



# Funciones Javascript

# 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**

```
function comer (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
```

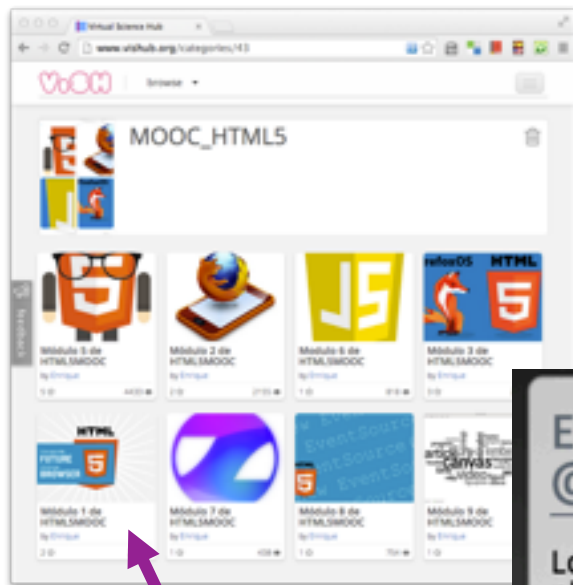
```
</script>
```

```
</body>
```

```
</html>
```



- ◆ función mostrar\_fecha()
  - Se define e invoca en scripts diferentes



<http://vishub.org/categories/43>

Seleccionar  
ejemplo

Hacer clic para  
actualizar pag.

Hacer clic  
en el  
Modulo 1

# Editor Interactivo

Editor interactivo de los ejemplos JavaScript del módulo 7 de @HTML5MOOC

Los ejemplos se pueden cambiar y ejecutar (visualizar) con los cambios introducidos pulsando el botón play, que está justo encima.

comer date concatenar clock event event\_id listener onload button input question

**Ejemplo:**

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

<body> <h2>Parámetros de función</h2>

<script type="text/javascript">

function comer(persona, comida)
{return (persona + " come " + arguments[2] + "<br>");};

document.write(comer('José','paella'));
document.write(comer('José','paella','carne'));
document.write(comer('José'));
</script>
</body>
</html>
```

**Resultado:**

**Parámetros de función**

José come undefined  
José come carne  
José come undefined

© Juan Quemada, DIT, UPM



# Bucles: sentencias while, for y do/while de JavaScript



# Bucle



- ◆ Un bucle es una **secuencia o bloque de instrucciones**
  - que **se repite** mientras se cumple una **condición** de permanencia
- ◆ Un bucle se controla con 3 elementos,
  - normalmente asociados a una variable(s) de control del bucle
    - ◆ **Inicialización:** fija los valores de arranque del bucle
    - ◆ **Permanencia en bucle:** indica si se debe volver a ejecutar el bloque
    - ◆ **Acciones de final bloque:** actualiza en cada repetición las variables de control
- ◆ Ilustraremos los bucles (**while**, **for** y **do/while**) con la función concatenar
- ◆ concatenar() acepta un número variable de parámetros
  - Transforma cada parámetro en el string equivalente
    - ◆ y los concatena incrementalmente utilizando un bucle

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

