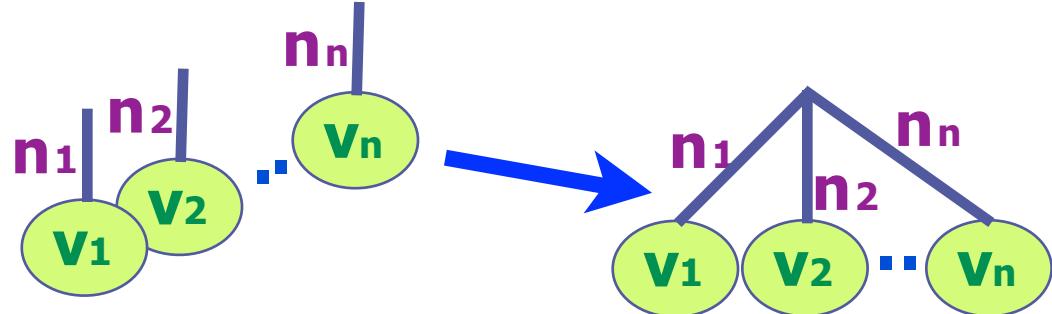




Objetos, propiedades y métodos

Objetos



◆ Los objetos son colecciones de variables

- agrupadas como un elemento estructurado que llamamos objeto
 - ◆ Las variables de un objeto se denominan **propiedades**

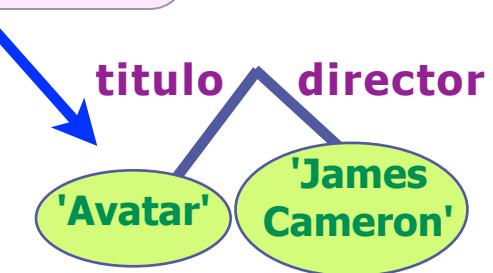
◆ Una **propiedad** es un par **nombre:valor** donde

- los **nombres** deben ser **todos diferentes** en un mismo objeto

◆ Se definen con el literal: { **nombre:valor**, ... }

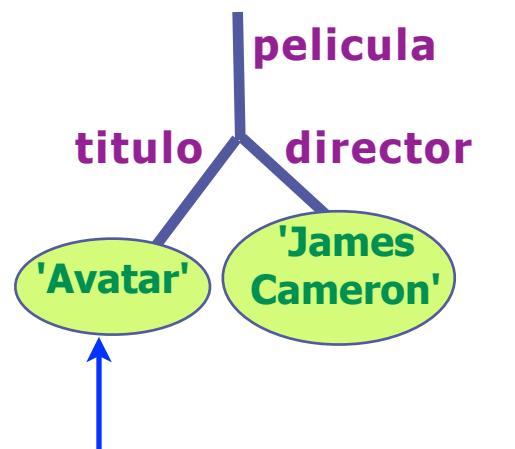
- **Por ejemplo:** {**titulo: 'Avatar'**, **director: 'James Cameron'**}

- ◆ crea un objeto con 2 propiedades:
 - **titulo:'Avatar'**
 - **director:'James Cameron'**



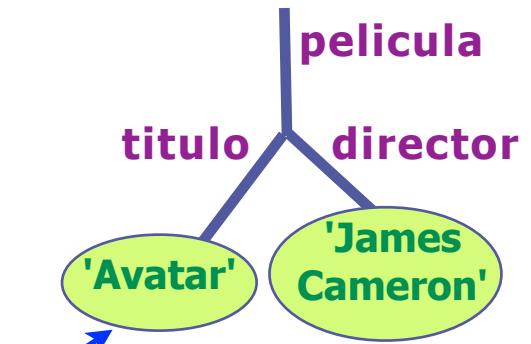
Propiedades

- ◆ El acceso a propiedades utiliza el operador punto
 - **obj.propiedad**
- ◆ Por ej. en: **var pelicula = {titulo: 'Avatar', director: 'James Cameron'}**
 - **pelicula.titulo** => "Avatar"
 - **pelicula.director** => "James Cameron"
 - **pelicula.fecha** => undefined // la propiedad fecha no existe
- ◆ Aplicar el operador punto sobre **undefined** o **null**
 - Provoca un **Error_de_ejecución** y aborta la ejecución del programa
- ◆ La notación punto solo acepta nombres de propiedades
 - Con la sintaxis de variables: **a, _method, \$1, ...**
 - No son utilizables: "#43", "?a=1",



Notación array

- ◆ La notación array es equivalente a la notación punto
 - **pelicula["titulo"]** es equivalente a **pelicula.titulo**
 - ◆ Al acceder a: **var pelicula = {titulo: 'Avatar', director: 'James Cameron'}**
- ◆ La notación array permite utilizar **strings arbitrarios** como nombres
 - por ejemplo, **objeto["El director"]**, **pelicula[""]** o **a["%43"]**
 - ◆ OJO! es conveniente utilizar siempre nombres compatibles con notación punto
- ◆ Nombres (strings) arbitrarios son posibles también en un literal de objeto:
 - Por ejemplo, **{"titulo": 'Avatar', "El director": 'James Cameron'}**



Nombres de propiedades como variables

- ◆ La notación array permite acceder también a propiedades
 - cuyo nombre esta en una variable en forma de string
 - ◆ Esto no es posible con la notación punto

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

```
x.titulo;      => 'Avatar'  
x['titulo'] ;  => 'Avatar'
```

```
var p = 'titulo'; // inicializada con string 'titulo'
```

```
x[p] ;        => 'Avatar'
```

```
x.p;          => undefined
```

x tiene una propiedad de nombre '**titulo**', que es el string que contiene p

El **objeto x** no tiene ninguna propiedad de nombre **p** y devuelve **undefined**

Downloads ⌘J
 Add-ons ⌘A
 Test Pilot ▶
 Set Up Sync... ▶
 GrabMyBooks ▶
 Geolocator ▶
Web Developer ▶
 Page Info ⌘I
 Delicious Options
 Live HTTP headers



The new Firefox for Android Home screen helps you navigate the Web quickly. [Install Firefox for Android!](#)

La consola nos va mostrando el resultado de ejecutar las sentencias JavaScript

Aquí se introduce la sentencia

Number – JavaScript | MDN Miriada X: Desarrollo en H... + https://www.miriadax.net/web/html5mooc/edicion?p_p_id=execactivity_WAR_liferaylmsportlet&p_p_lif...

