



Ejemplo de un Cronómetro

Cronómetro

- ◆ WebApp similar a un cronómetro digital
- ◆ Cuenta décimas de segundo (100 miliseg.)
 - El contador se inicializa con **0,0** segundos
 - ◆ **n.toFixed(1)** formatea con 1 decimal
- ◆ Tiene 2 botones
 - **arrancar/parar**: arranca o para la cuenta
 - ◆ a partir del valor en que quedo
 - arranca si cronómetro parado
 - para si cronómetro contando
 - **inicializar**: pone el contador a 0,0
- ◆ La librería **zepto.js** utilizada en los temas siguientes **equivale a jQuery 2.0**
 - Los ejemplos pueden sustituir zepto por jQuery y funcionarán igual
 - ◆ <http://zeptojs.com/>



```

<!DOCTYPE html>
<html>
<head><title>Event Example</title><meta charset="UTF-8">
<script type="text/javascript" src="zepto.min.js" > </script>
<script type="text/javascript">
    $(function(){
        var t, cl = $("#crono");

        function mostrar() { cl.html((+cl.html() + 0.1).toFixed(1)); };
        function arrancar() { t=setInterval(mostrar, 100);};
        function parar() { clearInterval(t); t=undefined; };
        function cambiar() { if (!t) arrancar(); else parar(); };

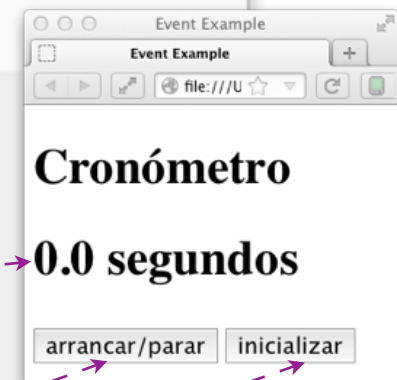
        $("#cambiar").on('click', cambiar);
        $("#inicializar").on('click', function(){ cl.html("0.0"); });
    });
</script>
</head>
<body>
<h2>Cronómetro</h2>

<h2><span id="crono"> 0.0 </span> segundos </h2>

<button type="button" id="cambiar"> arrancar/parar </button>
<button type="button" id="inicializar"> inicializar </button>
</body>
</html>

```

Cronómetro



DOM como almacén de datos

- ◆ El navegador guarda en **document** la página HTML que está mostrando
 - **document** es un objeto JavaScript con propiedades
 - ◆ que contienen todos los elementos de la página
- ◆ Las propiedades DOM son variables: **src**, **value**, **innerHTML**,
 - donde la información se puede guardar y recuperar
 - ◆ DOM solo contiene strings y todo debe convertirse a/de string
- ◆ Los elementos de DOM se pueden utilizar como variables
 - Hemos utilizado el elemento ****
 - ◆ para almacenar el contador de decimas de segundo



Memoria local en HTML5

Almacenamiento de datos en cliente

- ◆ HTML5 implementa nuevos tipos de almacenamiento de variables
 - Sencillas y eficientes de utilizar desde Javascript
 - ◆ Definición: <http://dev.w3.org/html5/webstorage/>
- ◆ **Variables locales**
 - los datos se guardan permanentemente, hasta que se borran
- ◆ **Variables de sesión**
 - Los datos solo se guardan solo **durante la sesión**
 - ◆ **Comienzo de sesión:** apertura de navegador o pestaña
 - ◆ **Final de sesión:** cierre de navegador o pestaña

Variables locales y de sesión

- ◆ Son **propiedades** de los **objetos** **localStorage** y **sessionStorage**
 - solo pueden contener **strings**, como por ejemplo
 - ◆ **localStorage.usuario** = “Pedro Pérez”;
 - ◆ **sessionStorage.apellido** = “Pérez”;
- ◆ Las variables locales están asociadas a **protocolo, dominio y puerto**
 - un programa solo puede acceder a propiedades de local/sessionStorage
 - ◆ creadas por otros programas cargados del mismo servidor
- ◆ **Same origin policy**
 - **Seguridad:** un programa solo confía en programas del mismo servidor
 - **Modularidad:** cada servidor tiene un espacio de nombres diferente

