Reilly 2012, 3rd Ed.

Tipos, objetos y valores

◆ Tipos de JavaScript

■ **number**

- ◆ números: **32**, **1000**, **3.8**



■ **boolean**

- ◆ los valores **true** y **false**

FALSE
true



■ **string**

- ◆ textos delimitados entre comillas o apóstrofes: **"hola que tal"**, **'hola que tal'**

■ **undefined**

- ◆ **undefined**: representa **indefinido**

UNDEFINED

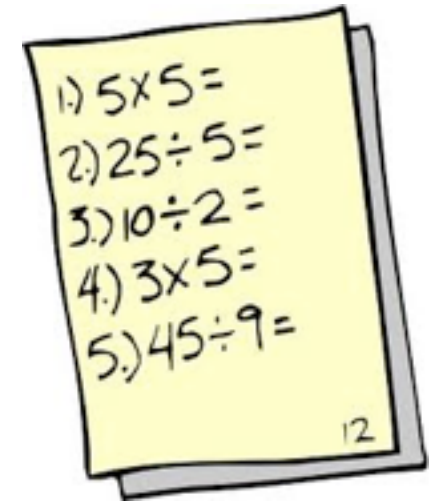
◆ Los **objetos** son agregaciones de valores de tipos

■ Se agrupan en **clases**: **Object**, **Array**, **Date**, ...

- ◆ Objeto **null**: valor especial que representa objeto nulo



Operadores y expresiones



- ◆ JavaScript incluye **operadores** de tipos y objetos
 - Los **operadores** permiten formar **expresiones**
 - ◆ Componiendo **valores** con los operadores
- ◆ Por ejemplo, con las operaciones aritmeticas +, -, *, /
 - podemos formar expresiones numéricas

13 + 7	=>	20	// Suma de números
13 - 7	=>	6	// Resta de números
(8*2 - 4)/3	=>	4	// Expresión compleja // Incluyendo paréntesis

Sobrecarga de operadores

- ◆ Algunos operadores tienen varias semánticas diferentes
- ◆ Por ejemplo, el operador **+** tiene 3 semánticas diferentes
 - **Suma de enteros** (operador binario)
 - **Signo de un número** (operador unitario)
 - **Concatenación de strings** (operador binario)



13 + 7 => 20 // Suma de números

+13 => 13 // Signo de un número

"Hola " + "Pepe" => "Hola Pepe" // Concatenación de strings



Conversión de tipos en expresiones

- ◆ JavaScript realiza conversión automática de tipos

- cuando hay ambigüedad en una expresión
 - ◆ utiliza las prioridades para resolver la ambigüedad

- ◆ La expresión **"13" + 7** es ambigua

- porque combina un **string** con un **number**
 - ◆ JavaScript asigna más prioridad al **operador +** de strings, convirtiendo **7** a string

- ◆ La expresión **+"13"** también necesita conversión automática de tipos

- El **operador +** solo está definido para **number**
 - ◆ JavaScript debe convertir el **string "13"** a **number** antes de aplicar operador **+**

13 + 7	=> 20
"13" + "7"	=> "137"
"13" + 7	=> "137"
+"13" + 7	=> 20



Clases, objetos y constructores

- ◆ JavaScript tiene **clases**, como **Object**, **Date**, **Array**, ...
 - Cada clase tiene un **constructor** de objetos con el mismo nombre
 - ◆ Constructores: **Object()**, **Date()**, **Array()**, ...

- ◆ Los objetos se construyen invocando el constructor con **new**, por ej.
 - **new Date();**
 - ◆ Crea objeto inicializado con fecha y hora en que ha sido creado
 - **new Object();**
 - ◆ Crea objeto vacío de la clase Object

Métodos

- ◆ Un **método** es una operación realizable sobre un objeto
 - Se invoca con la notación punto: **objeto.metodo(..params..)**
- ◆ Un objeto **hereda** métodos (de su **clase**)
 - que pueden invocarse sobre él (solo si los ha heredado)
- ◆ Por ejemplo
 - **document.write("Hola Mundo!")**
 - ◆ inserta string "Hola Mundo!" en página Web
 - **document.writeln("Hola Mundo!")**
 - ◆ inserta "Hola Mundo!" y **nueva linea** en la página Web



Operador typeof

- ◆ El operador **typeof** permite conocer el tipo de un valor
 - Devuelve un string con el nombre del tipo

typeof 7

=> "number"



typeof "hola"

=> "string"



typeof true

=> "boolean"

FALSE
true

typeof undefined

=> "undefined"

UNDEFINED

typeof null

=> "object"

typeof new Date()

=> "object"



• []	Acceso a propiedad o invocar método; índice a array
new	Crear objeto con constructor de clase
()	Invocación de función/método o agrupar expresión
++ --	Pre o post auto-incremento; pre o post auto-decremento
! ~	Negación lógica (NOT); complemento de bits
+ -	Operador unitario, números. signo positivo; signo negativo
delete	Borrar propiedad de un objeto
typeof void	Devolver tipo; valor indefinido
* / %	Números. Multiplicación; división; modulo (o resto)
+	Concatenación de string
+ -	Números. Suma; resta
<< >> >>>	Desplazamientos de bit
< <= > >=	Menor; menor o igual; mayor; mayor o igual
instanceof in	¿objeto pertenece a clase?; ¿propiedad pertenece a objeto?
== != === !==	Igualdad; desigualdad; identidad; no identidad
&	Operacion y (AND) de bits
^	Operacion ó exclusivo (XOR) de bits
 	Operacion ó (OR) de bits
&&	Operación lógica y (AND)
 	Operación lógica o (OR)
?:	Asignación condicional
=	Asignación de valor
OP=	Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= =
,	Evaluación múltiple

Operadores JavaScript

Los operadores están ordenados verticalmente por prioridades. Los más altos se evalúan antes.