```
    // concatenar(..) recorre la matriz de
    // argumentos utilizando un bucle while
function concatenar() {
    var i=0, resultado = "";
    while (i < arguments.length) {
        resultado += " " + arguments[i];
        ++i;
    }
    return resultado;
};
```

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

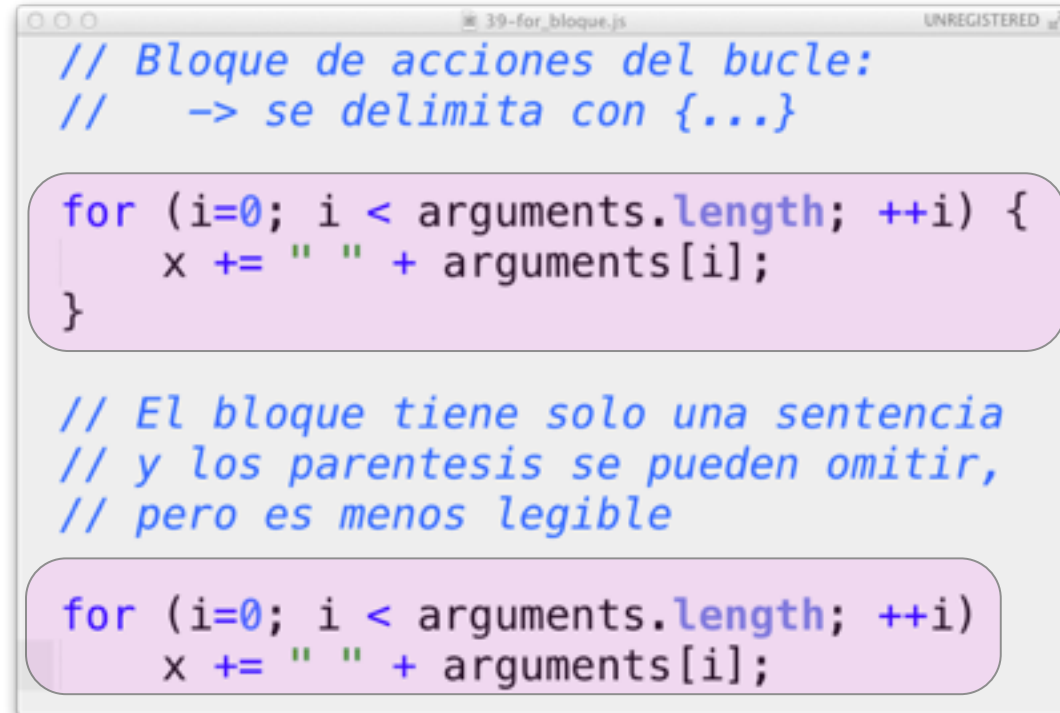
## Parámetros

```
concatenar('Hola,', 'que', 'tal!')
           =>  Hola, que tal!
```

concatenar(..):  
bucle while

# Sintaxis de la sentencia for

- ◆ La sentencia comienza conr **for**
- ◆ sigue la condición (con 3 partes)
  - **(i=0; i < arguments[i]; i++)**
    - ◆ **Inicialización:** i=0, ....
    - ◆ **Permanencia en bucle:** i < arguments.length
    - ◆ **Acción final bloque:** ++i, ...
- ◆ La sentencia termina con un bloque que debe delimitarse con {...}
- ◆ Bloques que contengan solo 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];
```

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

```
    // concatenar(..) recorre la matriz de
    // argumentos utilizando un bucle for
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>
```

## Parámetros

```
concatenar('Hola,', 'que', 'tal!')
=> Hola, que tal!
```

concatenar(..):  
bucle for

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

```
  // concatenar(..) recorre la matriz de
  // argumentos utilizando un bucle do/while
function concatenar() {
  var i=0, resultado = "";
  do {
    resultado += " " + arguments[i];
    ++i;
  } while (i < arguments.length)
  return resultado;
};
```

