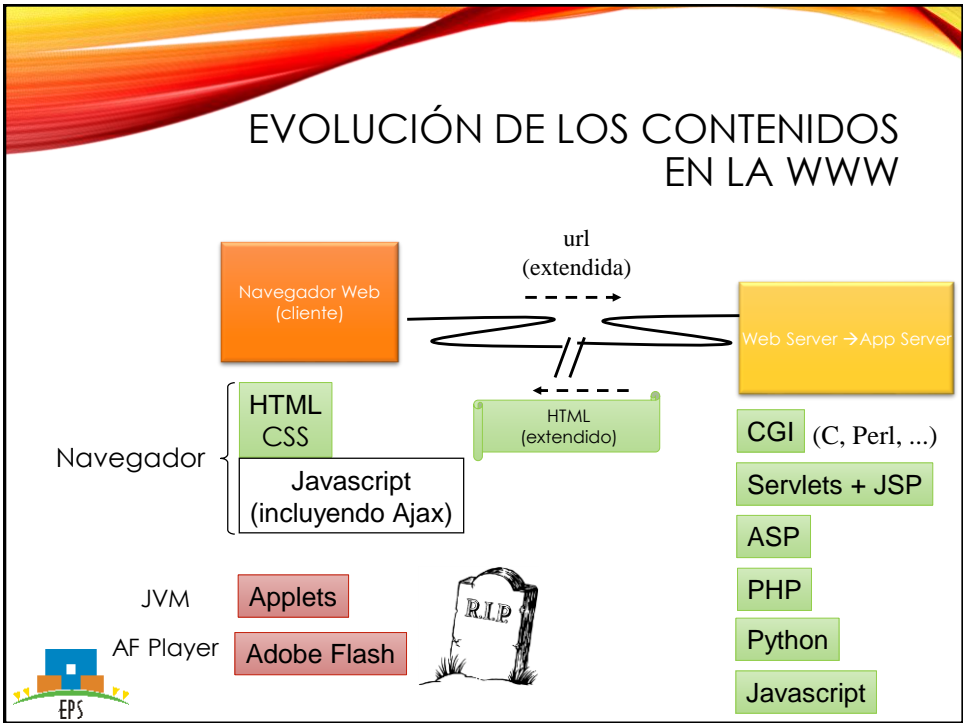


# SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB

Sistemas informáticos I

## 2.3 EJECUCIÓN DE CÓDIGO EN CLIENTE. JAVASCRIPT, AJAX, JQUERY

Sistemas distribuidos basados en la World Wide Web



# OBJETIVOS

- Aumentar la capacidad de interacción del usuario con la aplicación
- Disminuir el trasiego de datos entre el cliente y el servidor
  - Capturar los eventos generados por el usuario y responder a ellos sin depender del servidor
  - Realizar cálculos sencillos sin necesidad de comunicación con el servidor
  - Validar ciertos campos de formulario antes de enviarlos al servidor
- Algunas tecnologías asociadas al uso de *Javascript* en el cliente:
  - **DHTML** = *Dynamic HTML*
    - HTML + JavaScript + CSS
    - **NO** tiene nada que ver con el proceso de generación dinámica de documentos en el servidor (mediante Perl, PHP, JSP, ASP, etc.)
  - **AJAX** = *Asynchronous JavaScript And XML*
    - JavaScript + comunicación cliente-servidor asíncrona
      - Originalmente XML, actualmente más JSON
    - Google Web Toolkit (GWT): *framework* creado por Google que permite ocultar la complejidad de aspectos de la tecnología AJAX
    - *jQuery* es otra biblioteca

## JAVASCRIPT

- Originalmente propuesto por *Netscape Communications Corporation* en colaboración con Sun
- Estándar ECMA-262 → Estándar ISO → 10ª edición, junio 2019
  - <https://www.ecma-international.org/publications/standards/Ecma-262.htm>
- En su uso típico para desarrollar funcionalidad en cliente, enriquece las páginas HTML incorporando características dinámicas e interactivas
  - Interfaz de programación DOM (*Document Object Model*)
- Se ejecuta en cliente (en el navegador web)
- Usos comunes (algunos ejemplos):
  - Inclusión de efectos visuales en textos e imágenes de las páginas web
  - Manipulación de contenidos o aspecto de forma dinámica
  - Realización de operaciones matemáticas sencillas
  - Validación de datos introducidos en formularios
  - Gestión del sistema de navegación (menús desplegables)
  - Control del tipo y versión del navegador web, uso de la fecha y hora actuales, verificación de plugins...