Ejemplo de script: programa JavaScript



- ◆ Script: programa Javascript insertado en una página HTML
 - Se ejecuta al cargar la página HTML
 - ◆ Se delimita con la marca **<script>** con atributo mime “type=text/javascript”

The image shows a side-by-side comparison of an HTML file in an editor and its rendered output in a web browser. The editor window, titled '01-date.htm', contains the following code:

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

<h3>Mi primer script</h3>

<script type="text/javascript">
  document.write("3 + 2 => " + (3+2) + "<p>");
  document.write("Fecha: " + (new Date()));
</script>
</body>
</html>
```

The browser window, titled 'Mi primer script', displays the rendered page. It features a heading 'Mi primer script', a line showing the calculation '3 + 2 => 5', and a line showing the current date and time: 'Fecha: Tue Sep 10 2013 17:12:33 GMT+0200 (CEST)'. Purple arrows point from the corresponding code elements in the editor to the rendered output in the browser: from the <h3> tag to the heading, from the first document.write call to the calculation, and from the second document.write call to the date and time.

- ◆ **document.write(...)**
 - inserta string en la página
 - ◆ en donde está el script
- ◆ **new Date()** crea objeto Date
 - con fecha y hora actual



Aplicaciones de cliente

◆ Programa que se ejecuta en el navegador

- Se identificada por el URL de la página Web que los contiene
 - ◆ Por ejemplo: <http://upm.es/misapps/webapp.html>
- Se descarga con HTTP del servidor

1) Cliente solicita WebApp identificada con URL

2) Script se ejecuta al cargar la página Web en el navegador:

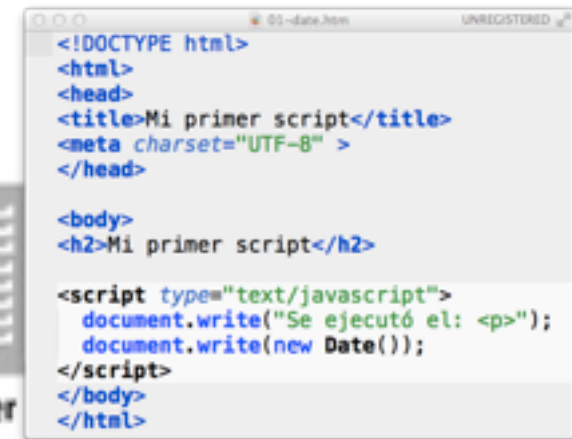


Solicitud HTTP asociada a un URL

Respuesta HTTP: página Web con script



Servidor sirve fichero identificado por URL



Ejercicio

◆ Indicar el resultado de evaluar las siguientes expresiones

- Tabla de conversión de tipos

- ◆ <https://www.inkling.com/read/javascript-definitive-guide-david-flanagan-6th/chapter-3/type-conversions>

10+23	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"
typeof (10+23)	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"
"10"+23	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"
typeof ("10"+23)	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"
+"10"+23	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"
typeof (new Array())	=> 10, 23, 33, 43, "1023", "1033", "number", "string", "object"

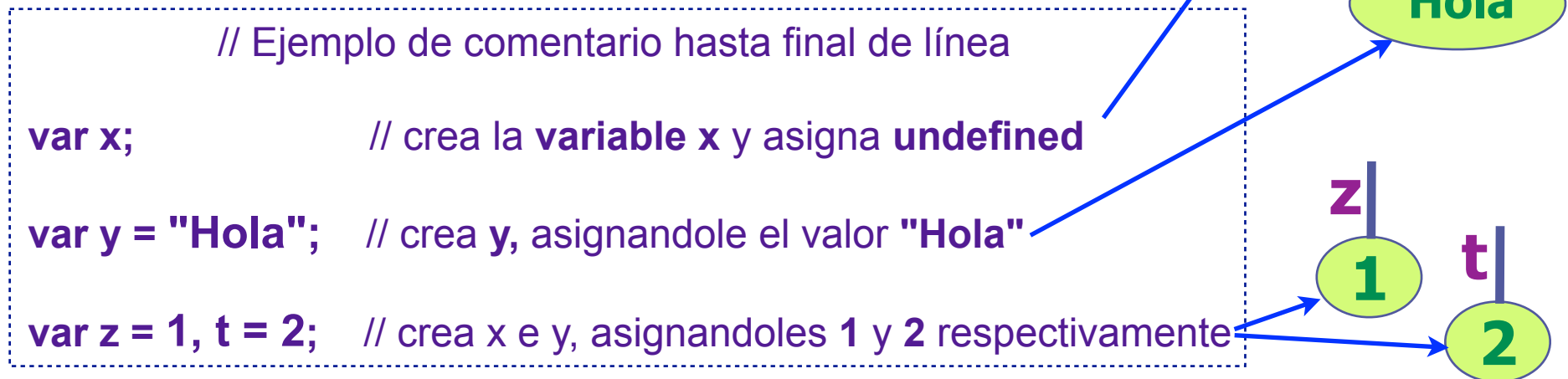


Tema 5.2

Javascript: Sentencias y variables

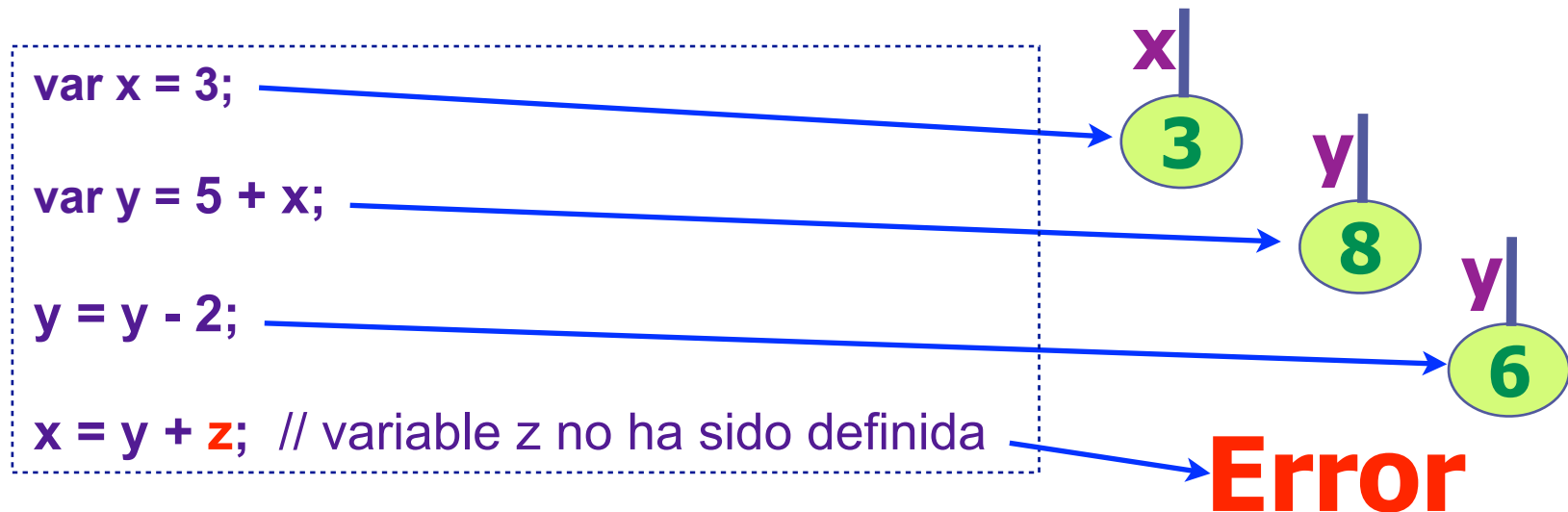
Definición de variables

- ◆ Las variables se definen con la palabra reservada **var**
 - Una variable puede inicializarse al definirse
- ◆ Una sentencia de definición de variable
 - puede definir varias variables simultaneamente



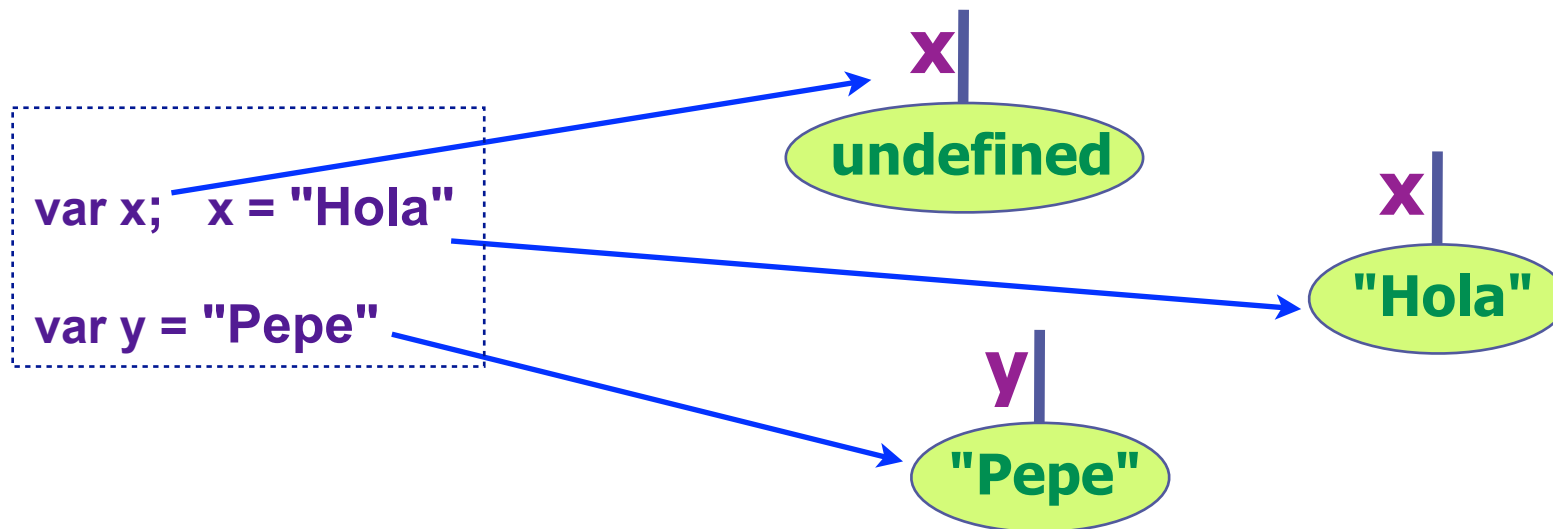
Expresiones con variables

- ◆ Las variables representan el valor que contienen
 - y pueden ser usadas en expresiones, siempre que hayan sido definidas
- ◆ Una sentencia puede contener solo una expresión a evaluar, p.e. $y = y - 2$;
- ◆ Usar una variable **no definida**, provoca **error** e **interrumpe el programa**



Programas y sentencias

- ◆ **Programa JavaScript:** secuencia de sentencias o instrucciones