Bancos Varios Cursos Tools Rails HTML5 HTML5-RTC MiTwitter

Indicar el resultado de evaluar las siguientes expresiones, despues de haber declarado las variables fruta y nada

```
var fruta = {peras:3, kiwis:7, fresas:5};
var nada = undefined;
```

Para superar este test, has de responder correctamente al menos el 60% de las preguntas.

fruta["peras"] se evalúa a:

3
 5
 7
 undefined
 Error_de_ejecución

Console Inspector Debugger Style Editor Profiler Network

Net CSS JS Security Logging Clear

```
< var fruta = {peras:3, kiwis:7, fresas:5};  

< undefined  

< fruta["peras"]  

< 3  

fruta["peras"]
```

Clases y herencia

- ◆ Todos los objetos de JavaScript pertenecen a la **clase Object**
 - Javascript posee mas clases predefinidas que derivan de Object
 - ◆ **Date, Number, String, Array, Function,**
 - ◆ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Predefined_Core_Objects
 - Un objeto hereda los métodos y propiedades de su clase
- ◆ Un **método** es una operación (~función) invocable sobre un objeto
 - Se invoca con la notación punto: **objeto.metodo(..params..)**
- ◆ Todas las clases tienen un constructor con el nombre de la clase
 - que permite crear objetos con el operador **new**
 - ◆ Por ejemplo, **new Object()** crea un objeto vacío equivalente a **{}**

Métodos de la clase

- ◆ Un objeto **hereda** métodos de su **clase**, por ejemplo
 - los objetos de la clase **Date** heredan métodos como
 - ◆ **toString()**, **getDay()**, **getFullYear()**, **getHours()**, **getMinutes()**, (ver ejemplo)
 - ◆ https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Date
- ◆ Solo se puede invocar métodos heredados o definidos en un objeto
 - Invocar un método **no heredado ni definido en un objeto**
 - ◆ provoca un **error_de_ejecución**

```
var fecha = new Date();
```

```
fecha.toString()      => Fri Aug 08 2014 12:34:36 GMT+0200 (CEST)
fecha.getHours()     => 12
fecha.getMinutes()   => 34
fecha.getSeconds()  => 36
```

Definición de un nuevo método de un objeto

- ◆ Los métodos se pueden definir también directamente en un objeto
 - El nuevo método solo se define para ese objeto (no es de la clase)
- ◆ Invocar un método cambia el **entorno de ejecución** de JavaScript
 - pasando a ser el **objeto invocado**, que se referencia con **this**
 - ◆ **this.titulo** referencia la propiedad **titulo** del objeto **pelicula**

```
var pelicula = {  
    titulo:'Avatar',  
    director:'James Cameron',  
  
    resumen:function (){  
        return "El director de " + this.titulo + " es " + this.director;  
    }  
}
```

pelicula.resumen() => "El director de Avatar es James Cameron"

Algunas Clases predefinidas

◆ Object

- Clase raíz, suele usarse el literal: `{a:3, b:"que tal"}`

◆ Array

- Colección indexable, suele usarse el literal: `[1, 2, 3]`

◆ Date

- Hora y fecha extraída del reloj del sistema: `new Date()`

◆ Function

- Encapsula código, suele usarse literal o def.: `function (x) {....}`

◆ RegExp

- Expresiones regulares, suele usarse el literal: `/(hola)+$/`

◆ Math

- Modulo con **constants y funciones matemáticas**

◆ Number, String y Boolean

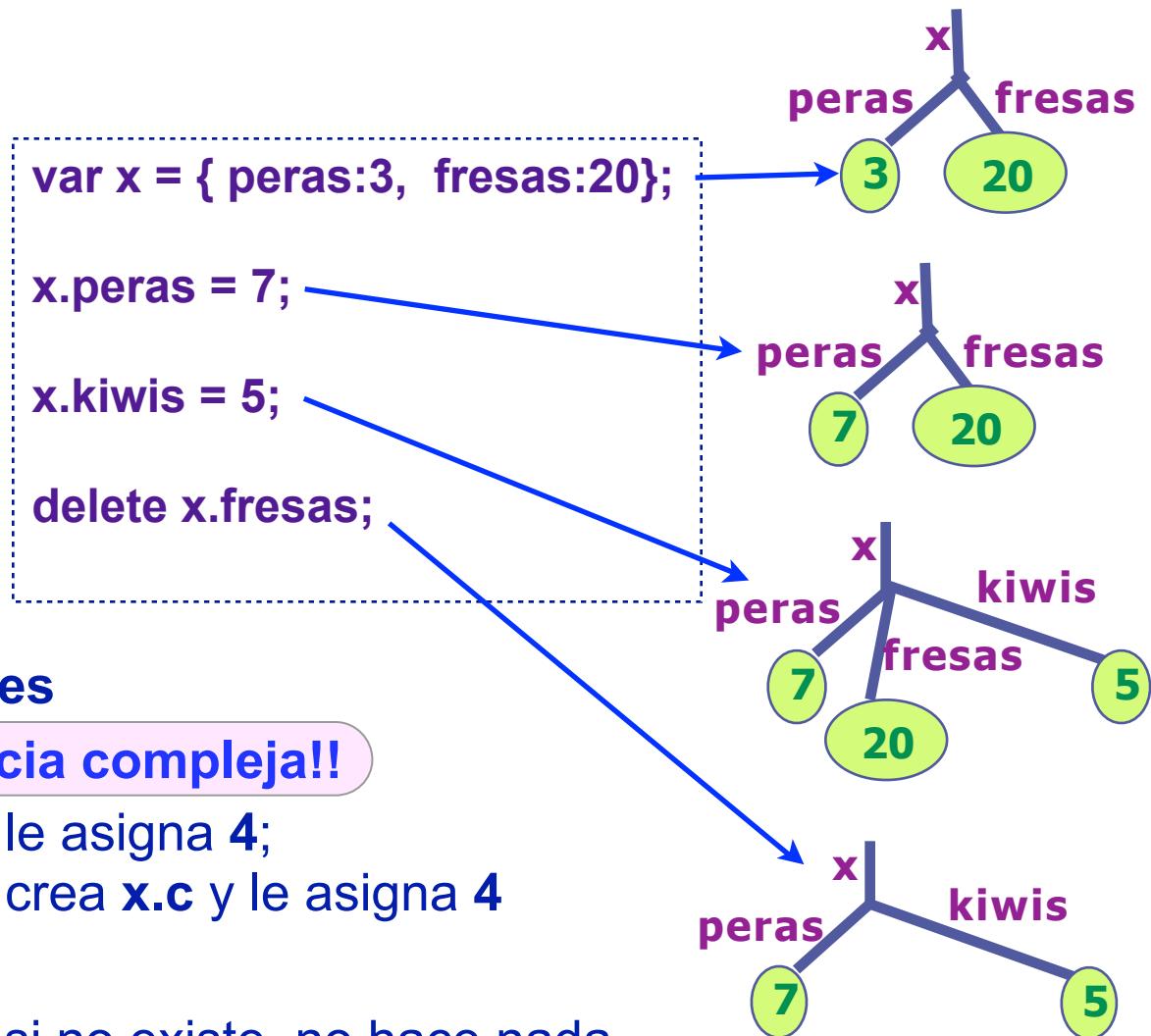
- Clases que encapsulan valores de los tipos **number, string y boolean** como objetos
 - Sus métodos se aplican a los tipos directamente, la conversión a objetos es automática