<http://www.w3schools.com/js/default.asp>

## JAVASCRIPT


- Lenguaje de scripting interpretado
- Sintaxis muy similar a Java (y a C)
  - Sensible a mayúsculas y minúsculas
  - Fin de sentencia con ";" o con salto de línea
- Orientado a objetos: objetos, propiedades, métodos, eventos
  - Peculiaridad: el concepto de clase no se introduce hasta versiones muy recientes
- Orientado a eventos
- Es un lenguaje de programación
  - Per se no proporciona los mecanismos para implementar funcionalidad a ejecutar en un cliente web → HTML DOM





HTML DOM

- Cuando un navegador carga un documento HTML crea el modelo HTML DOM (*Document Object Model*) del documento
- El modelo HTML DOM es un **estándar W3C** que define objetos para representar:
  - Elementos HTML
  - Propiedades de los elementos HTML
  - Métodos para navegar por el árbol DOM
  - Eventos para capturar la interacción del usuario de la aplicación con los elementos HTML en la pantalla del navegador
- Es la implementación del modelo HTML DOM en el navegador la que permite desarrollar funcionalidad DHTML





JAVASCRIPT EN EL CLIENTE

- Interpretado por el motor *Javascript* del navegador
  - No se compila
  - No se ejecuta de forma independiente como aplicación autónoma
  - Se inserta en el código HTML y es interpretado por el navegador junto al resto del documento
- Enfocado a manejar elementos del documento y el navegador mediante objetos para acceder y manipular sus componentes
  - *Window*
  - *Document*
  - *Frame*
  - *Location*
  - *Navigator*
  - *History*
  - *Button*
  - *Textarea*
  - *Radio*
- Proporciona eventos para vincular funcionalidad a acciones del usuario:
  - *click*
  - *change*
  - *focus*
  - *load*
  - *mouseover*
  - *select*



# CÓDIGO JAVASCRIPT EN EL DOCUMENTO HTML

- El código *JavaScript* se incluye en el documento HTML mediante la etiqueta `<script>`
  - *inline*
  - En ficheros externos usando el atributo `src`

```
<script>
// Tu código irá aquí
</script>
```

```
<script src="codigo.js"></script>
```

- En cualquier parte del documento dentro del `head` o el `body`
- Tantos bloques de código dentro de etiquetas `<script>` como se necesiten



# EJEMPLO JAVASCRIPT

```
<html>
<head>
  <title>Ejemplos Javascript: ejemplo pr&aacute;ctico </title>
  <script language="JavaScript">
    var id,pause=0,position=0;

    function scorrevole() {
      var i, k, msg="Tres tristes tigres ... - ";
      k=(100/msg.length)+1;

      for(i=0;i<=k;i++) msg+=" "+msg;
      document.form2.scorrevole.value=msg.substring(position,position+100);
      if(position++==100) position=0;
      id=setTimeout("scorrevole()",100);
    }
  </script>
</head>

<body bgcolor="white" onload="scorrevole()">
  <form name=form2><input type="text" name="scorrevole" size="40"></form>
</body>
</html>
```



## VARIABLES Y FUNCIONES

```
var x = valor
```

```
function nombre([parametros]) {  
  // Cuerpo de la función  
  [return [valor]]  
}
```


• Tipado dinámico/débil:

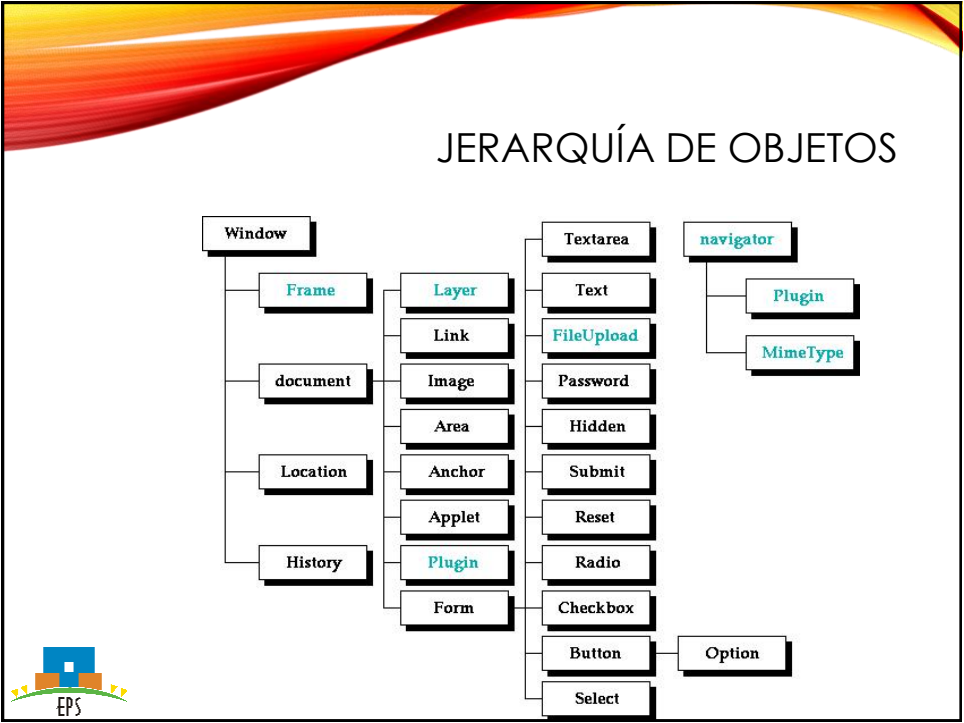
```
x = 2      // x es de tipo numérico, pero no se declara  
x = "hola" // El tipo de x se puede cambiar
```

• Siete tipos de datos:

- Number
- String
- Boolean
- Null
- Undefined
- Symbol
- Object

```
x = 1.7 // Número  
x = "hola" // Cadena de caracteres  
x = 'hola' // Cadena de caracteres  
x = 'hola ' + 'mundo' // Concatenación de cadenas  
x = true // Booleano  
x = null // Literal nulo  
x = [2, 3, 5, "Alice"] // Array, x[0], x[1], x[2], x[3]  
x = { a:2, b:'Bob' } // Array asociativo, x['a'], x['b']
```






# ¿CUÁNDO SE INTERPRETA EL CÓDIGO JS?


```
<HTML>
<HEAD>
</HEAD>
<BODY>
  <SCRIPT type="text/javascript">
    document.write("<H1><U>Títulos</U><BR></H1>")
    for (i=1; i<=6; i++) {
      document.write("<H" + i + ">Título " + i)
      document.write("</H" + i + "><BR>")
    }
  </SCRIPT>
</BODY>
</HTML>
```

Hay formas mejores de hacerlo



# ¿CUÁNDO SE INTERPRETA EL CÓDIGO JS?

```
<HTML>
<HEAD>
  <SCRIPT type="text/javascript">
    function mostrarTítulos() {
      document.write("<H1><U>Títulos</U><BR></H1>")
      for(i=1; i<=6; i++) {
        document.write("<H" + i + ">Título " + i)
        document.write("</H" + i + "><BR>")
      }
    }
  </SCRIPT>
</HEAD>
<BODY>
  <INPUT type="button" value="Mostrar" onClick="mostrarTítulos()">
</BODY>
</HTML>
```



## ¿CUÁNDO SE INTERPRETA EL CÓDIGO JS?

```
<HTML>
  <HEAD>
    <SCRIPT src="mostrar-titulos.js"></SCRIPT>
  </HEAD>
  <BODY>
    <INPUT type="button" value="Mostrar" onClick="mostrarTitulos()">
  </BODY>
</HTML>
```

```
function mostrarTitulos() {
  document.write("<H1><U>Titulos</U><BR></H1>")
  for (i=1; i<=6; i++){
    document.write("<H" + i + ">Titulo " + i)
    document.write("</H" + i + "><BR>")
  }
}
```



## OBJETOS

- Se pueden considerar arrays asociativos
- **Objeto = propiedades + métodos (+ eventos)**
- Propiedades
  - Características (atributos, elementos) de un objeto
  - Pueden ser a su vez objetos
  - Ejemplos: `unaCadena.length`, `window.document`
- Métodos
  - Funcionalidades (funciones) de un objeto
  - Ejemplos: `unaCadena.substring(2)`, `document.write('<p>Hola mundo</p>')`
- Eventos
  - Desencadenan la ejecución de cierta funcionalidad
    - por una acción del usuario: click de ratón, pulsación de tecla, etc...
    - asociada al control del documento o su lógica funcional: carga del contenido, minimización/maximización de la ventana...
  - Ejemplos:
 

```
<input type="button" onClick="window.alert('¡Botón pulsado!')>
```





## OBJETOS

- Creación de un objeto:

- Por asignación:

```
var profesor = new Object();  
profesor.nombre = "Roberto";  
profesor["apellido"] = "Latorre" // Incluso en tiempo de ejecución
```