 - Se ejecutan siguiendo el orden de declaración
- ◆ Una sentencia debe finalizarse con “;”
 - Puede finalizarse también con nueva línea
 - ◆ Pero **se recomienda terminar siempre con “;”** porque es mas legible y seguro



Las variables no tipadas

◆ Las variables de JavaScript

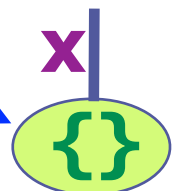
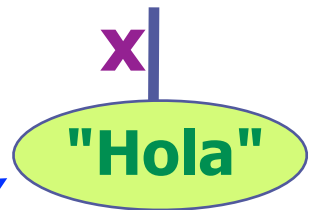
- pueden contener valores de cualquier tipo
 - ◆ Por ejemplo, número, texto, array, foto, color, ..

var x = 5; // Crea la variable x y le asigna el valor 5

x = "Hola"; // Asigna el string (texto) "hola" a la variable x

x = new Date(); // Asigna objeto de Date a variable x

x = new Object; // Asigna objeto de Object a variable x



Nombres de variables

- ◆ El **nombre** (o identificador) de una variable debe comenzar por:
 - **letra**, **_** o **\$**
 - ◆ El nombre pueden contener además **números**
 - Nombres **bien contruidos**: **x**, **ya_vás**, **\$A1**, **\$**, **_43dias**
 - Nombres **mal contruidos**: **1A**, **123**, **%3**, **v=7**, **a?b**,
- ◆ Un nombre de variable
 - **no** debe ser una **palabra reservada** de JavaScript
- ◆ Las variables son sensibles a **mayúsculas**
 - **mi_var** y **Mi_var** son variables distintas

STATEMENT SINTAXIS

block	{ statements };
break	break [label];
case	case expression:
continue	continue [label];
debugger	debugger:
default	default:
do/while	do statement while(expression);
empty	;
expression	expression;
for	for(init; test; incr) statement
for/in	for (var in object) statement
function	function name([param[,...]]) { body }
if/else	if (expr) statement1 [else statement2]
label	label: statement
return	return [expression];
switch	switch (expression) { statements }
throw	throw expression;
try	try {statements} [catch { statements }] [finally { statements }]
strict	"use strict";
var	var name [= expr] [,...];
while	while (expression) statement
with	with (object) statement

DESCRIPCIÓN DE LA SENTENCIA JAVASCRIPT

Agrupar un bloque de sentencias como 1 sentencia
Salir del bucle o switch o sentencia etiquetada
Etiquetar sentencia dentro de sentencia switch
Salto a sig. iteración de bucle actual/etiquetado
Punto de parada (breakpoint) del depurador
Etiquetar setencia default en sentencia switch
Alternativa al bucle while con condición al final

Sentencia vacía, no hace nada

Evaluar expresión (con efectos laterales)

Bucle sencillo. "init": inicialización;

"test": condición; "incr": acciones final bucle

Enumerar las propiedades del objeto "object"

Declarar una función llamada "name"

Ejecutar statement1 o statement2

Etiquetar sentencia con nombre "label"

Devolver un valor desde una función

Multiopción con etiquetas "case" o "default"

Lanzar una excepción

Gestionar excepciones

Activar restricciones strict a script o función

Declarar e inicializar una o mas variables

Bucle básico con condición al principio

Extender cadena de ámbito (no recomendado)



Script que no hace nada

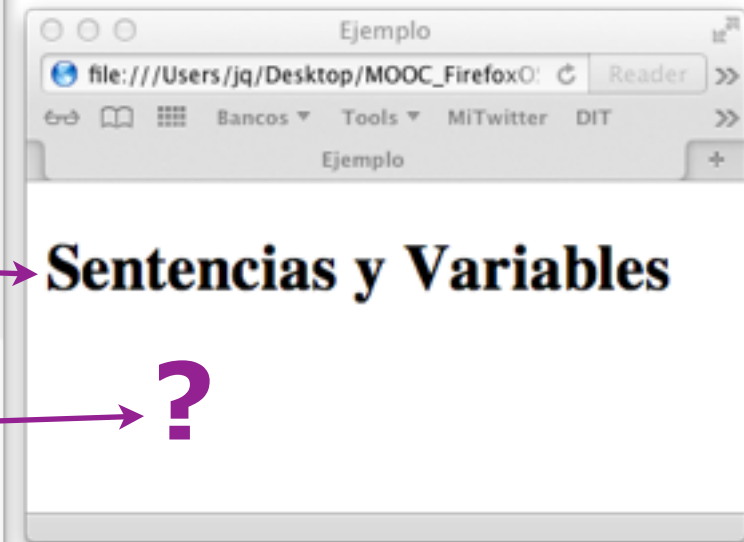
- ◆ El script del ejemplo se ejecuta al cargar la página
 - pero no muestra nada, solo define variables y les asigna valores

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

<body>
<h2>Sentencias y Variables</h2>

<script type="text/javascript">
    var x;
    x = "Hola";
    var y = "Pepe";
</script>
</body>
</html>
```