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

## Parámetros

```
concatenar('Hola,', 'que', 'tal!')
      =>  Hola, que tal!
```

concatenar(..):  
bucle do/while



# Funciones JavaScript 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"
```

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

## Operadores JavaScript

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

## STATEMENT SINTAXIS

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

## 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

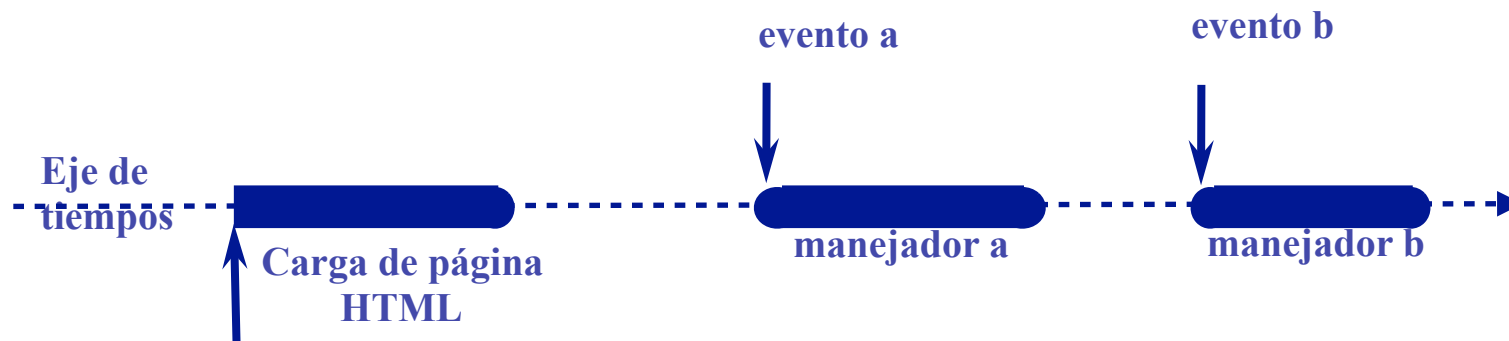
Extender cadena de ámbito (no recomendado)



# Eventos Javascript

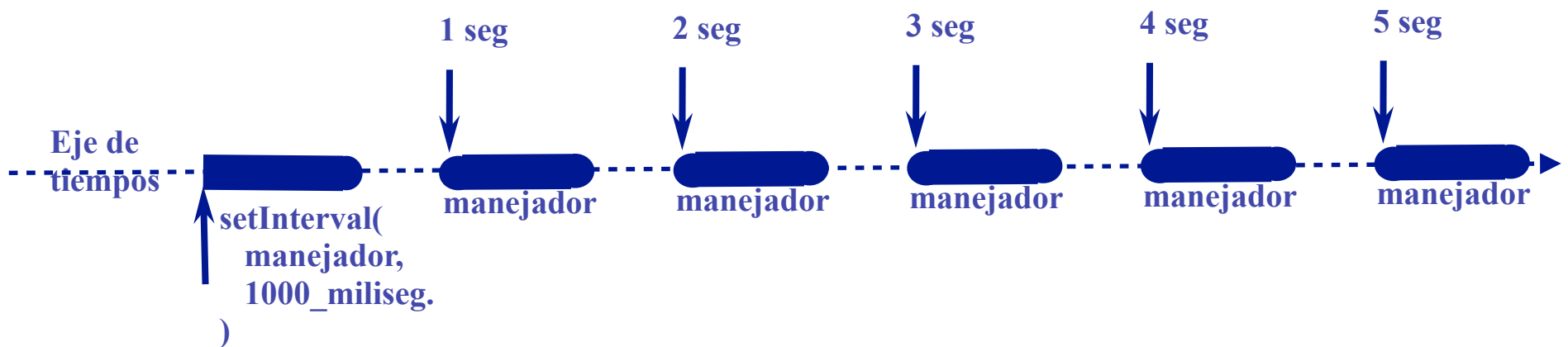
# Eventos y Manejadores

- ◆ 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>

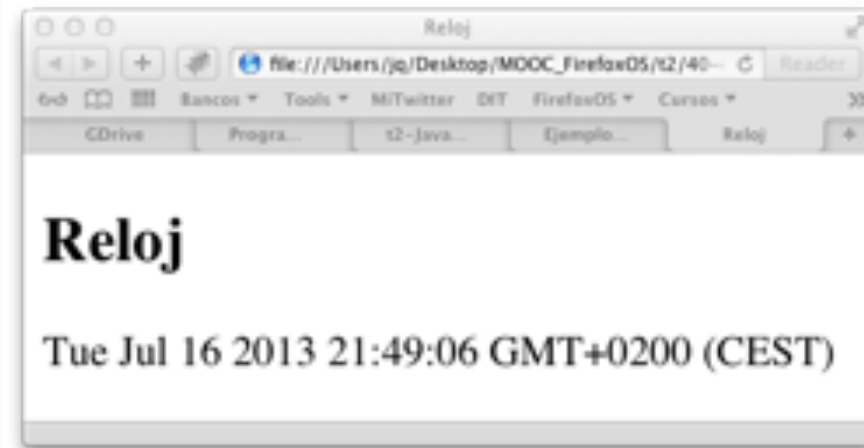
<body>
<h2>Reloj</h2>

<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](http://www.w3schools.com/tags/ref_eventattributes.asp)



# Eventos en HTML

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

```
<!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 `<img ...>` 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>

