



Tema 7.1

Javascript: Funciones

Función

```
function comer(persona, comida) {  
    return (persona + " come " + comida);  
};
```

```
comer('José', 'paella');
```

=> 'José come paella'

◆ Función:

- bloque de código invocable a través del nombre
 - ◆ Con parámetros

◆ Sentencia “**return expr**”

- finaliza y devuelve el valor de la expresión **expr**

◆ La función devuelve **undefined**

- si alcanza el final del bloque de código sin haber ejecutado ningún **return**

Parámetros de una función

- ◆ La función se puede invocar con un
 - **número variable de parámetros**
- ◆ Un **parámetro inexistente** está **undefined**

```
function comer(persona, comida) {  
    return (persona + " come " + comida);  
};
```

```
comer('José', 'paella');           => 'José come paella'
```

```
comer('José', 'paella', 'carne');   => 'José come paella'  
comer('José');                      => 'José come undefined'
```

El array de argumentos

- ◆ Los parámetros de la función están accesibles también a través del
 - array de argumentos: **arguments[....]**
 - ◆ Cada parámetro es un elemento del array
- ◆ En: **comer('José', 'paella')**
 - **arguments[0]** => 'José'
 - **arguments[1]** => 'paella'

```
function comer() {  
    return (arguments[0] + " come " + arguments[1]);  
};
```

```
comer('José', 'paella');    => 'José come paella'
```

```
comer('José', 'paella', 'carne');    => 'José come paella'  
comer('José');                    => 'José come undefined'
```

Parámetros por defecto

- ◆ Funciones invocadas con un número variable de parámetros
 - Pueden definir parámetros por defecto con el operador ||
 - ◆ "x || <parámetro_por_defecto>"
- ◆ Si x es "undefined"
 - se evaluará a false y || devolverá: **parámetro por defecto**

```
var comer = function(persona, comida) {  
    var persona = (persona || 'Nadie');  
    var comida = (comida || 'nada');  
    return (persona + " come " + comida);  
};
```

```
comer('José');    => 'José come nada'  
comer();          => 'Nadie come nada'
```

Varios scripts

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

```
<!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>
```



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

```
<!DOCTYPE html><html>
<head>
<meta charset="UTF-8">
</head>
<body>
<h3>Parámetros</h3>
<pre>
<script
  type="text/javascript">
```

```
function concatenar() {
  var i, resultado = "";
  for (i=0; i < arguments.length; ++i) {
    resultado += " " + arguments[i];
  }
  return resultado;
};
```

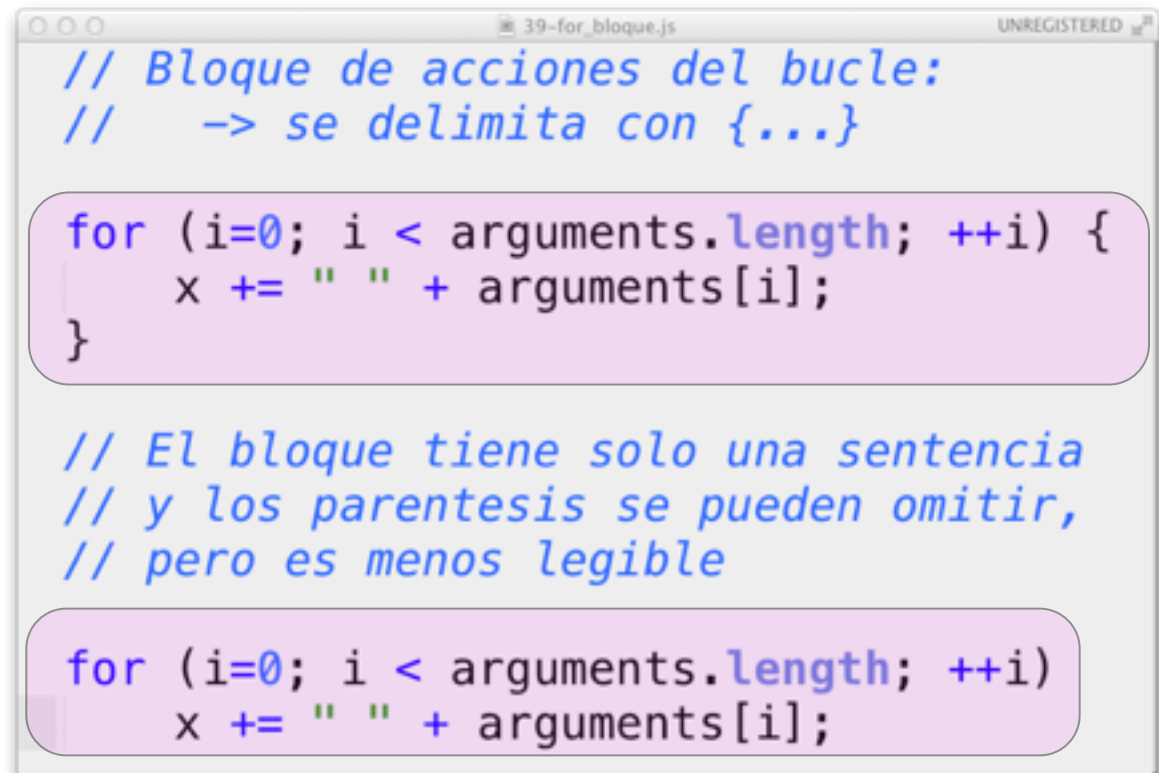
```
document.write("concatenar('Hola,', 'que', 'tal!')\n");
document.write("=> " + concatenar('Hola,', 'que', 'tal!'));
</script>
</pre>
</body>
</html>
```