© Juan Quemada, DIT, UPM





Inserción de HTML en la Página Web

- ◆ La salida debe insertarse con **document.write(...)**
 - inserta un string en la página HTML donde está el script

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

<body>
<h2>Sentencias y Variables</h2>

<script type="text/javascript">
  var x;
  x = "Hola";
  var y = "Pepe";
  document.write("Ver variables x e y: " + x + " " + y);
</script>
</body>
</html>
```

Sentencias y Variables

Ver variables x e y: Hola Pepe

Ejercicio

- ◆ Indicar los 5 nombres de variables incorrectos
 - en las siguientes declaraciones de variable

```
var holaPepe = "hello";  
var /fichero.html = "hello";  
var _hola_pepe = "hello";  
var ¿viénes? = "hello";  
var $vás = "hello";  
var console.log = "hello";  
var %vás = "hello";  
var console-log = "hello";  
var niño_$ = "hello";  
var _$ = "hello";  
var console_log = "hello";
```



Tema 5.3

Javascript: Strings

El tipo string

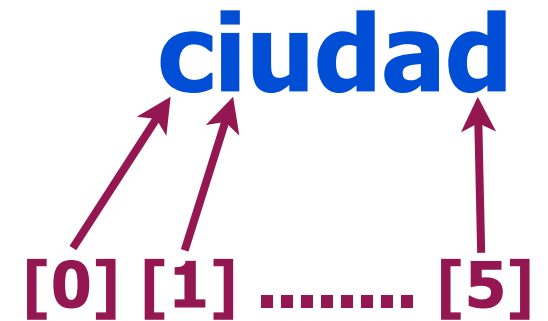


- ◆ Un texto se define en JavaScript con el tipo **string**
 - Un **string** se delimita con **comillas** (dobles) o con **apóstrofes**

- ◆ Ejemplos
 - **"hola, que tal"** o **'hola, que tal'**
 - ◆ 2 strings equivalentes
 - 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>
```

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

◆ Además 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

Ejercicio

◆ Cuantas líneas tiene el string del ejemplo

- 2, 3, 4 o 5

'¿Cuántas líneas \n tiene \u000A este \x27string\x27?'

◆ Como se evalúan las siguientes expresiones

'hola cocacola'.indexOf('la')	=> 1, 2, 3, 4, 9, 10, 11 o 13
'hola cocacola'.length	=> 1, 2, 3, 4, 9, 10, 11 o 13
'hola cocacola'.substring(5,7)	=> 'la', ' c', 'co', 'coc' o 'ca'

◆ Hacer una hoja Web similar a los ejemplos anteriores

- con un script que muestre el primer string
 - ◆ encapsulado dentro de marcas `<pre> .. texto </pre>` de HTML
 - para que mantenga el formato de texto y se vea el número de líneas

Ejercicio UNICODE

Completar el programa JavaScript adjunto, con los códigos UNICODE que generan las letras alfa, beta, gamma y delta del alfabeto griego, tanto en minúsculas y mayúsculas (como se ve en la captura adjunta).