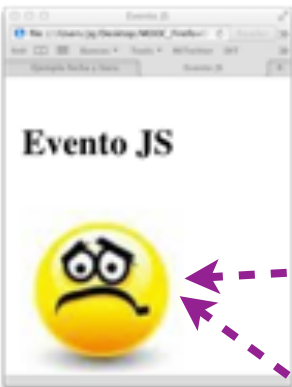
  <script type="text/javascript">

    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 **<img ...>** 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>

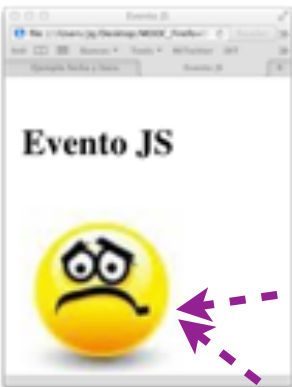
  

  <script type="text/javascript">
    var i=document.getElementById('i1');

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

The image shows a side-by-side comparison of HTML code and its rendered output. On the left, a code editor window titled '50-button\_input.htm' displays the following HTML 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 code snippets for the input and button are highlighted with purple rounded rectangles. On the right, a web browser window titled 'Input y Button' shows the rendered page. It features a heading 'Input y Button', a text input field containing 'responda aquí', and a button labeled 'consultar'. Dashed purple arrows point from the highlighted code snippets to their corresponding elements in the browser window: one from the input tag to the text field and another from the button tag to the 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 initial state of the web application. The address bar shows a file path. The main content area displays the title 'Pregunta' and the question '¿Quien descubrió América?'. Below the question is an empty text input field labeled 'respuesta' and a button labeled 'consultar'.

Two screenshots showing the application state after user interaction. A dashed purple arrow points from the 'consultar' button in the top screenshot to the first of these two screenshots. The first screenshot shows the input field filled with 'Cristobal Colón' and the button 'consultar'. Below the input field, the text 'Correcto' is displayed. The second screenshot shows the input field filled with 'Cristobal Pérez' and the button 'consultar'. Below the input field, the text 'No es correcto' is displayed.



# 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');

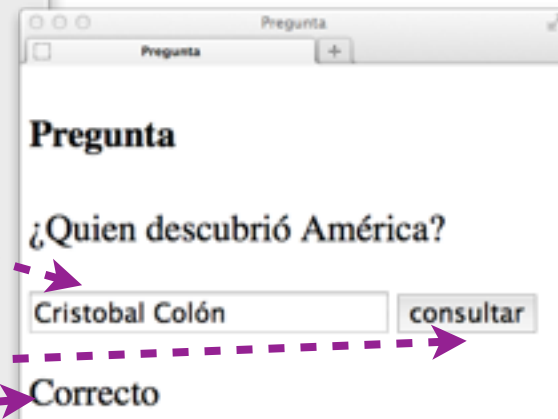
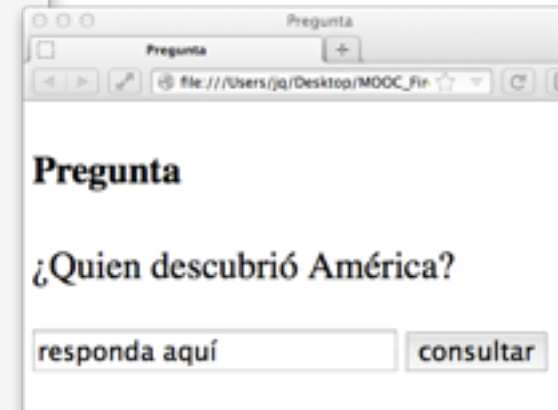
  if (respuesta.value === "Cristobal Colón")
    resultado.innerHTML = "Correcto";
  else resultado.innerHTML = "No es correcto";
}