Objetos: características avanzadas

Propiedades dinámicas

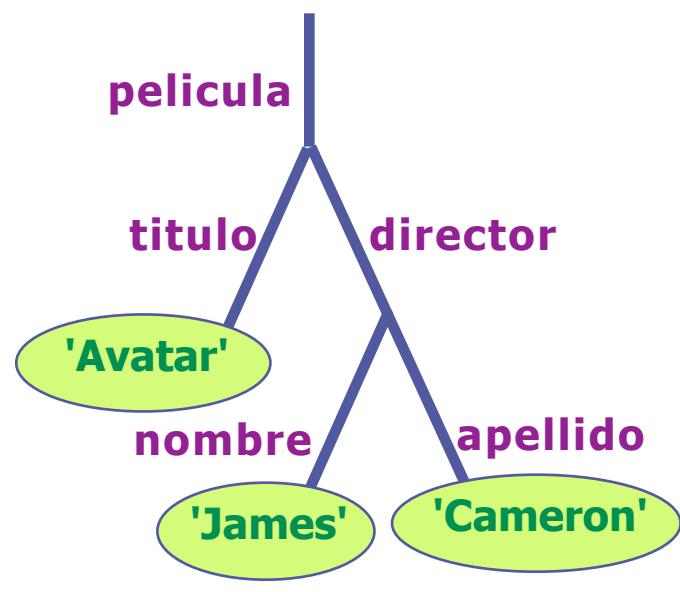
- ◆ Las propiedades de objetos
 - Pueden **crearse**
 - Pueden **destruirse**
- ◆ Operaciones sobre **propiedades**
 - **x.c = 4** **¡¡OJO: sentencia compleja!!**
 - ◆ si propiedad **x.c** existe, le asigna **4**;
 - ◆ si **x.c** no existe, crea **x.c** y le asigna **4**
 - **delete x.c**
 - ◆ si existe **y.c**, la elimina; si no existe, no hace nada
 - **"c" in x**
 - ◆ si **x.c** existe, devuelve **true**, sino devuelve, **false**



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**
 - ◆ Las siguientes expresiones se evalúan así:
 - `pelicula.director.nombre` => 'James'
 - `pelicula['director']['nombre']` => 'James'
 - `pelicula['director'].apellido` => 'Cameron'
 - `pelicula.estreno` => undefined
 - `pelicula.estreno.año` => Error_de_ejecución

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



Usar propiedades dinámicas

- ◆ Las propiedades dinámicas de JavaScript
 - son muy útiles si se utilizan bien
- ◆ Un objeto solo debe definir las propiedades
 - que contengan información conocida
 - ◆ añadirá mas solo si son necesarias
- ◆ La información se puede consultar con
 - **prop1 && prop1.prop2**
 - ◆ para evitar errores de ejecución
 - ◆ si las propiedades no existen

// Dado un objeto **pel** definido con

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

// se puede añadir **pel.estreno** con

```
pel.estreno = {  
    año: '2009',  
    cine: 'Tivoli'  
}
```

// La expresión

pel.estreno && pel.estreno.año

// devuelve **pel.estreno** o **undefined**,
// evitando **ErrorDeEjecución**, si
// **pel.estreno** no se hubiese creado

```

var x = {} // x e y tienen la
var y = x; // misma referencia

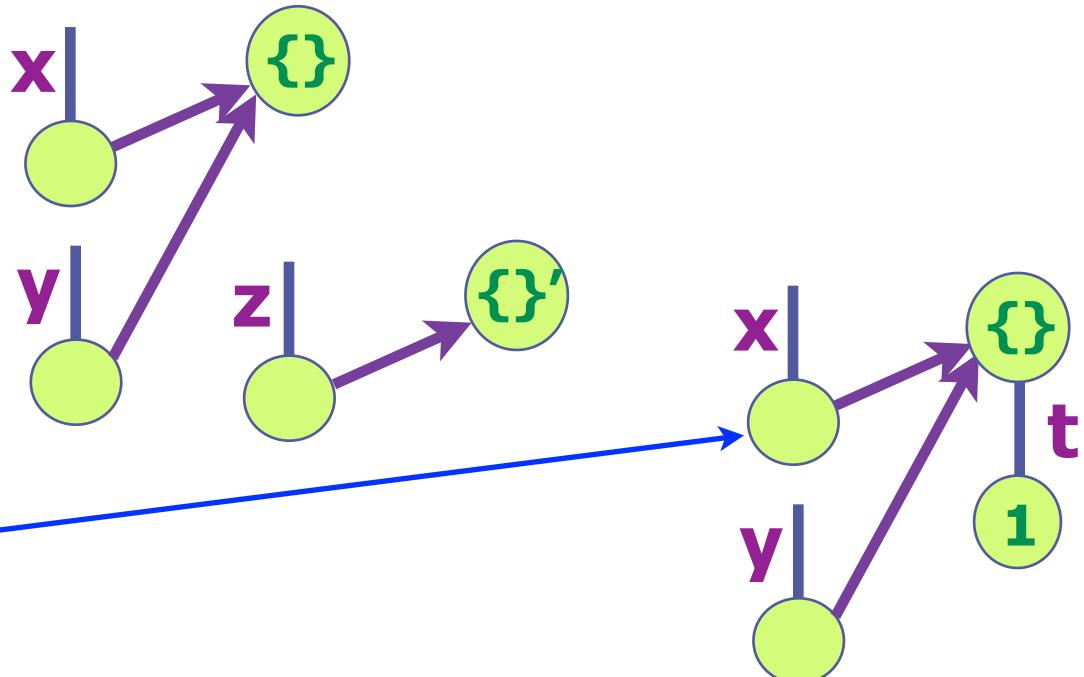
var z = {} // la referencia a z
// es diferente de
// la de x e y

x.t = 1;

x.t    => 1 // x accede al mismo
y.t    => 1 // objeto que y
z.t    => undefined