Ejemplo de localStorage

- ◆ Cada usuario que acceda a esta página tendrá una cuenta diferente
 - La variable está en su navegador

```
65-visitCount.html UNREGISTERED
<!DOCTYPE html>
<html><head><meta charset="UTF-8">
<script type="text/javascript" src="zepto.min.js"></script>
<script type="text/javascript">
  $(function() {
    // si variable no existe se crea (primera visita)
    localStorage.cuenta = (localStorage.cuenta || 0);

    localStorage.cuenta++; // incrementamos cuenta de visitas

    $('#cuenta').html(localStorage.cuenta);
  });
</script>
</head><body>
  <h3>Ejemplo de localStorage</h3>

  Ha visitado esta página <span id='cuenta'></span> veces!
</body>
</html>
```



Cronómetro con memoria

- ◆ Nueva versión del cronómetro con **localStorage**
 - así mantiene la cuenta de décimas de segundos
 - ◆ entre usos sucesivos de la aplicación
- ◆ El cronómetro utiliza ahora la variable
 - **localStorage.c**
 - ◆ para guardar la cuenta de segundos
- ◆ Debemos inicializar localStorage.c
 - con parámetro por defecto para cuando se ejecute por primera vez
- ◆ Como la información se guarda ahora en localStorage y no en DOM
 - hay que actualizar primero localStorage y luego mostrar en DOM



Cronómetro: localStorage

```

<!DOCTYPE html>
<html>
<head><title>Cronómetro</title><meta charset="UTF-8">
<script type="text/javascript" src="zepto.min.js" > </script>
<script type="text/javascript">
  $(function(){
    localStorage.c = (localStorage.c || "0.0");

    var t, cl = $("#crono");

    function incr() { localStorage.c = +localStorage.c + 0.1; }
    function mostrar() { cl.html((+localStorage.c).toFixed(1)); };
    function arrancar() { t=setInterval(function(){incr(); mostrar();}, 100);};
    function parar() { clearInterval(t); t=undefined; };
    function cambiar() { if (!t) arrancar(); else parar(); };

    $("#cambiar").on('click', cambiar);
    $("#inicializar").on('click', function(){ localStorage.c="0.0"; mostrar();});
    mostrar();
  });
</script>
</head>
<body>
<h2>Cronómetro</h2>

<h3><span id="crono"> 0.0 </span> segundos </h3>

<button type="button" id="cambiar"> arrancar/parar </button>
<button type="button" id="inicializar"> inicializar </button>
</body>
</html>