concatenar(...)

Sintaxis de la sentencia for

- ◆ Comienza por **for**
- ◆ seguido de la condición
 - **(i=0; i < arguments[i]; i++)**
- ◆ La condición tiene 3 partes
 - **Inicialización:** i=0
 - **Permanencia en bucle:** i < arguments.length
 - **Acción final bloque:** ++i
- ◆ Los bloques de más de 1 sentencia deben delimitarse con {...}
- ◆ Bloques de 1 sentencia
 - pueden omitir {..}, pero se mejora la legibilidad delimitandolos con {..}



```
// Bloque de acciones del bucle:  
// -> se delimita con {...}  
  
for (i=0; i < arguments.length; ++i) {  
    x += " " + arguments[i];  
}  
  
// El bloque tiene solo una sentencia  
// y los parentesis se pueden omitir,  
// pero es menos legible  
  
for (i=0; i < arguments.length; ++i)  
    x += " " + arguments[i];
```


Ejercicio

◆ Si tenemos las siguientes definiciones de funciones

```
function f_1 (x) {   return (x);       }
```

```
function f_2 (x) {   return (x + "");  }
```

```
function f_3 (x) {   return (!x);      }
```

◆ Como se evaluarán las siguientes 2 expresiones

```
f_1 (7)           =>  7, "7", undefined, "undefined", false, "false", true, "true"
```

```
f_1 (undefined)   =>  7, "7", undefined, "undefined", false, "false", true, "true"
```

```
f_2 (7)           =>  7, "7", undefined, "undefined", false, "false", true, "true"
```

```
f_2 (undefined)   =>  7, "7", undefined, "undefined", false, "false", true, "true"
```

```
f_3 (7)           =>  7, "7", undefined, "undefined", false, "false", true, "true"
```

```
f_3 (undefined)   =>  7, "7", undefined, "undefined", false, "false", true, "true"
```



Tema 7.2

Javascript: Funciones como objetos

Funciones como objetos

- ◆ Las funciones son **objetos** de pleno derecho
 - pueden asignarse a **variables, propiedades, parámetros,**
- ◆ “**function literal**”: es una función que se define sin nombre
 - Se suele asignar a una variable, que le da su nombre
 - ◆ Se puede invocar a través del nombre de la variable

```
var comer = function(persona, comida) {  
    return (persona + " come " + comida);  
};
```

```
comer('José','paella');           => 'José come paella'
```

Operador de invocación de una función

- ◆ El objeto función puede asignarse o utilizarse como un valor
- ◆ el operador (...) invoca la función a la que se aplica
 - Solo es aplicable a funciones

```
var comer = function(persona, comida) {  
    return (persona + " come " + comida);  
};
```

```
var x = comer;           // asigna a x el código de la función  
x('José','paella'); => 'José come paella'
```

```
var y = comer();        // asigna a y el resultado de invocar la función  
y                       => 'undefined come undefined'
```

Métodos de objetos

- ◆ Los métodos son funciones asignadas a propiedades de un objeto
 - acceden al objeto referenciándolo con **this**
 - ◆ **this** es opcional y puede omitirse: **this.titulo** es equivalente a **titulo**
- ◆ Un método se invoca en un objeto
 - con la notación punto: **obj.método(..)**

```
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"
```

. []	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.