```

Referencias a objetos



- ◆ Las variables que contienen objetos
 - solo contienen la referencia al objeto
- ◆ El objeto esta en otro lugar en memoria
 - indicado por la referencia
- ◆ Esto produce efectos laterales
 - como ilustra el ejemplo

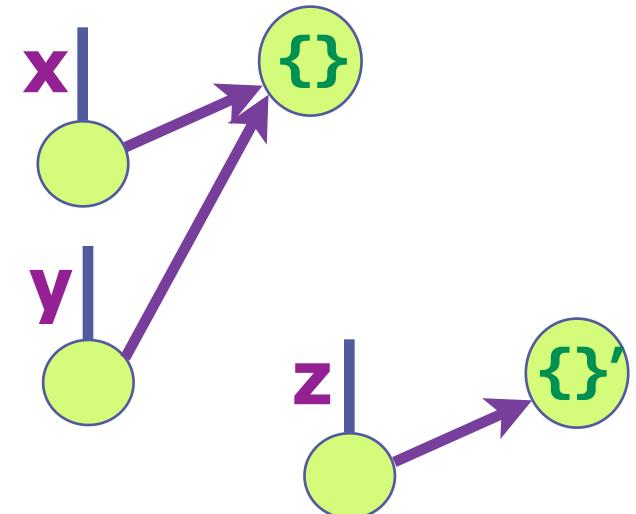
Identidad de objetos

- ◆ Las referencias a objetos afectan a la identidad
 - porque identidad de objetos
 - ◆ es identidad de referencias
 - los objetos no se comparan
 - ◆ se comparan solo las referencias
 - es poco útil si no se redefine
- ◆ Igualdad (debil) de objetos `==` y `!=`
 - no tiene utilidad tampoco con objetos
 - ◆ no se debe utilizar

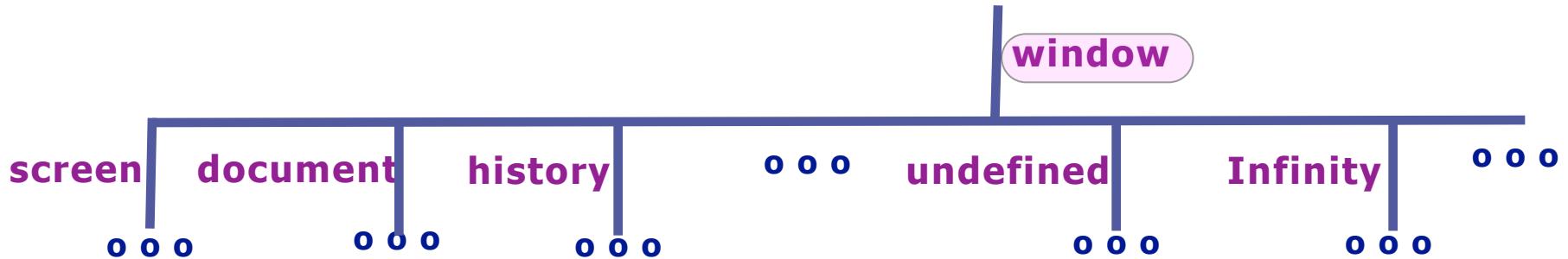
```
var x = {} // x e y contienen la  
var y = x; // misma referencia
```

```
var z = {} // la referencia a z  
// es diferente de x e y
```

<code>x === y</code>	=> true
<code>x === {}</code>	=> false
<code>x === z</code>	=> false

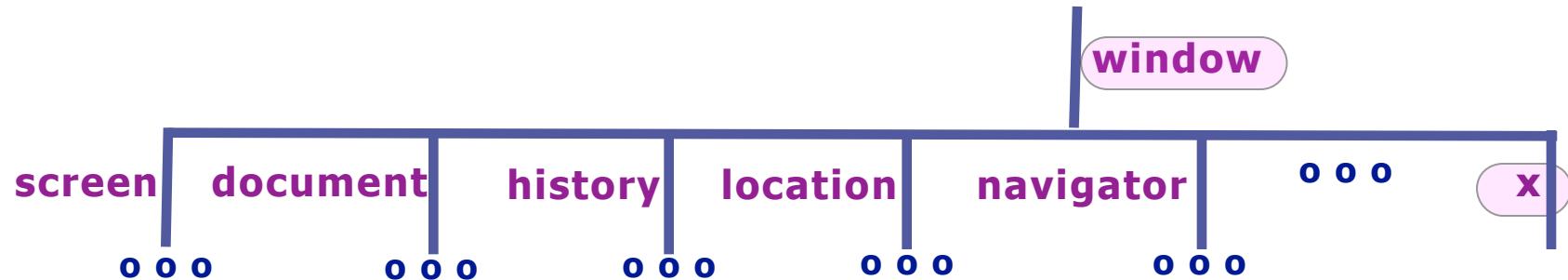


Objeto window o this



- ◆ El entorno de ejecución de JavaScript es el **objeto global window**
 - El **objeto global window** tiene propiedades con información sobre
 - ◆ Objetos predefinidos de JavaScript, el **navegador**, el **documento HTML**,
- ◆ **window** se referencia también como **this** en el entorno global
 - La **propiedad document** de **window** se referencia como
 - ◆ **window.document**, **this.document** o **document**
- ◆ Documentación: <https://developer.mozilla.org/en-US/docs/Web/API/Window>