```





JavaScript y HTML5: iframes y origin policy

iFrame

- ◆ **iFrame**
 - Importa un recurso Web
 - ◆ en un marco de navegación independiente
- ◆ Un iFrame crea una caja de arena segura
 - donde poder importar objetos externos
- ◆ Ejemplo: enlaza un juego en otro servidor
 - El iFrame evita que se introduzcan virus
 - ◆ Acceso JavaScript limitado a caja de arena



```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" >
    <title> Ejemplo iFrame</title>
  </head>
  <body>
    <h1>Canvas en iFrame</h1>

    iFrame con animación en canvas de
    <a href="http://www.chromeexperiments.com/mobile/">
    Google Chrome Mobile Experiments</a> <p><p>

    <iframe src="http://www.goodboydigital.com/runpixierun/"
    width="600" height="400"> </iframe>

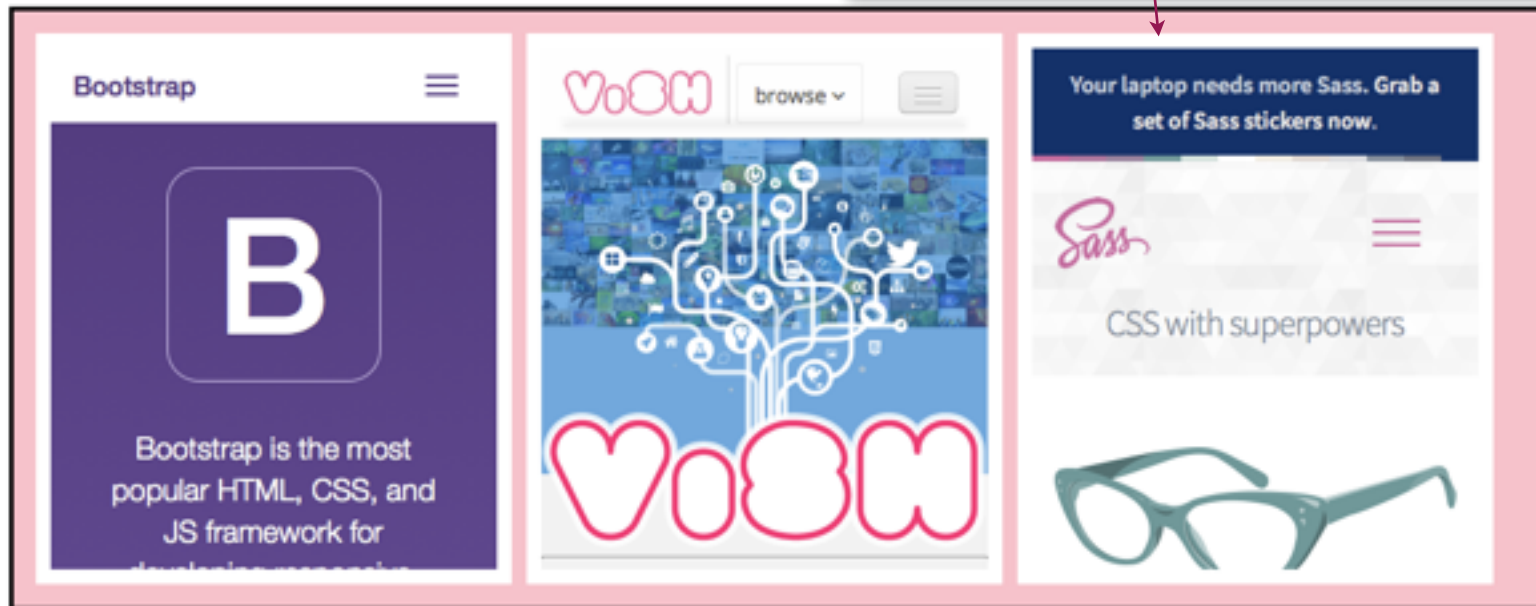
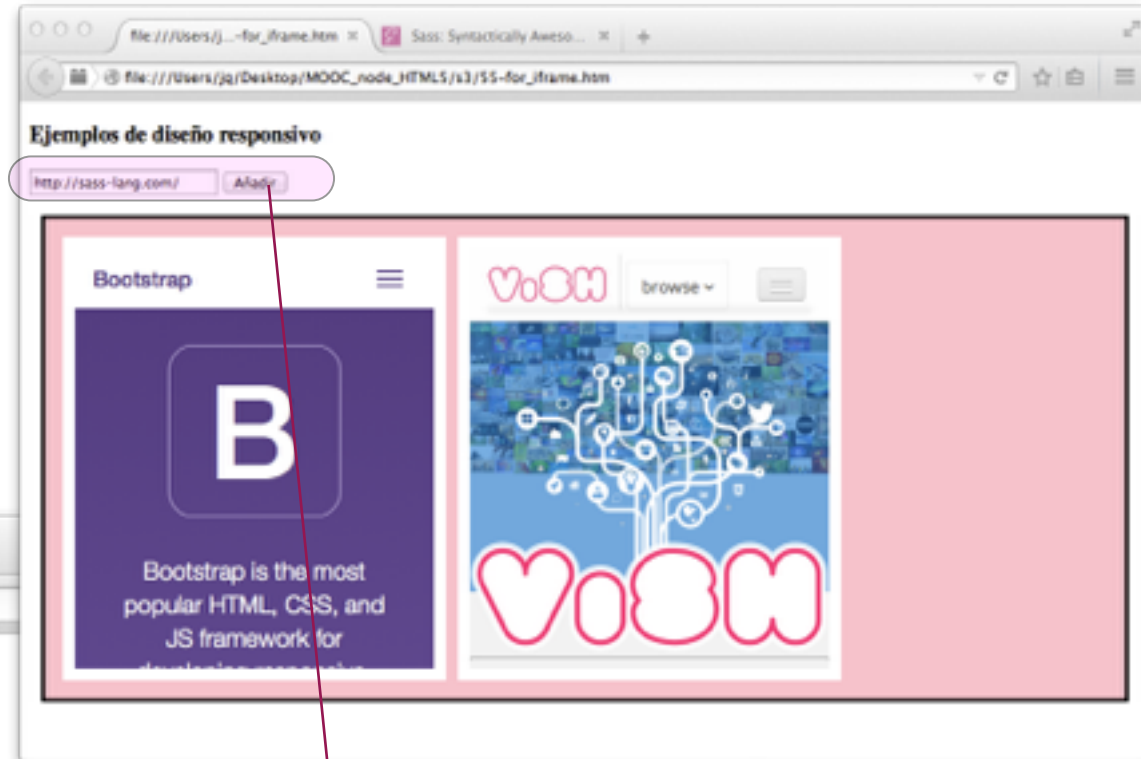
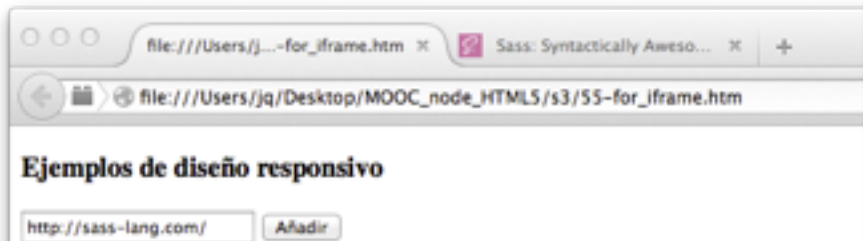
  </body>
</html>