</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>
```





# Librerías JavaScript jQuery y Zepto

# Librerías Javascript



- ◆ Las librerías JavaScript actuales son **multi-navegador**
  - Funcionan en IE, Firefox, Safari, Chrome, Opera, ...
    - ◆ Ahorran mucho tiempo -> **utilizarlas siempre que existan**
  - Ejemplos: **jQuery**, **Zepto**, Prototype, Dojo, lungo.js, PhoneGap, ...
- ◆ **jQuery** es muy popular (<http://jquery.com/>)
  - Procesar DOM, gestionar eventos, animaciones y estilos CSS, Ajax, ..
- ◆ **Zepto**: subconjunto compatible de **jQuery** para móviles (<http://zeptojs.com>)
  - **zepto.min.js** ocupa solo **9,7Kbytes** (gzipped)
    - ◆ ¡OJO! Soporta browsers móviles actuales, pero no **Internet Explorer**
  - Añade **eventos táctiles** para móviles y tabletas
  - Es equivalente a **jQuery 2.0** aparecida recientemente

# Objetos y función jQuery (o Zepto)

- ◆ **Objeto jQuery**: representación más eficaz de un **objeto DOM**
  - se procesan en bloque (array) con métodos de jQuery como **html(...)**
- ◆ **Función jQuery**: **jQuery("<selector CSS>")** o **\$("<selector CSS>")**
  - devuelve el **array de objetos jQuery** que casan con **<selector CSS>**
    - ◆ Si no casa ninguno, devuelve null o undefined
  - **<selector CSS>** selecciona objetos DOM igual que en CSS

```
document.getElementById("fecha").innerHTML = "Hola";
```

// es equivalente a:

```
$("#fecha").html("Hola");
```

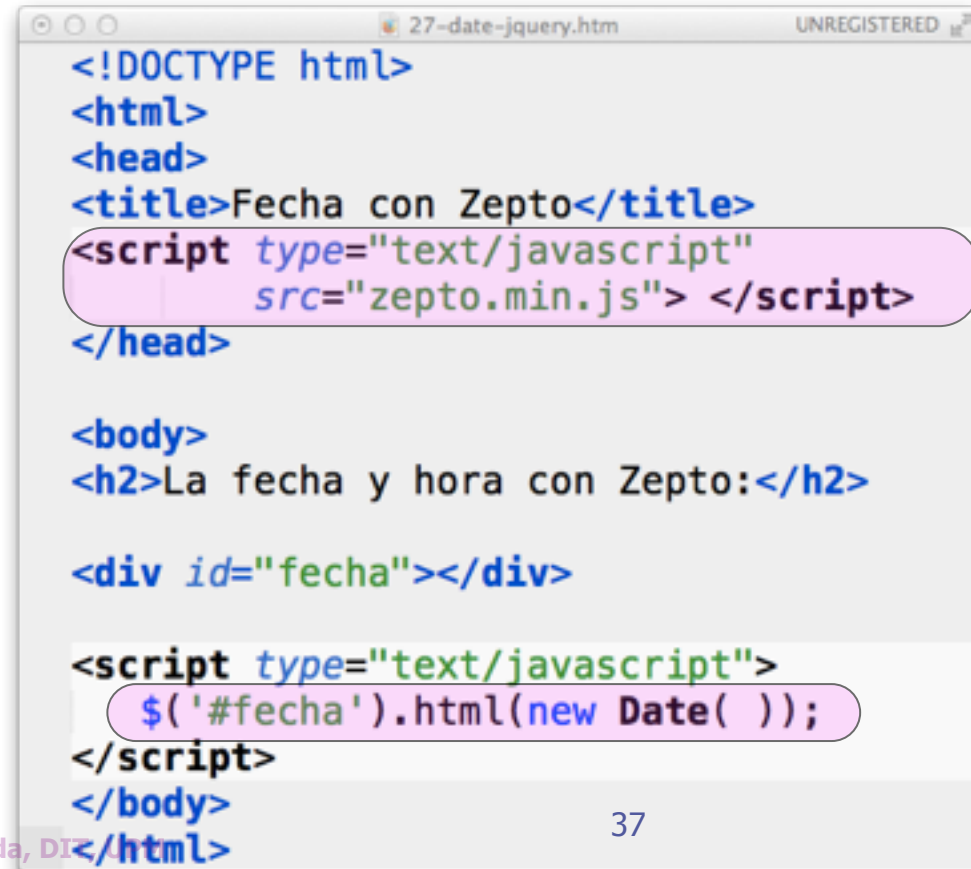
- ◆ La función jQuery convierte además **objetos DOM** y **HTML** a **objetos jQuery**