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

- ◆ Si tenemos el objeto pelicula defino asi

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

- ◆ Como se evaluarán las siguientes 2 expresiones

typeof pelicula	=>	"null", "undefined", "string", "object", "function"
typeof pelicula.titulo	=>	"null", "undefined", "string", "object", "function"
typeof pelicula.resumen	=>	"null", "undefined", "string", "object", "function"
typeof pelicula.resumen()	=>	"null", "undefined", "string", "object", "function"

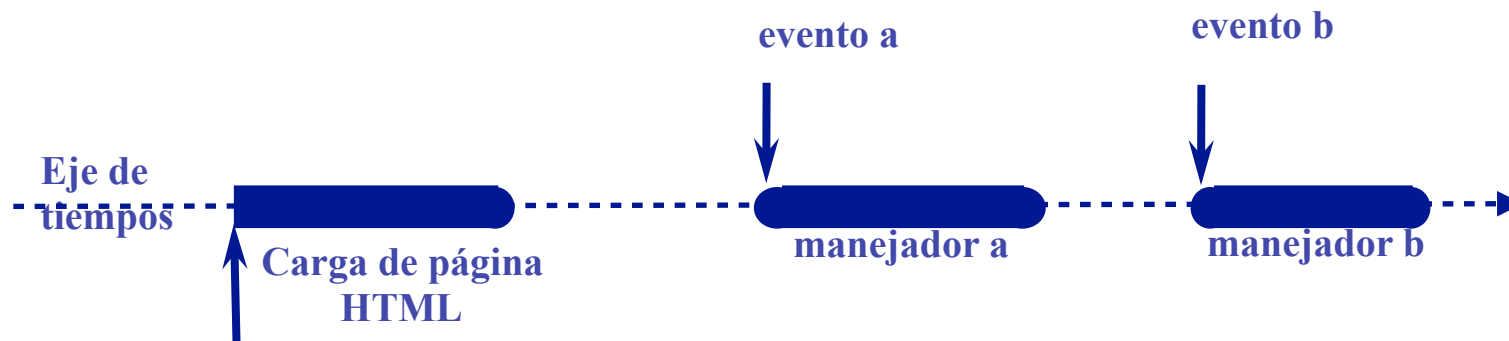


Tema 7.3

Javascript: Eventos

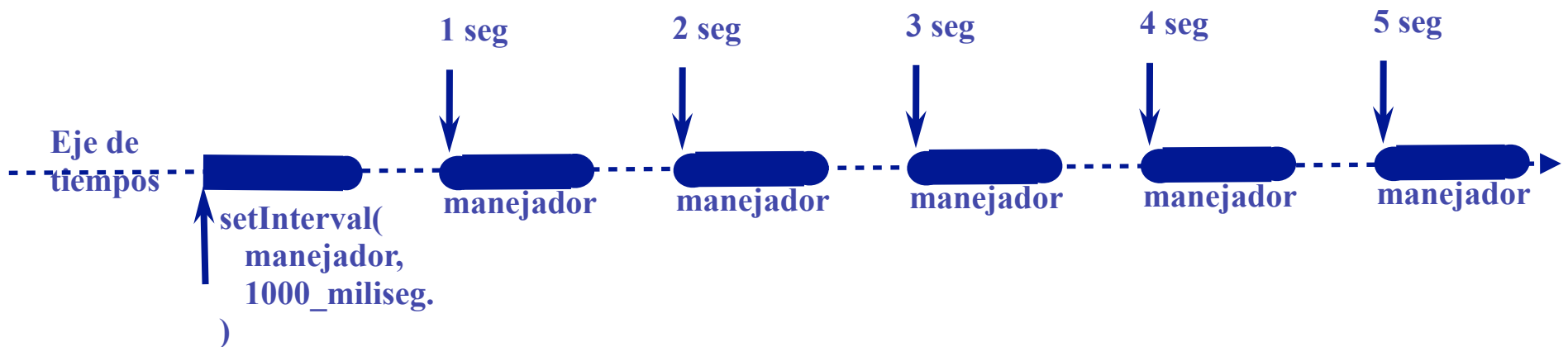
Eventos y Manejadores

- ◆ JavaScript utiliza eventos para interaccionar con el entorno
 - Hay eventos de muchos tipos
 - ◆ Temporizadores, clicks en boton, 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



```
35-clock.htm UNREGISTERED
<!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>

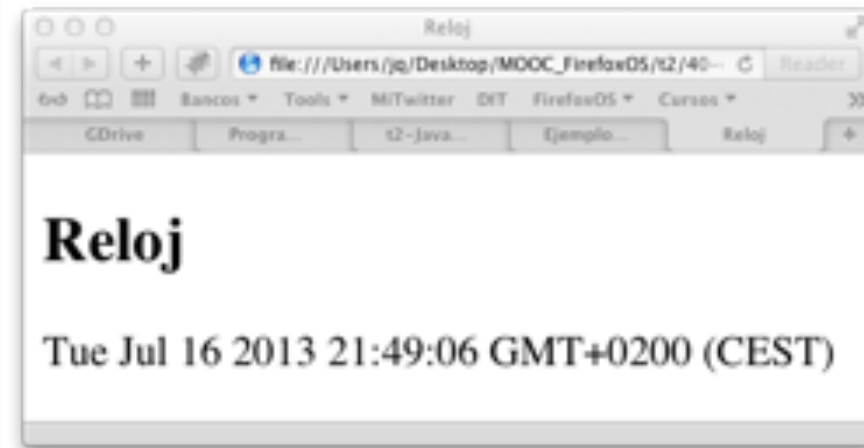
<body>
<h2>Reloj</h2>

<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

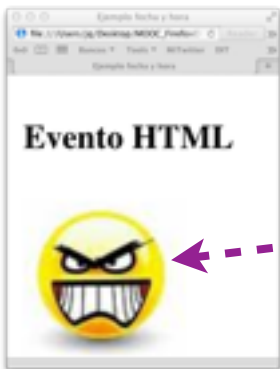
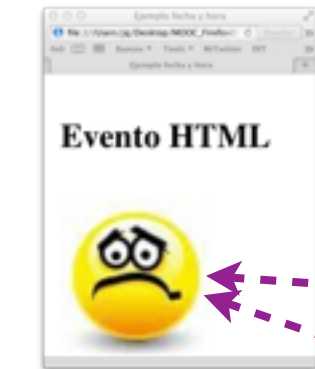
◆ Ejemplo:

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

◆ Tutorial: http://www.w3schools.com/tags/ref_eventattributes.asp

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: ****



```
05-event.htm UNREGISTERED
<!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 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
- ◆ Los eventos también se pueden definir con
 - **objeto.addEventListener(evento, manejador)**
- ◆ Ejemplos
 - `img.onclick=function() {... código...}`
 - `img.addEventListener("onclick", function() {... código ...})`

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

Evento como propiedad

```
06-event_id.htm UNREGISTERED
<!DOCTYPE html>
<html>