```

Seguridad Web: “Same Origin Policy”

- ◆ La seguridad se controla en las aplicaciones JavaScript
 - Permitiendo que un **programa JavaScript** en un **iframe solo acceda**
 - ◆ Al **árbol DOM** de la **página principal** si **proviene del mismo origen**
 - Esto **evita** en el ejemplo anterior que el juego
 - ◆ **robe o modifique información o datos** del usuario en la página externa
- ◆ Origen
 - **protocolo, servidor y puerto** del URL
- ◆ La restricción de pertenecer al “mismo origen”
 - Solo afecta al recurso principal: página Web, recurso, ...
 - ◆ Los scripts o los estilos no están afectados y pueden venir de otro servidor
- ◆ Así es posible hacer “**mash-ups**” seguros
 - de contenidos que no estén en nuestra cadena de confianza

Ejemplo con iFrames



Flexbox

Flexbox permite un diseño
responsivo fácil y flexible:

display: flex;
flex-wrap: wrap;

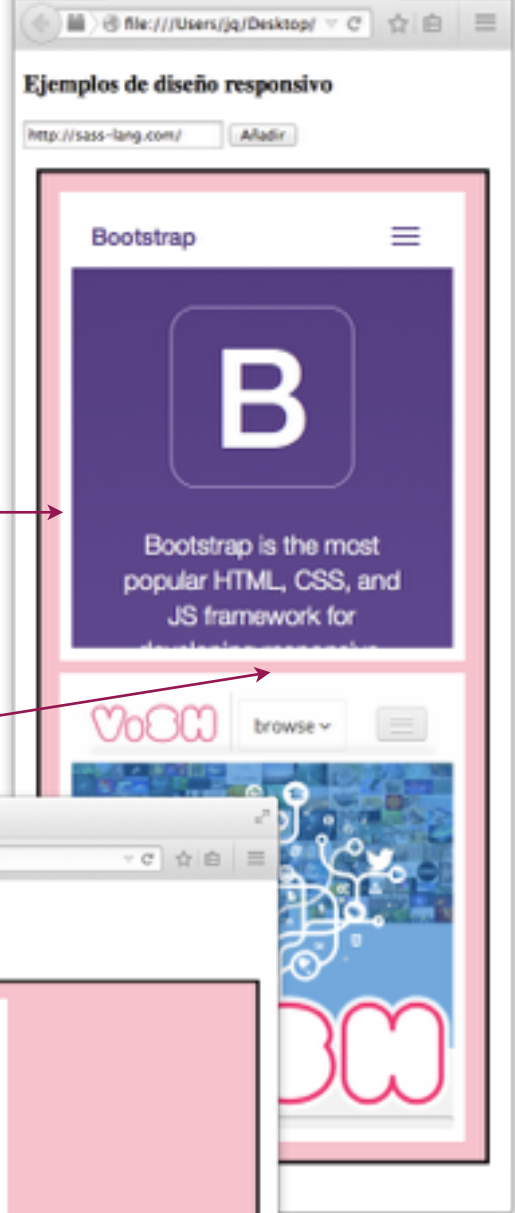
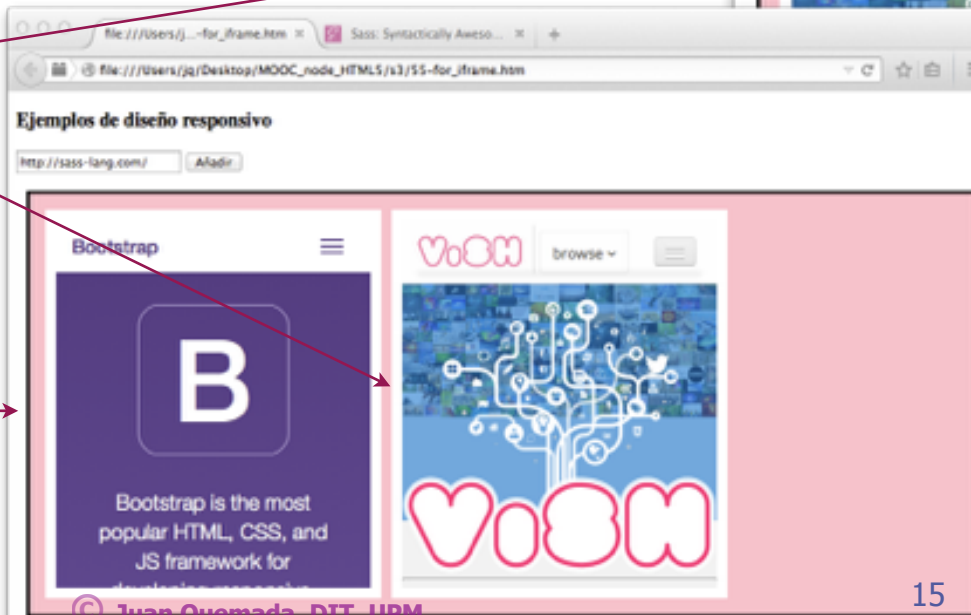
coloca cada iframe a la derecha
del anterior. Al llegar al limite
pasa a la línea siguiente.

```
<style>
```

```
iframe {  
  padding: 10px;  
  border: 5px solid pink;  
  background-color: white;  