Variables globales y el entorno de ejecución

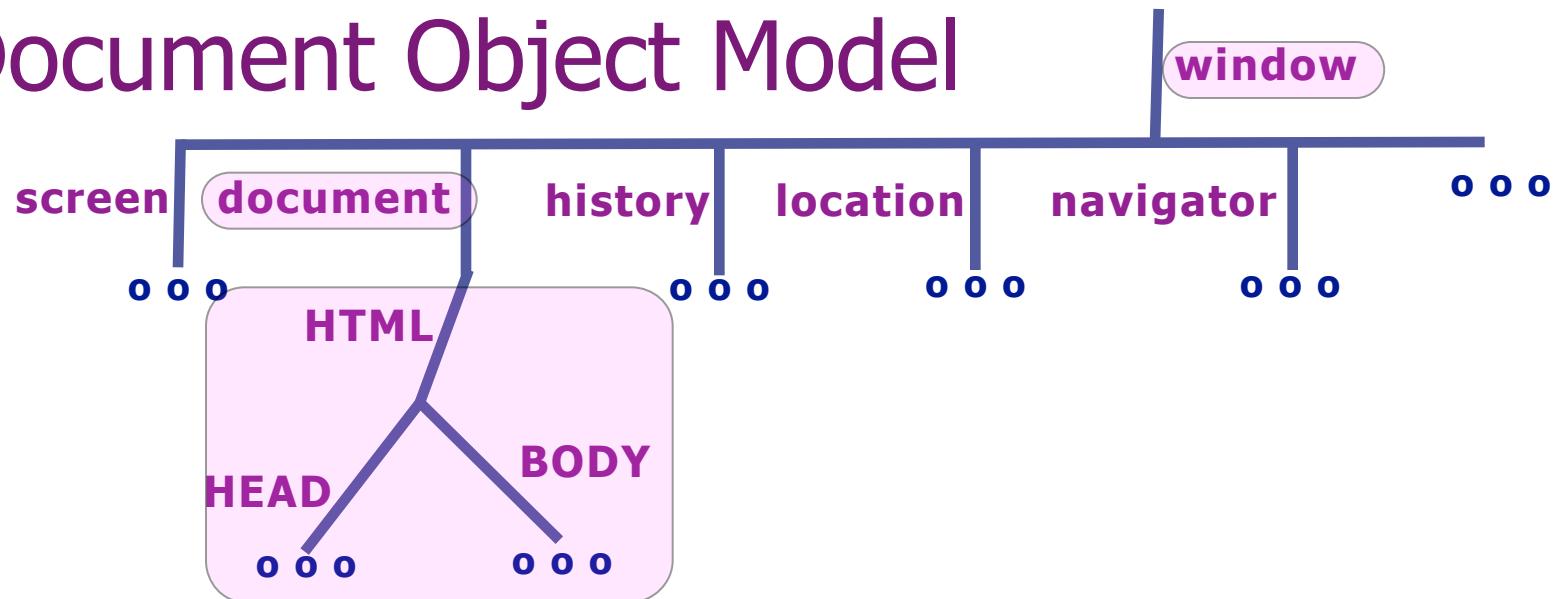


- ◆ Un programa JavaScript se ejecuta con el objeto **window** como entorno
 - una asignación a una variable no definida como **x = 1;**
 - ◆ Crea una nueva **propiedad de window** de nombre **x**, porque
 - **x = 1;** es equivalente a **this.x = 1;** y a **window.x = 1;**
- ◆ Olvidar definir una variable, es un error muy habitual
 - y al asignar un valor a la variable no definida, JavaScript no da error
 - ◆ sino que crea una nueva **propiedad de window**
 - Es un error de diseño de JavaScript y hay que tratar de evitarlo



DOM: Document Object Model

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
 - ◆ **Doc:** <https://developer.mozilla.org/en/docs/Web/API/Document>
- ◆ 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:** nombre puede coincidir 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 diagram illustrates the DOM structure and its representation in a browser. On the left, a code editor shows the HTML and JavaScript code for the page. The HTML includes a title, meta charset, and a body section with an

element and a element. The JavaScript uses `document.getElementById` to get the `div#fecha` element and sets its `innerHTML` to the current date and time. On the right, a browser window titled "Ejemplo fecha y hora" displays the resulting page. The page content is "La fecha y hora son:" followed by the current date and time: "Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)". A large purple arrow on the right points from the text "Acceso a DOM" towards 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>

```

Ejemplo fecha y hora

La fecha y hora son:

Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)

Acceso a DOM

```

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

<script type="text/javascript">

function mostrar_fecha( ) {
    var cl = document.getElementById("fecha");
    cl.innerHTML = new Date( );
}

</script>

</head>

<body>
<h2>Ejemplo con función</h2>
<div id="fecha"></div>

<script type="text/javascript">
mostrar_fecha( );      // Llamar función
</script>

</body>
</html>

```

Varios scripts

- ◆ Una página
 - con varios scripts
 - ◆ es un único programa
- ◆ Scripts se juntan siguiendo el orden de aparición en la página



- ◆ función mostrar_fecha()
 - Se define e invoca en scripts diferentes

- ◆ 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

The diagram illustrates the interaction between an HTML file and a web browser. On the left, a code editor shows the content of the file "22-date_id.htm". The code includes an HTML structure with a title, meta charset, and body containing an h2 element and a div with id="fecha". A script block uses the window.fecha property to set the innerHTML of the #fecha div to the current date and time. A dashed arrow points from the "fecha" id in the HTML to the "fecha" property in the script. On the right, a screenshot of a Firefox browser window titled "Ejemplo fecha y hora" displays the output: "La fecha y hora son:" followed by the current date and time ("Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)").