- Con un inicializador:

```
var profesor = {nombre: "Alvaro", apellido: 'Ortigosa', 3: 5};
```

- Con un constructor:

```
function Profesor(nombre, apellido) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
}  
var profesor = new Profesor("José Antonio", "Macías");
```



## OBJETOS

- Y lo mismo para los métodos:

```
function imprimirProfesor(consola){  
    mensaje = this.nombre + " " + this.apellido;  
  
    if(consola) consola.log(mensaje);  
    else alert(mensaje);  
}  
  
function Profesor(nombre, apellido) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
  
    this.print = imprimirProfesor;  
}  
  
var profesor = new Profesor("El", "Boss");  
profesor.print(false);
```



## CLASES

- A partir de ECMAScript 2015:

```
class Profesor {  
  constructor(nombre, apellido) {  
    this.nombre = nombre;  
    this.apellido = apellido;  
  }  
  
  get nombreCompleto() { return this.nombre + " " + this.apellido; }  
  
  print(console) {  
    if(console) console.log(this.nombreCompleto);  
    else alert(this.nombreCompleto);  
  }  
}  
  
var profesor = new Profesor("Roberto", "Latorre");  
profesor.print(false);
```



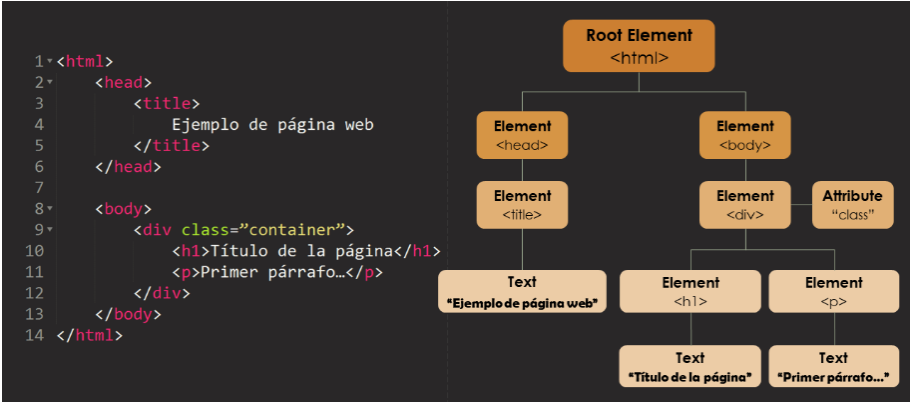
## NAVEGAR Y MANIPULAR DINÁMICAMENTE EL ÁRBOL DOM

- Tres métodos para obtener elementos específicos del árbol DOM:
  - `document.getElementById("id")`
  - `document.getElementsByTagName("tag")`
  - `document.getElementsByClassName("clase")`
- Una vez identificado un nodo:
  - Métodos para crear, eliminar, sustituir, concatenar... nodos:
    - `createElement`, `createTextNode`, `createAttribute...`
    - `insertBefore...`
    - `removeChild`, `removeAttribute...`
    - `appendChild...`
  - Recorrer el árbol en sus distintas dimensiones:
    - `childNodes`, `parentNode`, `nextSibling`, `previousSibling...`
  - Atributos para acceder/modificar el contenido HTML/Texto del nodo:
    - `innerHTML`, `outerHTML`
    - `innerText`, `outerText`



[https://www.w3schools.com/jsref/dom\\_obj\\_document.asp](https://www.w3schools.com/jsref/dom_obj_document.asp)  
[https://www.w3schools.com/jsref/dom\\_obj\\_all.asp](https://www.w3schools.com/jsref/dom_obj_all.asp)

# EJEMPLO DE DOCUMENTO HTML/ÁRBOL DOM



# EJEMPLO PRÁCTICO 1

**DEPRECATED**

- Entrada salida de datos mediante pop-ups

```
<HTML>
<BODY>
  <SCRIPT type="text/javascript">
    var num1, num2, sum
    num1 = window.prompt("Introduce el primer numero")
    num2 = window.prompt("Introduce el segundo numero")
    sum = window.parseInt(num1) + window.parseInt(num2)
    window.alert("Suma = " + sum)
  </SCRIPT>
</BODY>
</HTML>
```