  width: 300px;  
  height: 350px;  
}
```

```
#iframes {  
  display: -webkit-box;  
  display: -moz-box;  
  display: -ms-flexbox;  
  display: -webkit-flex;  
  display: flex;  
  
  -webkit-flex-wrap: wrap;  
  flex-wrap: wrap;  
}
```

```
#marco {  
  background-color: pink;  
  padding: 10px;  
  margin: 10px;  
  border: 3px solid black;  
}  
</style>
```



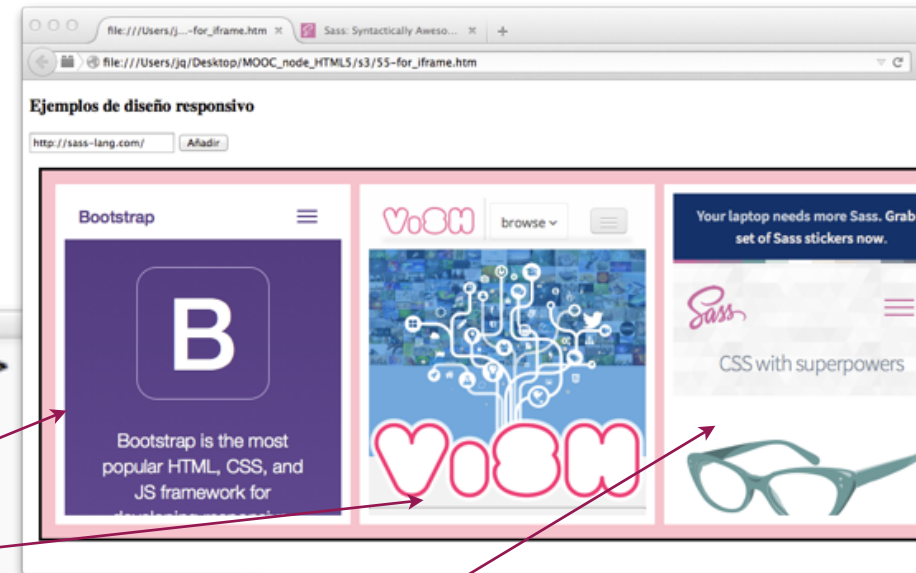
Código

```
<script type="text/javascript" src="zepto.min.js" >
</script>
<script type="text/javascript">
$(function(){
  var urls = ["http://getbootstrap.com",
             "http://vishub.org"];

  function mostrar(urls) {
    var i, iframes="";
    for (i=0; i < urls.length; ++i){
      iframes += "<iframe src='" + urls[i] + "'></iframe>";
    }
    $('#iframes').html(iframes);
  };

  $("#boton").on('click', function(){
    urls.push($('#nuevo').val());
    mostrar(urls);
  });

  mostrar(urls);
});
</script>
```



```
<body>
<h3>Ejemplos de diseño responsivo</h3>

<input type="text" id="nuevo" value="Nuevo URL" />
<button type="button" id="boton"> Añadir </button>
<p>
  <div id='marco'><div id="iframes" /></div>

</body>
```