<head><meta charset="UTF-8"></head>
<body>

  <h4>Evento JS</h4>

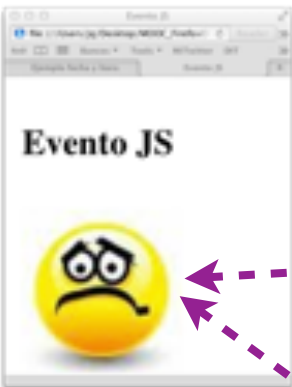
  <script type="text/javascript">

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

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

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

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



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

Evento como propiedad

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

  <h4>Evento JS</h4>

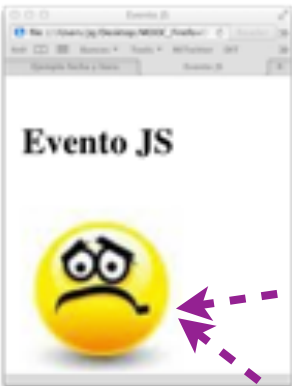
  

  <script type="text/javascript">
    var i=document.getElementById('i1');

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

    i.addEventListener('click', 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 ejecuta al ocurrir **onload**
 - ◆ **onload** ocurre con la página HTML ya cargada y el objeto DOM construido



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

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

```
function inicializar() {
    var i=document.getElementById('i1');
    i.ondbclick = 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

Ejercicio

- ◆ Modificar el reloj del ejemplo anterior para que presente
 - Horas, minutos, segundos y milisegundos con el formato
 - ♦ 12 horas, 23 minutos, 10 segundos, 123 ms
 - Se recomienda variar el intervalo del evento a 1 ms
 - Utilizar métodos que extraen de Date(): horas, minutos, segundos y ms
 - ♦ Tutorial Date: http://www.w3schools.com/jsref/jsref_obj_date.asp
 - Modificar el formato de presentación en **mostrar_hora()**
- ◆ Hacer que el reloj pare al hacer click sobre él (si está en marcha)
 - y que vuelva a arrancar al hacer click sobre él cuando esté parado
 - ♦ utilizar **addEventListener(..)** para añadir manejadores
 - Tutorial timers: http://www.w3schools.com/js/js_timing.asp