```

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

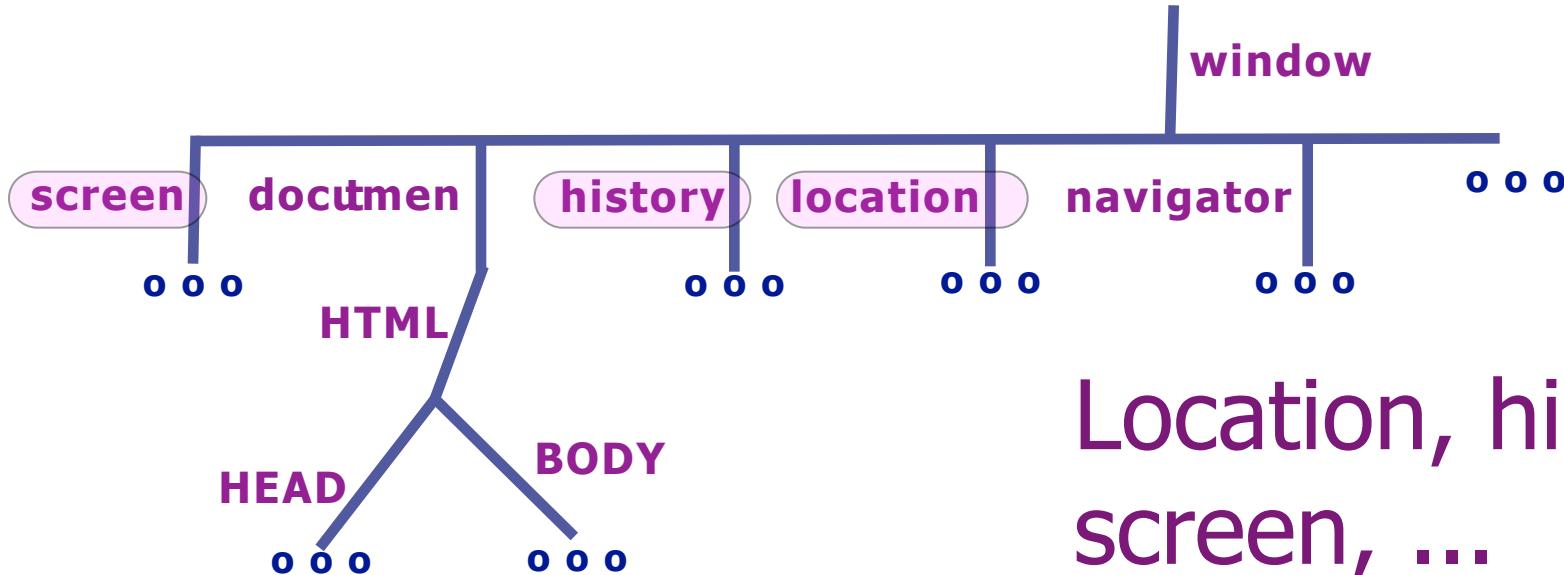
<body>
<h2>La fecha y hora son:</h2>
<div id="fecha"></div>
<script type="text/javascript">
  fecha.innerHTML = new Date( );
  // Insertar Date en elem con id="fecha"
</script>
</body>
</html>

```

La fecha y hora son:

Sun Sep 08 2013 12:46:50 GMT+0200 (CEST)

Acceso a DOM



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: <https://developer.mozilla.org/en-US/docs/Web/API/Window>

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
 - ◆ El objeto DOM con el identificador buscado o null si no lo encuentra
 - ¡Un identificador solo puede estar en un objeto de una página HTML!

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

- Devuelven una matriz de objetos
 - ◆ Por ejemplo: getElementsByName("my_name")[0]
 - referencia el primer elemento con atributo **name="my_name"**
- **Doc:** <https://developer.mozilla.org/en/docs/Web/API/Document>

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

La fecha y hora son:

hola

08:52:41.433 | ⚠ MutationObserver instead.

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

08:52:41.435 | » "hola"

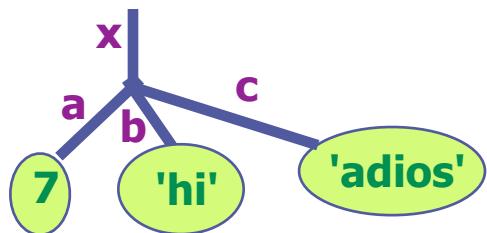


Sentencia for/in de JavaScript

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

The diagram shows a hierarchical tree structure where node 'x' branches into nodes 'a', 'b', and 'c'. Node 'a' has value '7', node 'b' has value 'hi', and node 'c' has value 'adios'.

Code:

```
<!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>
```

Output:

Sentencia for/in:

Propiedad a = 7
Propiedad b = hi
Propiedad c = adios

© Juan Quemada, DIT, UPM

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 delimitados con **{..}**

The screenshot shows a browser developer tools console window titled "14-for_in_bloque.js" with the status "UNREGISTERED". It contains the following code examples:

- // Utilizar notación 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];
```

```

<!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+="|  |  |
| --- | --- |
| " + i + " | = " + screen[i] + " |
";
}
</script>
</body>
</html>

```

window.screen

Screen

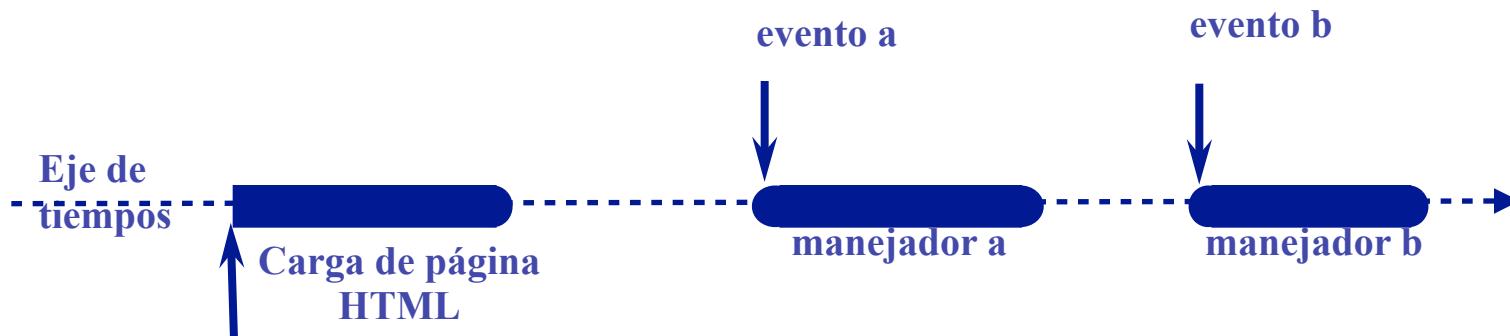
Propiedad	Valor
availWidth	= 2514
availHeight	= 1418
availTop	= 22
availLeft	= 46
pixelDepth	= 24
colorDepth	= 24
width	= 2560
height	= 1440