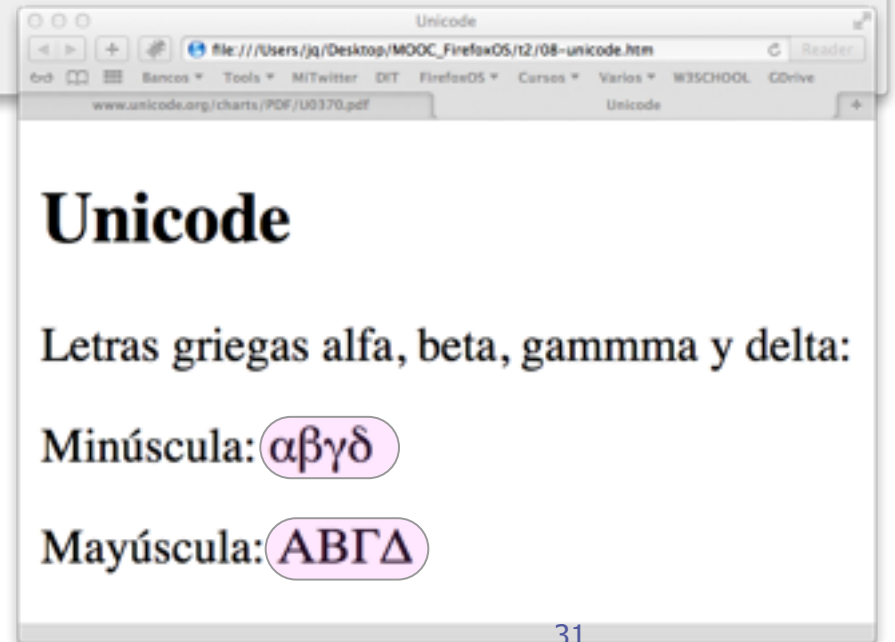
Buscar los códigos en la Web de UNICODE:
<http://www.unicode.org/charts/>

```
08-unicode.htm UNREGISTERED
<!DOCTYPE html>
<html><head>
<title>Unicode</title><meta charset="UTF-8"></head>

<body>
<h2>Unicode</h2>

Letras griegas alfa, beta, gamma y delta: <p>

<script type="text/javascript">
    document.write("Minúscula: \uXXXX...\uXXXX <p>");
    document.write("Mayúscula: \uXXXX...\uXXXX <p>");
</script>
</body>
</html>
```

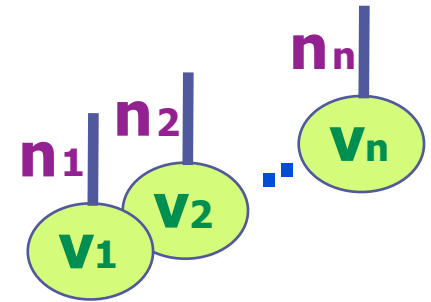




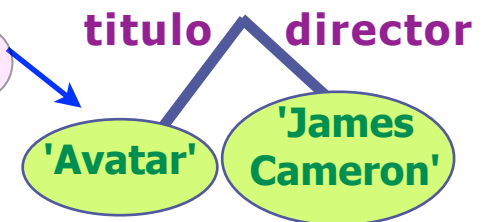
Tema 5.4

Javascript: Objetos de la clase Object

Objetos JavaScript: la clase Object



- ◆ Un objeto es una **colecciones de propiedades** relacionadas
 - Las **propiedades** son variables inicializadas
 - ◆ representadas por un par **nombre:valor**
- ◆ Los nombres de las propiedades de un objeto deben ser todos diferentes
 - y seguir las reglas sintácticas de las variables: **a**, **_method**, **\$1**, ...
- ◆ Los objetos se construyen con literales: **{nombre:valor, ...}**
 - **{}** crea un objeto vacio, igual que el constructor **new Object()**
- ◆ Ejemplo: **{titulo: 'Avatar', director: 'James Cameron'}**



Acceso a propiedades de un objeto

◆ Javascript tiene dos notaciones para referenciar propiedades de objetos

- Notación punto y array

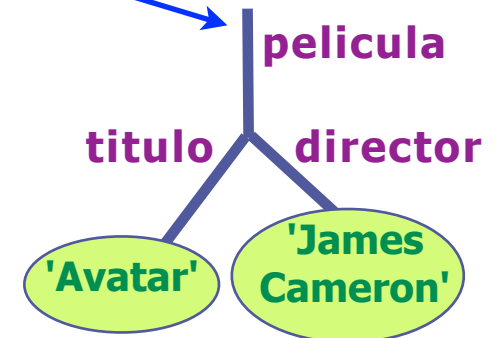
- ◆ Ejemplo de objeto: `var pelicula = {titulo: 'Avatar', director: 'James Cameron'}`

◆ Notación punto:

- `pelicula.titulo`, `pelicula.director`

◆ Notación array:

- `pelicula["titulo"]`, `pelicula["director"]`



Script ilustrativo

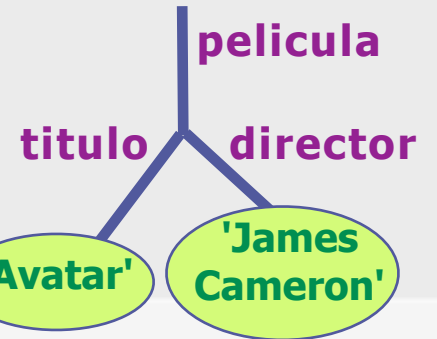
```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo typeof</title>
<meta charset="UTF-8">
</head>
<body>
<pre>
<script type="text/javascript">
```

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

```
document.writeln(" pelicula.titulo    => " + pelicula.titulo);
document.writeln(" pelicula['titulo'] => " + pelicula['titulo']);
document.writeln(" pelicula.director  => " + pelicula.director);
</script>
</pre>
</body>
</html>
```