```
$(objetoDOM); // convierte objetoDOM a objeto jQuery
```

```
$("<ul><li>Uno</li><li>Dos</li></ul>") // convierte HTML a objeto jQuery
```

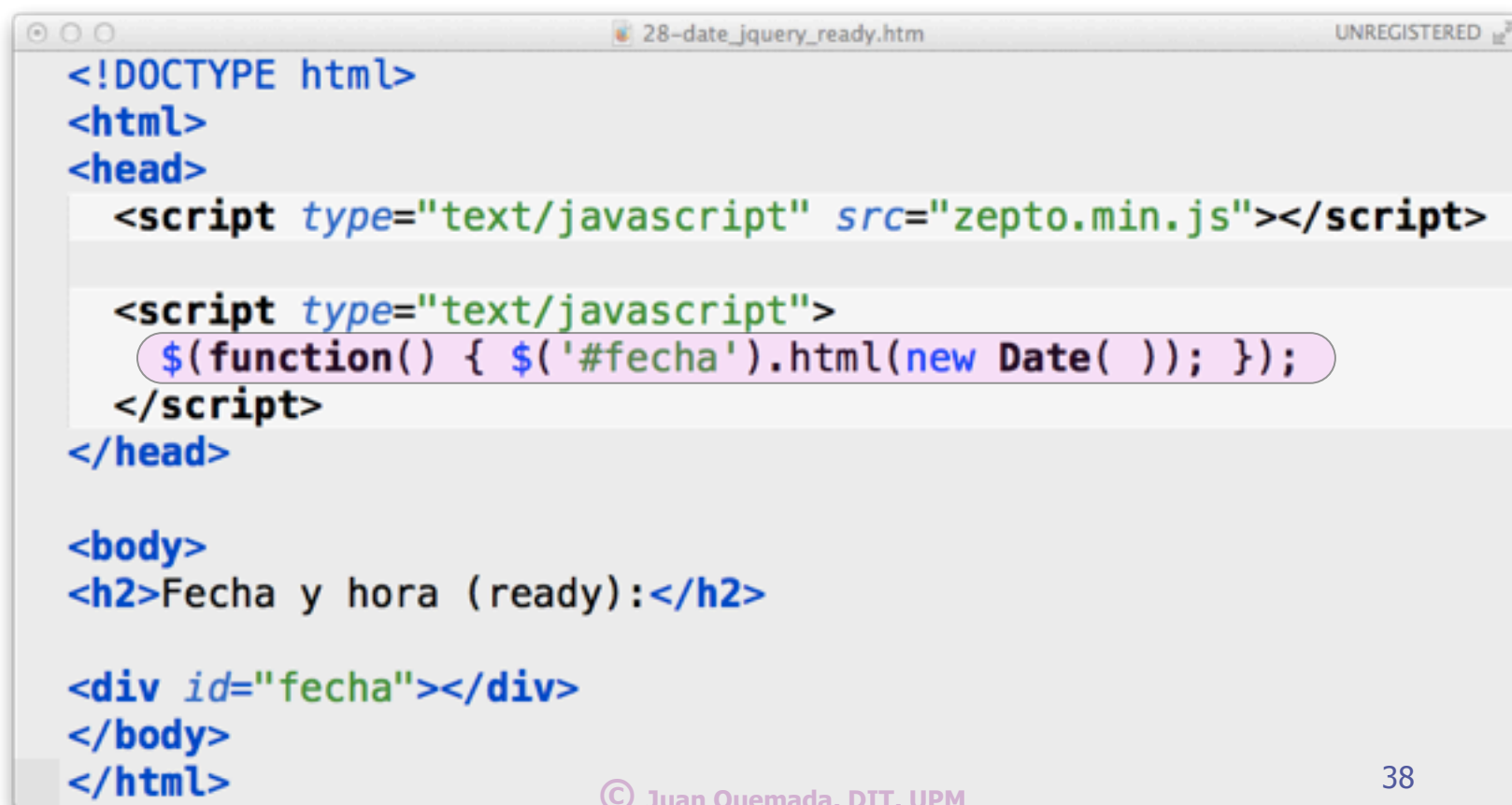
# Fecha y hora con jQuery/Zepto

- ◆ **Libreria:** script externo identificado por un **URL** que hay que cargar con:
  - `<script type="text/javascript" src="zepto.min.js" > </script>`
- ◆ `$("#fecha")` devuelve el objeto jQuery/Zepto del elemento con id="clock"
- ◆ `$("#fecha").html(new Date())` inserta
  - **new Date()** en objeto jQuery
    - ◆ devuelto por `$("#fecha")`



# Función ready: esperar a página cargada

- ◆ Función ready(): ejecuta un script con el objeto DOM está construido
  - Invocación: `$(document).ready(function() { .. código js ... });`
    - ◆ Suele utilizarse la sintaxis abreviada: `$(function() { .. código .. });`



The screenshot shows a web browser window titled "28-date\_jquery\_ready.htm" with a status bar indicating "UNREGISTERED". The browser's developer tools or source view displays the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
  <script type="text/javascript" src="zepto.min.js"></script>

  <script type="text/javascript">
    $(function() { $('#fecha').html(new Date( )); });
  </script>
</head>

<body>
<h2>Fecha y hora (ready):</h2>

<div id="fecha"></div>
</body>
</html>
```