Eventos Javascript

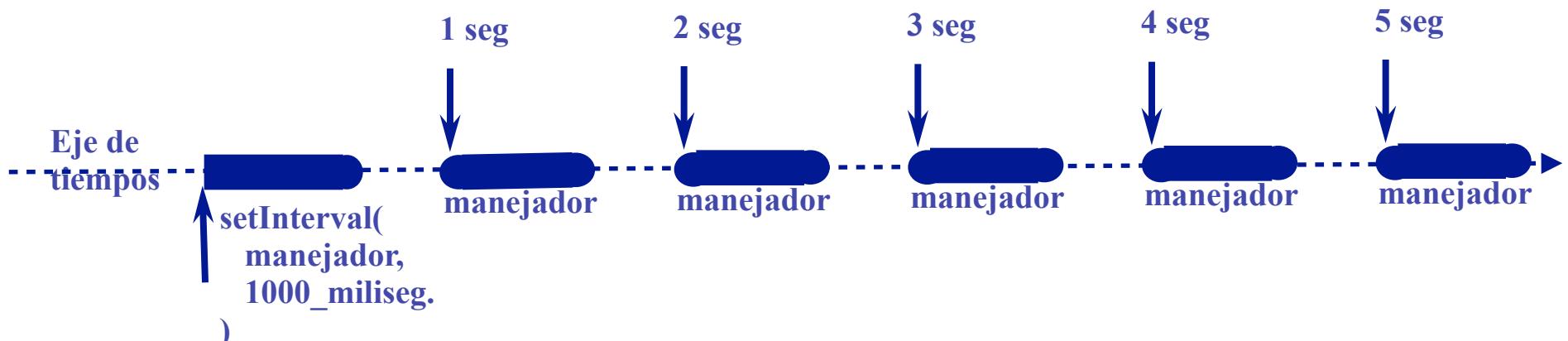
Eventos y Manejadores

- ◆ JavaScript utiliza eventos para interaccionar con el entorno
 - Hay eventos de muchos tipos
 - ◆ Temporizadores, clicks en botón, tocar en pantalla, pulsar tecla, ...
- ◆ Manejador (callback) de evento
 - función que se ejecuta al ocurrir el evento
- ◆ El script inicial debe configurar los manejadores (callbacks)
 - a ejecutar cuando ocurra cada evento que deba ser atendido



Eventos periódicos con setInterval(....)

- ◆ JavaScript tiene una función **setInterval(..)**
 - para programar eventos periódicos
- ◆ **setInterval (manejador, periodo_en_milisegundos)**
 - tiene 2 parámetros
 - ◆ **manejador**: función que se ejecuta al ocurrir el evento
 - ◆ **periodo_en_milisegundos**: tiempo entre eventos periódicos



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

<script type="text/javascript">

function mostrar_fecha() {
    var cl = document.getElementById("fecha");
    cl.innerHTML = new Date();
}
</script>

</head>

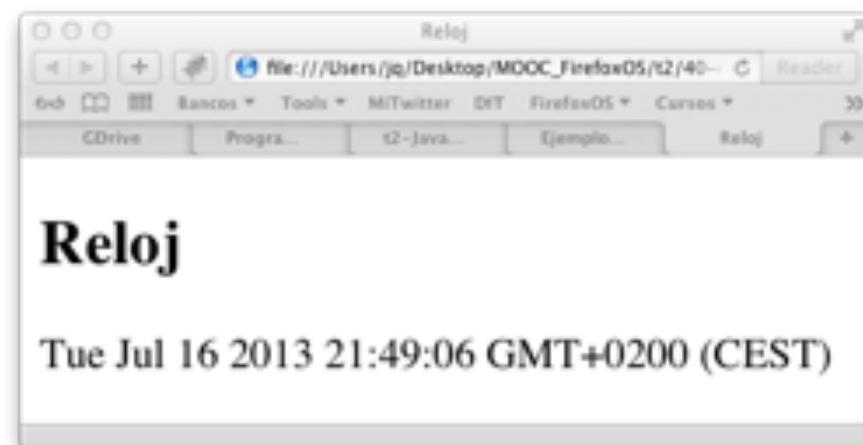
<body>
<h2>Reloj</h2>

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

<script type="text/javascript">
    mostrar_fecha(); // muestra fecha al cargar
                    // actualiza cada segundo
    setInterval(mostrar_fecha, 1000);
</script>
</body>
</html>
```

Reloj

- ◆ Utilizamos la función
 - **setInterval(manejador, T)**
 - ◆ para crear un reloj
- ◆ Cada segundo se muestra
 - El valor de reloj del sistema



Eventos DOM

◆ Los eventos DOM se asocian a elementos HTML

- como atributos: 'onclick', 'ondblclick', 'onload',
 - ◆ donde el manejador es el valor asignado al atributo

◆ Ejemplo:

-
 - ◆ Código del manejador: "**this.src='img2.png'"** (valor del atributo)
 - **this** referencia el objeto DOM asociado al manejador

◆ Tutorial:

- https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Event_attributes

Eventos en HTML

- ◆ Definimos **2 manejadores** de evento en elem. ``
 - Atributo **onclick**: muestra el icono enfadado
 - Atributo **ondblclick**: muestra el icono pasivo
- ◆ **this.src** referencia atributo **src** de ``
 - **this** referencia objeto DOM asociado: ``

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

<h4> Evento HTML </h4>



</body>
</html>
```