Tema 7.4

Javascript: Formularios

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 cajetin
- ◆ **Botón:** elemento gráfico que invita a hacer clic
 - Se define con `<buton type=button ...>nombre</button>`

The image shows two windows side-by-side. The left window is an HTML editor titled '50-button_input.htm' showing the following code:

```
<!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>
```

The right window is a web browser titled 'Input y Button' showing the rendered output. It has a title bar with 'Input y Button' and a toolbar with navigation buttons. The address bar shows 'file:///Users/jq/Desktop/MOC'. The main content area displays the text 'Input y Button' followed by a text input field containing 'responda aquí' and a button labeled 'consultar'. Dashed purple arrows point from the code snippets in the left window to the corresponding UI elements in the right window: one from the `<input type="text" value="responda aquí"/>` line to the text input field, and another from the `<button type="button">consultar</button>` line to the 'consultar' button.

© Juan Quemada, DIT, UPM

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 boton “**consultar**”
 - ◆ para saber si es correcto
- ◆ Según sea la respuesta se responde
 - “**Correcto**” o “**No es correcto**”

A browser window titled 'Pregunta' showing the question '¿Quien descubrió América?' and an empty input field labeled 'responda' with a 'consultar' button next to it.

Two screenshots showing the application's response to user input. The top screenshot shows the input 'Cristobal Pérez' and the message 'No es correcto'. The bottom screenshot shows the input 'Cristobal Colón' and the message 'Correcto'. A dashed purple arrow points from the 'consultar' button in the first screenshot to the top screenshot.

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');

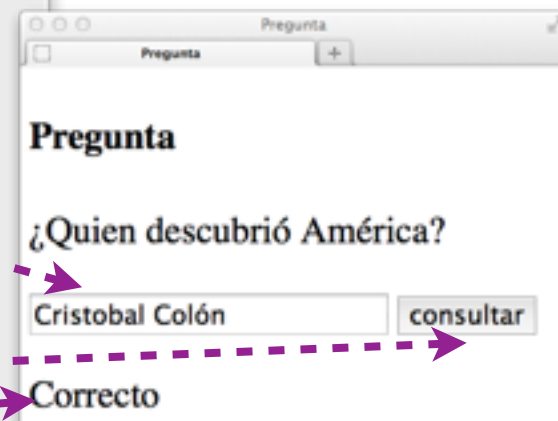
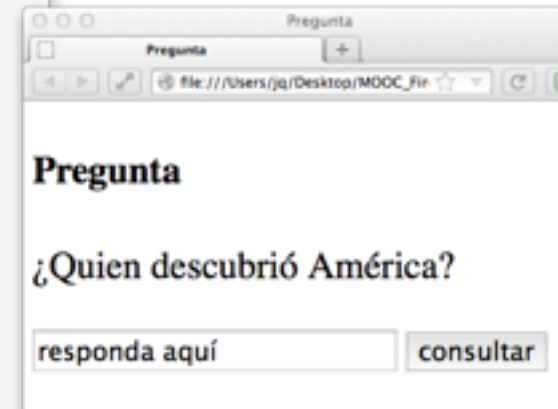
  if (respuesta.value === "Cristobal Colón")
    resultado.innerHTML = "Correcto";
  else resultado.innerHTML = "No es correcto";
}

</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>
```



Ejercicio

- ◆ Si tenemos una página HTML con el siguiente contenido

```
<!DOCTYPE html>
<html> ... <body>

  <h4 id="id1" >Título</h4>

  <input type="text" id="id2" name= "caja" value="7">

  <script type="text/javascript">
    ..... // script con expresiones de abajo
  </script>
</body>
</html>
```

- ◆ Como se evaluarán las siguientes expresiones si estuviesen en el script

document.getElementById("id1").innerHTML	=> undefined, "", "null", "Título", 7, "7"
document.getElementById("id1").value	=> undefined, "", "null", "Título", 7, "7"
document.getElementById("id2").innerHTML	=> undefined, "", "null", "Título", 7, "7"
document.getElementById("id2").value	=> undefined, "", "null", "Título", 7, "7"

Ejercicio

- ◆ Modificar el ejemplo anterior para que si la respuesta es incorrecta
 - además de indicar “No es correcto”
 - ♦ inicialice el cajetin con el texto “pruebe otra vez”
 - Mas información sobre atributos de `<input type="text" ..>` en:
 - ♦ http://www.w3schools.com/jsref/dom_obj_text.asp