The code demonstrates the use of the `$(function() { ... });` shorthand for the `ready()` event. The script loads `zepto.min.js` and then executes a function that updates the content of the `#fecha` element with the current date. The `$(function() { ... });` line is highlighted with a pink oval.

# Cache y CDN (Content Distribution Network)

- ◆ Cache: contiene recursos accedidos anteriormente para acceso rapido
  - Caches identifican recursos por igualdad de URLs
- ◆ CDN: utilizar URL común a Google, Microsoft, jQuery, Zepto, ...
  - Para maximizar probabilidad de que recursos estén ya en la cache



The screenshot shows a web browser window titled "29-date\_zepto\_cdn.htm" with a status bar indicating "UNREGISTERED". The browser's content area displays the following HTML code:

```
<html>
<head>
<script type="text/javascript" src="http://zeptojs.com/zepto.min.js">
</script>

<script type="text/javascript">
  $(function() { $('#clock').html(new Date( )); });
</script>
</head>
<body>
<h2>Fecha y hora, con CDN Zepto</h2>

<div id="clock"></div>
</body>
</html>
```



# Selectores tipo CSS de jQuery/Zepto

## SELECTORES DE MARCAS CON ATRIBUTO ID

`$("#h1#d83")` /\* devuelve objeto con marca **h1** e **id="d83"** \*/  
`$("#d83")` /\* devuelve objeto con **id="d83"** \*/

## SELECTORES DE MARCAS CON ATRIBUTO CLASS

`$("#h1.princ")` /\* devuelve array de objetos con marcas **h1** y **class="princ"** \*/  
`$(".princ")` /\* devuelve array de objetos con **class="princ"** \*/

## SELECTORES DE MARCAS CON ATRIBUTOS

`$("#h1[border]")` /\* devuelve array de objetos con marcas **h1** y **atributo border** \*/  
`$("#h1[border=yes]")` /\* devuelve array de objetos con marcas **h1** y **atributo border=yes** \*/

## SELECTORES DE MARCAS

`$("#h1, h2, p")` /\* devuelve array de objetos con marcas **h1, h2 y p** \*/  
`$("#h1 h2")` /\* devuelve array de objetos con marca **h2 después de h1** en el árbol \*/  
`$("#h1 > h2")` /\* devuelve array de objetos con marca **h2 justo después de h1** en árbol \*/  
`$("#h1 + p")` /\* devuelve array de objetos con **marca p** adyacente a **h1** del mismo nivel \*/

.....



# jQuery/Zepto: Metodos modificadores

## ◆ \$('#id3').html('Hello World!')

- sustituye **HTML** por **'Hello World!'** en el elemento con **id='id3'**
  - ◆ \$('#id3').html() devuelve contenido **HTML** de \$('#id3')
- \$('#id3').append('Hello World!') añade **HTML** al final del existente

## ◆ \$('.expr').val('3')

- inserta atributo **value='3'** a elementos de la clase **'expr'**
  - ◆ \$('#id3').val() devuelve atributo **value** de \$('#id3')

## ◆ \$('#id3').attr('rel', 'license')

- inserta atributo **rel='license'** a elemento con **id=id3**
  - ◆ \$('#id3').attr('rel') devuelve atributo **rel** de \$('#id3')