JSON: JavaScript Object Notation

JSON

- ◆ JSON: formato textual de representación de tipos y objetos JavaScript
 - <http://json.org/json-es.html>
- ◆ Un **objeto JavaScript** se transforma a un **string JSON** con
 - **JSON.stringify(object)**
- ◆ Un **string JSON** se transforma en el **objeto original** con
 - **JSON.parse(string_JSON)**

```
var x = {a:1, b:{y:[false, null, ""]}}, y, z;
```

```
y = JSON.stringify(x);      => '{"a":1, "b":{"y":[false, null, ""]}}'
```

```
z = JSON.parse(y);          => {a:1, b:{y:[false, null, ""]}}
```

Serialización de datos

◆ Serialización:

- transformación **reversible** de un tipo u objeto (en memoria) en un **string equivalente**

◆ La serialización es un formato de intercambio de datos

- **Almacenar datos** en un fichero
- **Enviar datos** a través de una línea de comunicación
- **Paso de parámetros** en interfaces REST

◆ En JavaScript se realiza desde ECMAScript 5 con

- **JSON.stringify(...)** y **JSON.parse(...)**

◆ Otros formatos de serialización: XML, HTML, XDR(C), ...

- Estos formatos están siendo desplazados por JSON, incluso XML
 - ◆ Hay bibliotecas de JSON para los lenguajes más importantes

Características de JSON

◆ JSON puede serializar

- objetos, arrays, strings, números finitos, true, false y null
 - ◆ NaN, Infinity y -Infinity se serializan a null
 - ◆ Objetos Date se serializan a formato ISO
 - la reconstrucción devuelve un string y no el objeto original
- No se puede serializar
 - ◆ Funciones, RegExp, errores, undefined

◆ Admite filtros para los elementos no soportados

- ver doc de APIs JavaScript

```
JSON.stringify(new Date())    => '"2013-08-08T17:13:10.751Z"'
```

```
JSON.stringify(NaN)          => 'null'
```

```
JSON.stringify(Infinity)     => 'null'
```

JSON en ejemplo con iframes, array y for



```
<script type="text/javascript">
$(function(){
    localStorage.str1 = localStorage.str1
    || '["http://getbootstrap.com","http://vishub.org"]';

    var urls = JSON.parse(localStorage.str1);

    function mostrar(urls) {
        var i, iframes="";
        for (i=0; i < urls.length; ++i) {
            iframes += "<iframe src='" + urls[i] + "'></iframe>";
        }
        $('#iframes').html(iframes);
    };

    $("#boton").on('click', function(){
        urls = JSON.parse(localStorage.str1);
        urls.push($('#nuevo').val());
        mostrar(urls);
        localStorage.str1 = JSON.stringify(urls);
    });

    mostrar(urls);
});
</script>
```

Si queremos almacenar en el navegador el array de urls para que no se pierdan los urls introducidos, hay que guardarlo serializado con JSON en localStorage

```
<body>
<h3>Ejemplos de diseño responsivo</h3>

<input type="text" id="nuevo" value="Nuevo URL" />
<button type="button" id="boton"> Añadir </button>
<p>
<div id='marco'><div id="iframes" /></div>
</body>
```



Final del tema
Muchas gracias!