Ejemplo typeof

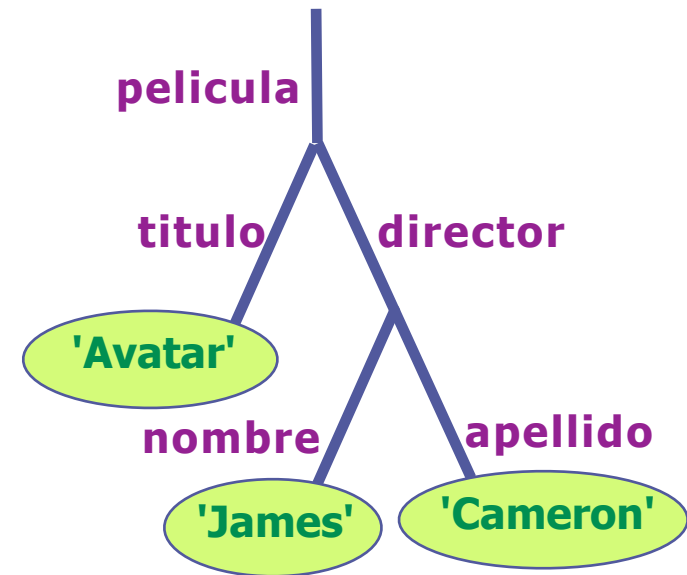
```
pelicula.titulo    => Avatar
pelicula['titulo'] => Avatar
pelicula.director  => James Cameron
```



Objetos anidados: árboles

- ◆ Los objetos pueden **anidarse** entre si
 - Los objetos anidados representan **árboles**
- ◆ La notación punto o array puede **encadenarse**
 - representando un **camino en el árbol**
 - ◆ `pelicula.director.nombre`
 - ◆ `pelicula['director'].apellido`
 - ◆ `pelicula['director']['nombre']`

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



Propiedades dinámicas

◆ Las propiedades de objetos

- son **dinámicas**
 - ◆ Pueden **crearse** y **destruirse**

◆ Operaciones sobre **propiedades**

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

Ejemplo de propiedades dinámicas

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo typeof</title>
<meta charset="UTF-8">
</head>
<body> Propiedades dinámicas
```

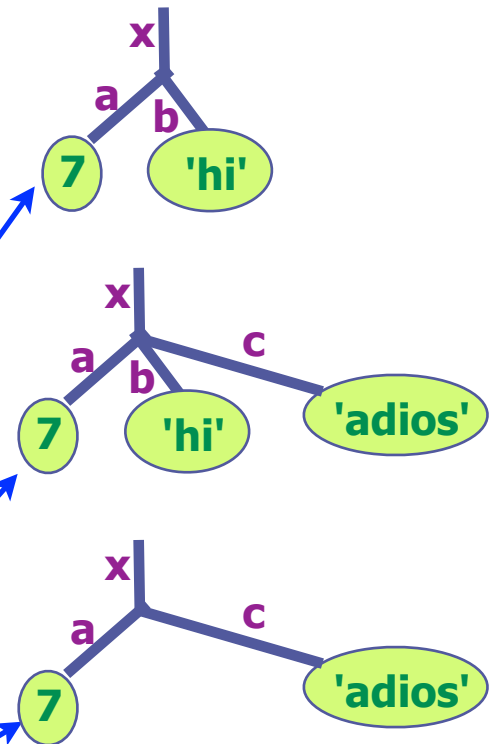
```
<pre>
<script type="text/javascript">
var x = { a:7, b:'hi' }; // Crea variable x
```

```
x.c = 'adiós'; // Crea propiedad x.c
```

```
delete x.b; // Destruye propiedad x.b
```

```
document.writeln(" x.a    => " + x.a);
document.writeln(" x.b    => " + x.b);
document.writeln(" x.c    => " + x.c);
</script>
```

```
</pre>
</body>
</html>
```



Ejemplo typeof

Propiedades dinámicas

<code>x.a</code>	<code>=> 7</code>
<code>x.b</code>	<code>=> undefined</code>
<code>x.c</code>	<code>=> adiós</code>

. **[]**
new
()
++ **--**
! **~**
+ **-**
delete
typeof **void**
***** **/** **%**
+
+ **-**
<< **>>** **>>>**
< **<=** **>** **>=**
instanceof **in**
== **!=** **===** **!==**
&
^
|
&&
||
?:
=
OP=
,

Acceso a propiedad o invocar método; índice a array
Crear objeto con constructor de clase
Invocación de función/método o agrupar expresión
Pre o post auto-incremento; pre o post auto-decremento
Negación lógica (NOT); complemento de bits
Operador unitario, números. signo positivo; signo negativo
Borrar propiedad de un objeto
Devolver tipo; valor indefinido
Números. Multiplicación; división; modulo (o resto)
Concatenación de string
Números. Suma; resta
Desplazamientos de bit
Menor; menor o igual; mayor; mayor o igual
¿objeto pertenece a clase?; ¿propiedad pertenece a objeto?
Igualdad; desigualdad; identidad; no identidad
Operacion y (AND) de bits
Operacion ó exclusivo (XOR) de bits
Operacion ó (OR) de bits
Operación lógica y (AND)
Operación lógica o (OR)
Asignación condicional
Asignación de valor
Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=
Evaluación múltiple

Operadores JavaScript

Los operadores están ordenados
 verticalmente por prioridades.
 Los más altos se evalúan antes.

Ejercicio

- ◆ Indicar el resultado de evaluar las siguientes expresiones