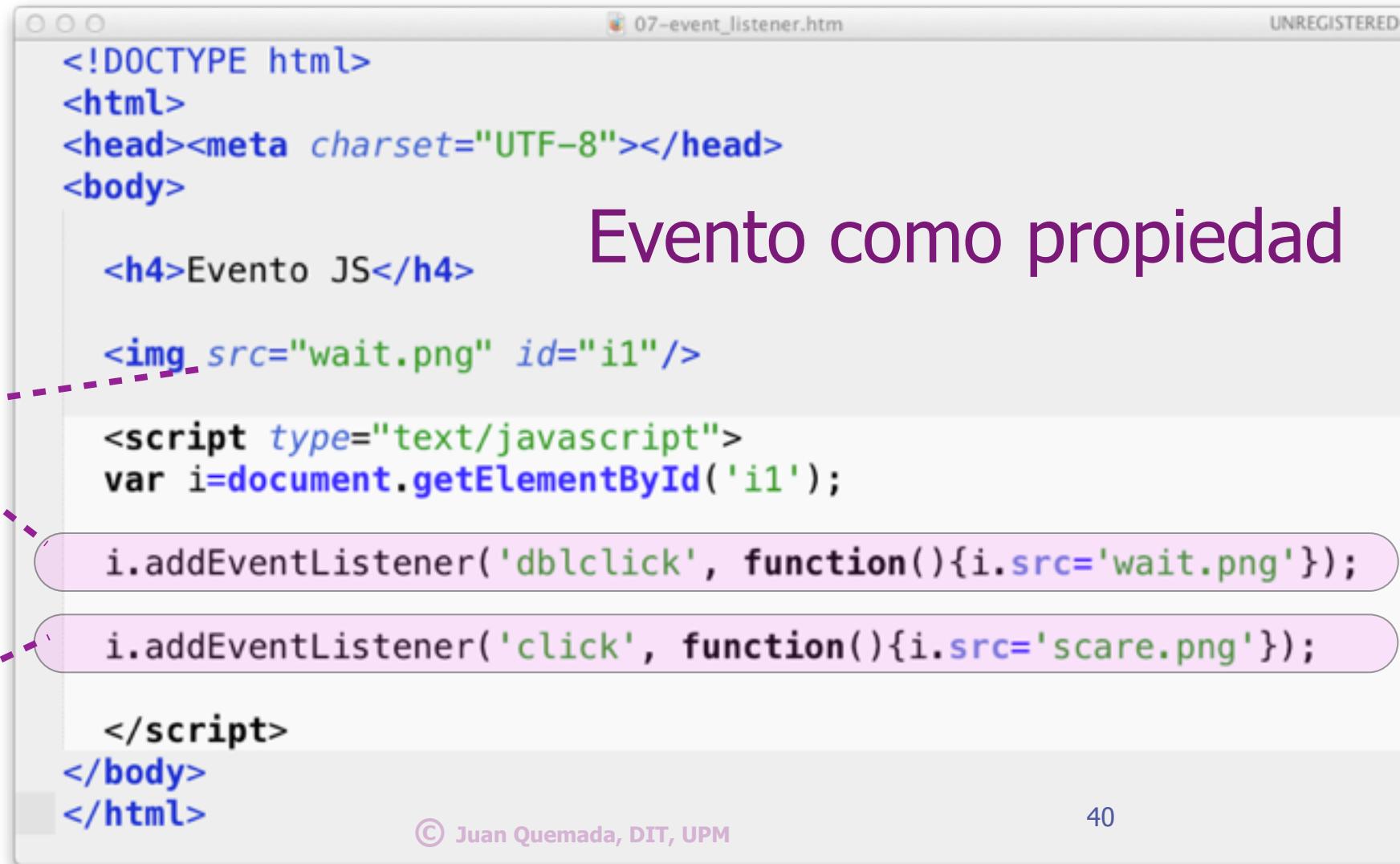
© Juan Quemada, DIT, UPM

Eventos definidos directamente en Javascript

- ◆ Los manejadores de eventos se pueden definir con
 - **objeto.addEventListener(evento, manejador)**
- ◆ También se pueden definir como propiedades
 - **objeto.evento = manejador**
 - ◆ objeto: objeto DOM al que se asocia el evento
 - ◆ evento: nombre (onload, onclick, onmouseover, etc.)
 - ◆ manejador: función ejecutada al ocurrir un evento
- ◆ Ejemplos
 - `img.addEventListener("onclick", function() { ... código ...})`
 - `img.onclick=function() { ... código...}`

- ◆ Los manejadores de evento se definen ahora en un script separado
 - El objeto `` se identifica desde JavaScript con `getElementById(..)`
 - manejador se añade con método: `object.addEventListener(event, manejador)`
 - Actualmente se recomienda usar siempre `addEventListener(....)`

Evento como propiedad



```

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

<h4>Evento JS</h4>



<script type="text/javascript">
var i=document.getElementById('i1');

i.addEventListener('dblclick', function(){i.src='wait.png'});
i.addEventListener('click', function(){i.src='scare.png'});

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

```

- ◆ Los manejadores de evento se definen también así en un script separado
 - El objeto `` se identifica desde JavaScript con `getElementById(..)`
 - Sintaxis de los manejadores: `object.event= manejador`

Evento como propiedad

```

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

<h4>Evento JS</h4>



<script type="text/javascript">

  var i=document.getElementById('i1');

    i.ondblclick = function(){ i.src='wait.png'; };

    i.onclick      = function(){ i.src='scare.png'; };

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

```

- ◆ El script pasa a la cabecera, se separa del documento HTML
 - El código se mete en la función **inicializar()**, que se ejecutara al ocurrir **onload**
 - **onload** ocurre con la página HTML ya cargada y el objeto DOM construido

Evento onload



Evento onload

08-event_onload.htm UNREGISTERED

```

<!DOCTYPE html>
<html>
<head><title>Evento onload</title><meta charset="UTF-8">

<script type="text/javascript">

    function inicializar() {
        var i=document.getElementById('i1');
        i.ondblclick = function () {i.src='wait.png'};
        i.onclick     = function () {i.src='scare.png'};
    }

</script>
</head>

<!-- El arbol DOM ya esta construido al ocurrir onload -->
<body onload="inicializar()">

    <h4>Evento onload</h4>

</body>
</html>

```

Evento onload



Botones y formularios en JavaScript

Entradas y botones

- ◆ **Entrada:** un cajetín en pantalla para introducir texto en una aplicación
 - Se define con `<input type="text" ...>`
 - ◆ el atributo `value="texto"` representa en texto dentro del cajetín
- ◆ **Botón:** elemento gráfico que invita a hacer clic
 - Se define con `<button type="button" ...>nombre</button>`

```
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8"></head>
<body>
  <h4> Input y Button </h4>
  <input type="text" value="responda aquí"/>
  <button type="button">consultar</button>
</body>
</html>
```



Ejemplo Pregunta

- ◆ Esta WebApp plantea la pregunta
 - ¿Quien descubrió América?
 - ◆ para ilustrar como interaccionar
 - a través de formularios y botones
- ◆ Escribir la respuesta en el cajetín
 - y pulsar el botón “**consultar**”
 - ◆ para saber si es correcto
- ◆ Según sea la respuesta se responde
 - “**Correcto**” o “**No es correcto**”

Pregunta

¿Quien descubrió América?

responda

consultar

Pregunta

¿Quien descubrió América?

Cristobal Peréz

consultar

No es correcto

Pregunta

¿Quien descubrió América?

Cristobal Colón

consultar

Correcto

Pregunta

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

function res() {
    var respuesta = document.getElementById('respuesta');
    var resultado = document.getElementById('resultado');

    if (respuesta.value === "Cristobal Colón")
        resultado.innerHTML = "Correcto";
    else resultado.innerHTML = "No es correcto";
}

</script>
</head>
<body>
    <h4> Pregunta </h4>
    <p> ¿Quien descubrió América? </p>

    <input type="text" id="respuesta" value="responda aquí">
    <button type="button" onclick="res()">consultar</button>
    <p><div id="resultado" /></p>

</body>
</html>
```