## ◆ \$('ul').addClass('visible')

- inserta atributo **class='visible'** a elementos **<ul>** (lista)

## ◆ \$('h1').hide() y \$('h1').show()

- oculta o muestra elementos **<h1>**

# 4 Modos de invocar Zepto (o jQuery)

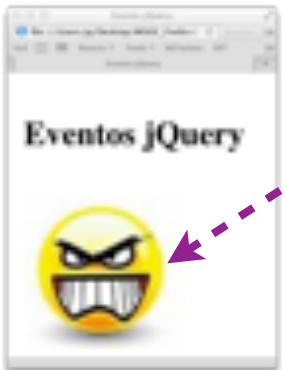
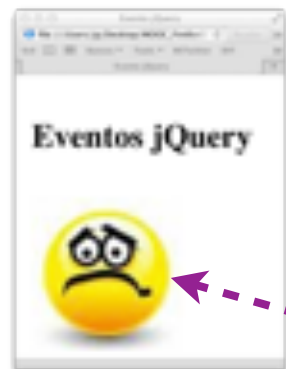
- ◆ **String con selector CSS:** `$("#<selector CSS>")`
  - Devuelve un array de objetos jQuery que casan con **<selector CSS>**
- ◆ **"String HTML":** `$("#<ul><li>Uno</li><li>Dos</li></ul>")`
  - Devuelve objeto jQuery equivalente a string interpretado como **HTML**
- ◆ **"Objeto DOM":** `$(document)`
  - Transforma objeto DOM en objeto jQuery equivalente
- ◆ **"Función ready":** `$(function(){..})`
  - Ejecuta la función cuando el **árbol DOM está construido**



# Eventos con jQuery y Zepto

# Eventos con jQuery/Zepto

- ◆ Manejadores de eventos: se definen sobre el objeto jQuery `i` de `<img.. id=i1>`
  - con la función `on` -> `i.on('evento', manejador)`



```
20_event_id_zepto.htm UNREGISTERED
<!DOCTYPE html>
<html><head><title>Evento jQuery</title><meta charset="UTF-8">
<script type="text/javascript" src="zepto.min.js" > </script>
<script type="text/javascript">
    $(function(){
        var i = $('#i1');

        i.on('dblclick', function(){i.attr('src', 'wait.png')});

        i.on('click', function(){i.attr('src', 'scare.png')});
    });
</script>
</head>
<body>
    <h4>Eventos jQuery</h4>

    
</body>
</html>
```

```
<!DOCTYPE html>
<html><head><title>Pregunta (jQuery)</title><meta charset="UTF-8">

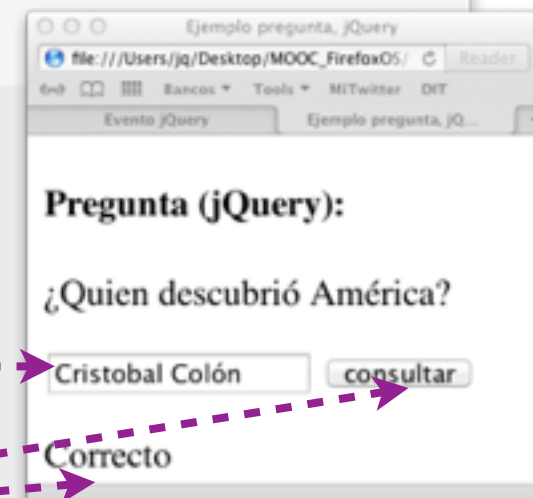
<script type="text/javascript" src="zepto.min.js" > </script>

<script type="text/javascript">
$(function(){
    $("#boton").on('click', function(){
        if ($('#respuesta').val() === "Cristobal Colón")
            $('#resultado').html("Correcto");
        else
            $('#resultado').html("No es correcto");
    });
});
</script>
</head>
<body>
<h4> Pregunta (jQuery):</h4>
¿Quien descubrió América? <p>

<input type="text" id="respuesta" value="responda aquí">
<button type="button" id="boton"> consultar </button>

<p><div id="resultado" />
</body>
</html>
```

## Pregunta Zepto





Final del tema  
Muchas gracias!