```
var x = {a:{b:{c:3, d:4}, e:5}, f:6};
```

x.a.b.c => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

x.a.c.b => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

x.a.e => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

x.a.b => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

x.a.f => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

x.f => se evalúa a 3, 4, 5, 6, {c:3,d:4} o undefined

- ◆ Hacer una hoja Web similar a los ejemplos anteriores, que incluya
 - un script que muestre como JavaScript evalúa estas expresiones

Nombres de propiedades con strings arbitrarios

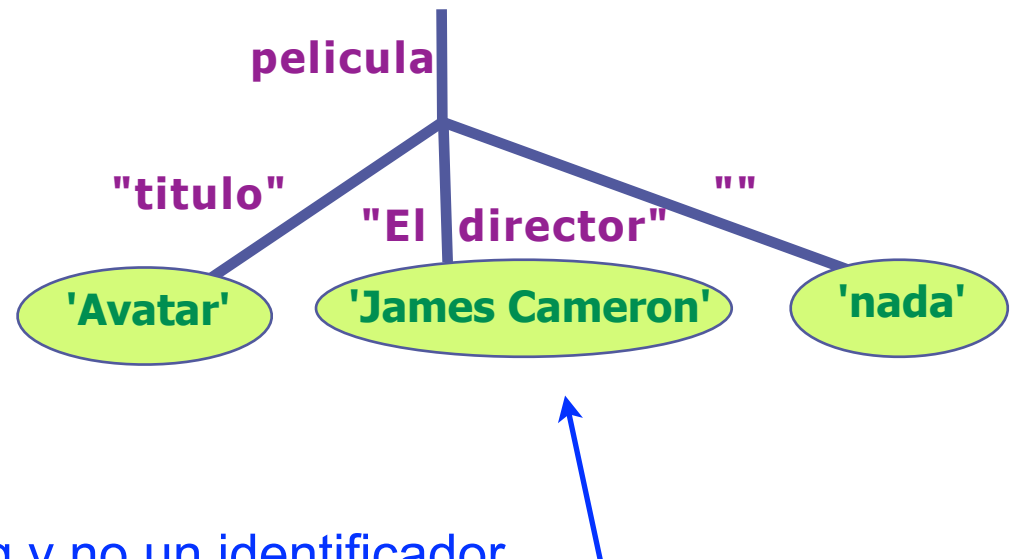
- ◆ Los strings arbitrarios pueden utilizarse como nombres en
 - en **Notación array** y en **Literales de objeto**
 - ◆ Por ejemplo: **"El director"**, **""**

- ◆ **Notación array**

- **pelicula["titulo"]**
- **pelicula["El director"]**
- **pelicula[""]**

- ◆ **Literales de objetos**

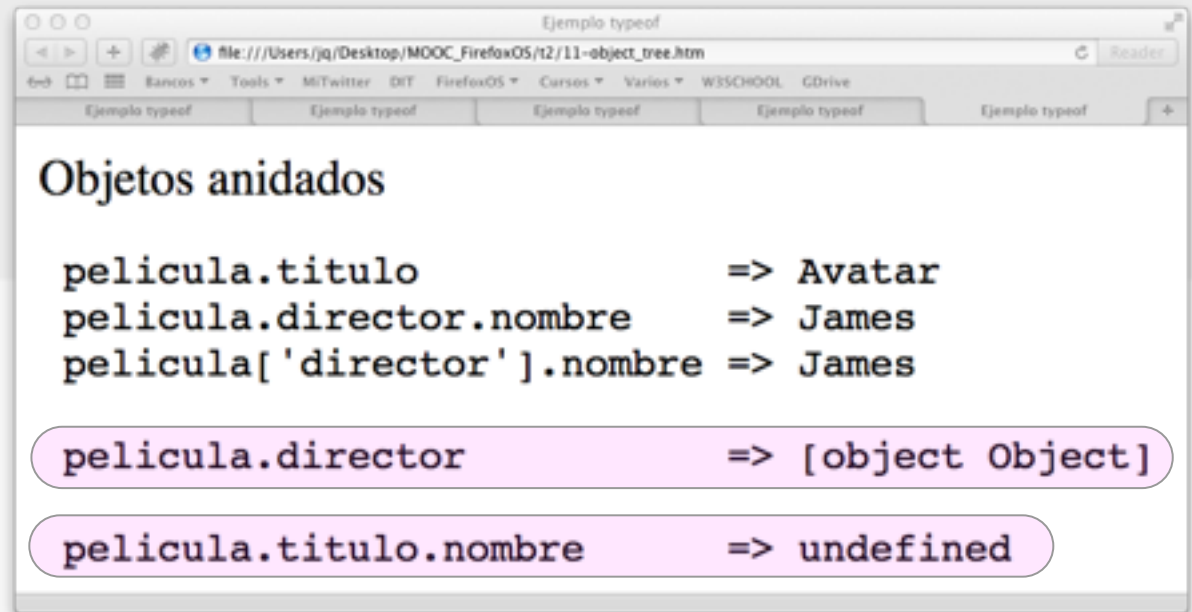
- cuando el nombre es un string y no un identificador
 - ◆ Ejemplo: **var pelicula = { "titulo":'Avatar', "El director":'James Cameron', "':'nada' }**



Script ilustrativo

```
<!DOCTYPE html>
<html>
<head>
<title>Ejemplo typeof</title>
<meta charset="UTF-8">
</head>
<body>
Objetos anidados
<pre>
<script type="text/javascript">
var pelicula = {
  titulo: 'Avatar',
  director:{
    nombre: 'James',
    apellido: 'Cameron'
  }
};
```

```
document.writeln(" pelicula.titulo           => " + pelicula.titulo);
document.writeln(" pelicula.director.nombre    => " + pelicula.director.nombre);
document.writeln(" pelicula['director'].nombre => " + pelicula['director'].nombre);
document.writeln();
document.writeln(" pelicula.director          => " + pelicula.director);
document.writeln();
document.writeln(" pelicula.titulo.nombre    => " + pelicula.titulo.nombre);
</script>
</pre>
</body>
</html>
```



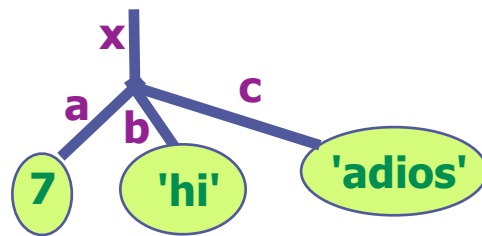


Tema 5.5

Javascript: Sentencia for/in

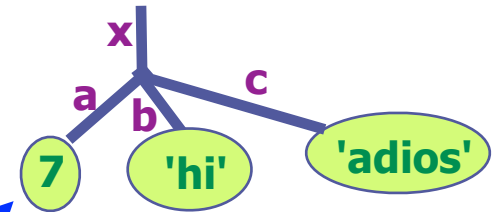
Sentencia for/in

- ◆ **for (i in x) {..bloque de instrucciones..}**
 - itera en todas las propiedades del objeto **x**
 - ◆ "i" contiene el nombre de la propiedad en cada iteración



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];
```

STATEMENT SINTAXIS

block	{ statements };
break	break [label];
case	case expression:
continue	continue [label];
debugger	debugger;
default	default:
do/while	do statement while(expression);
empty	;
expression	expression;
for	for(init; test; incr) statement
for/in	for (var in object) statement
function	function name([param[,...]]) { body }
if/else	if (expr) statement1 [else statement2]
label	label: statement
return	return [expression];
switch	switch (expression) { statements }
throw	throw expression;
try	try {statements} [catch { statements }] [finally { statements }]
strict	"use strict";
var	var name [= expr] [,...];
while	while (expression) statement
with	with (object) statement

DESCRIPCIÓN DE LA SENTENCIA JAVASCRIPT

Agrupar un bloque de sentencias como 1 sentencia
Salir del bucle o switch o sentencia etiquetada
Etiquetar sentencia dentro de sentencia switch
Salto a sig. iteración de bucle actual/etiquetado
Punto de parada (breakpoint) del depurador
Etiquetar setencia default en sentencia switch
Alternativa al bucle while con condición al final

Sentencia vacía, no hace nada

Evaluar expresión (con efectos laterales)

Bucle sencillo. "init": inicialización;

"test": condición; "incr": acciones final bucle

Enumerar las propiedades del objeto "object"

Declarar una función llamada "name"

Ejecutar statement1 o statement2

Etiquetar sentencia con nombre "label"

Devolver un valor desde una función

Multiopción con etiquetas "case" o "default"

Lanzar una excepción

Gestionar excepciones

Activar restricciones strict a script o función

Declarar e inicializar una o mas variables

Bucle básico con condición al principio

Extender cadena de ámbito (no recomendado)

Ejercicio

- ◆ Crear una página HTML con un script
 - que cree el objeto
 - ◆ `var y = {x:'hola', y:'que', z:'tal', t: 'estás'}`
- ◆ El script debe concatenar despues
 - los strings que contienen sus propiedades en un único string
 - con los valores separados por una coma y un espacio en blanco
- ◆ Mostrando el string concatenado en la página HTML



Tema 5.6

Javascript: boolean, igualdad y sentencia if/else

Tipo boolean

FALSE
true

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

!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: igualdad de referencias
 - Inútil si no se redefine
 - 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 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**

false && true	=> false
0 && true	=> 0
1 && 5	=> 5
13 0	=> 13
undefined 0	=> 0
null 0	=> 0
(7)? 0 : 1	=> 0
("")? 0 : 1	=> 1

. []
 new
 ()
 ++ --
 ! ~
 + -
 delete
 typeof void
 * / %
 +
 + -
 << >> >>>
 < <= > >=
 instanceof in
 == != === !==
 &
 ^
 |
 &&
 ||
 ?:
 =
 OP=
 ,

Acceso a propiedad o invocar método; índice a array
 Crear objeto con constructor de clase
 Invocación de función/método o agrupar expresión
 Pre o post auto-incremento; pre o post auto-decremento
 Negación lógica (NOT); complemento de bits
 Operador unitario, números. signo positivo; signo negativo
 Borrar propiedad de un objeto
 Devolver tipo; valor indefinido
 Números. Multiplicación; división; modulo (o resto)
 Concatenación de string
 Números. Suma; resta
 Desplazamientos de bit
 Menor; menor o igual; mayor; mayor o igual
 ¿objeto pertenece a clase?; ¿propiedad pertenece a objeto?
 Igualdad; desigualdad; identidad; no identidad
 Operacion y (AND) de bits
 Operacion ó exclusivo (XOR) de bits
 Operacion ó (OR) de bits
 Operación lógica y (AND)
 Operación lógica o (OR)
 Asignación condicional
 Asignación de valor
 Asig. con operación: += -= *= /= %= <<= >>= >>>= &= ^= |=
 Evaluación múltiple

Operadores JavaScript

Los operadores están ordenados
 verticalmente por prioridades.
 Los más altos se evalúan antes.

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

STATEMENT SINTAXIS

block	{ statements };
break	break [label];
case	case expression:
continue	continue [label];
debugger	debugger;
default	default:
do/while	do statement while(expression);
empty	;
expression	expression;
for	for(init; test; incr) statement
for/in	for (var in object) statement
function	function name([param[,...]]) { body }
if/else	if (expr) statement1 [else statement2]
label	label: statement
return	return [expression];
switch	switch (expression) { statements }
throw	throw expression;
try	try {statements} [catch { statements }] [finally { statements }]
strict	"use strict";
var	var name [= expr] [,...];
while	while (expression) statement
with	with (object) statement

DESCRIPCIÓN DE LA SENTENCIA JAVASCRIPT

Agrupar un bloque de sentencias como 1 sentencia
Salir del bucle o switch o sentencia etiquetada
Etiquetar sentencia dentro de sentencia switch
Salto a sig. iteración de bucle actual/etiquetado
Punto de parada (breakpoint) del depurador
Etiquetar setencia default en sentencia switch
Alternativa al bucle while con condición al final

Sentencia vacía, no hace nada

Evaluar expresión (con efectos laterales)

Bucle sencillo. "init": inicialización;

"test": condición; "incr": acciones final bucle

Enumerar las propiedades del objeto "object"

Declarar una función llamada "name"

Ejecutar statement1 o statement2

Etiquetar sentencia con nombre "label"

Devolver un valor desde una función

Multiopción con etiquetas "case" o "default"

Lanzar una excepción

Gestionar excepciones

Activar restricciones strict a script o función

Declarar e inicializar una o mas variables

Bucle básico con condición al principio

Extender cadena de ámbito (no recomendado)

Ejercicio

◆ Indicar el resultado de evaluar las siguientes expresiones

■ Tabla de conversión de tipos

- ◆ <https://www.inkling.com/read/javascript-definitive-guide-david-flanagan-6th/chapter-3/type-conversions>

!10	=> se evalúa a 0, 10, null, undefined, false o true
!!null	=> se evalúa a 0, 10, null, undefined, false o true
!(0 === "")	=> se evalúa a 0, 10, null, undefined, false o true
true ? 10 : null	=> se evalúa a 0, 10, null, undefined, false o true
(0 undefined) && true	=> se evalúa a 0, 10, null, undefined, false o true
(!0 !undefined) && !true	=> se evalúa a 0, 10, null, undefined, false o true

◆ Hacer una hoja Web similar a los ejemplos anteriores, que incluya

- un script que muestre como JavaScript evalúa estas expresiones