## EJEMPLO PRÁCTICO 2

- Eventos de ratón y objeto evento

```
<HTML>
<HEAD>
  <SCRIPT type="text/javascript">
    function identificar(evento) {
      if(evento.button==0 || evento.button==1){
        window.alert("Has hecho click con el boton izquierdo") }
      else {
        window.alert("Has hecho click con el boton derecho")
      }
    }
  </SCRIPT>
</HEAD>
<BODY onmouseup="identificar(event)" onclick="identificar(event)">
  <H1>Haz click donde quieras!</H1>
</BODY>
</HTML>
```



## EJEMPLO PRÁCTICO 3

- Evento de carga del documento y acceso a objetos con el ratón

```
<HTML>
<HEAD>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <TITLE>Intercambio de imágenes</TITLE>
</HEAD>
<BODY onload="window.alert('¡Bienvenido! La página se ha cargado!')">
  <IMG id="unaImagen" src="bob1.jpg"
    onmouseover="this.src='bob2.jpg'"
    onmouseout="this.src='bob1.jpg'" />
</BODY>
</HTML>
```



# EJEMPLO PRÁCTICO 4

DEPRECATED

- Evento del foco

```
<HTML>
<HEAD>
  <TITLE> Color de Fondo </TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<CENTER>
<H1>Cambio del color de fondo</H1><BR>
<FORM>
  <INPUT type="text" name="bgr" value="rojo" onfocus="document.bgColor='red'"><BR>
  <INPUT type="text" name="bgg" value="verde" onfocus="document.bgColor='green'"><BR>
  <INPUT type="text" name="bgb" value="azul" onfocus="document.bgColor='blue'"><BR>
</FORM>
</CENTER>
</BODY>
</HTML>
```



# EJEMPLO PRÁCTICO 5

- Acceso a un objeto por ID

```
<HTML>
<HEAD>
  <SCRIPT type="text/javascript">
    function factorial(n) {
      if (n < 2) { return 1}
      else { return n * factorial(n-1)}
    }
  </SCRIPT>
</HEAD>
<BODY>
  <FORM>
    Entrada:&nbsp;
    <INPUT id="argumento" type="text"
      onchange="res=factorial(this.value);
        document.getElementById('resultado').value = res;" />
    <BR><BR> Salida:&nbsp;
    <INPUT id="resultado" type="text" />
  </FORM>
</BODY>
</HTML>
```



## EJEMPLO PRÁCTICO 6

- Acceso a un objeto por tipo de elemento

```
<HTML>
<HEAD>
  <STYLE>
    .clase1{text-decoration:none;color:red;}
    .clase2{text-decoration:none;color:green;}
  </STYLE>
  <SCRIPT type="text/javascript">
    function cambiarEstilo(estilo){
      var lista=document.getElementById("miLista");
      var items=lista.getElementsByTagName("li");
      for(i=0; i<items.length; i++){
        items[i].getElementsByTagName("a")[0].setAttribute("class",estilo);
      }
    }
  </SCRIPT>
</HEAD>
<BODY>
  <H1><UL id="miLista">
    <LI><A href="javascript:cambiarEstilo('clase1');">Estilo1</A></LI>
    <LI><A href="javascript:cambiarEstilo('clase2');">Estilo2</A></LI>
  </UL></H1>
</BODY>
</HTML>
```



## OCULTAR EL CÓDIGO JAVASCRIPT

- Si el navegador no reconoce *Javascript*, el código será mostrado como parte del contenido de la página
- Para evitarlo:

```
<html>
<body>
<script type="text/javascript">
  <!--
    document.getElementById("demo").innerHTML=Date();
  //-->
</script>
</body>
</html>
```

- Actualmente no es necesario



# IDENTIFICACIÓN DE OBJETOS

forms[0]

- elements[0]
- elements[1]
- elements[2]
- elements[3]
- links[0]
- links[1]
- links[2]
- images[0]
- images[1]
- images[2]

# IDENTIFICACIÓN DE OBJETOS

```
graph TD
    document["document<br/>página"] --> forms["forms[0]<br/>formulario"]
    document --> links0["links[0]<br/>primer hipervínculo"]
    document --> links1["links[1]<br/>segundo hipervínculo"]
    document --> links2["links[2]<br/>tercer hipervínculo"]
    forms --> elements0["elements[0]<br/>input: nombre"]
    forms --> elements1["elements[1]<br/>select: sexo"]
    forms --> elements2["elements[2]<br/>input: edad"]
    forms --> elements3["elements[3]<br/>input: botón submit"]
    links0 --> images0["images[0]<br/>primera imagen"]
    links1 --> images1["images[1]<br/>segunda imagen"]
    links2 --> images2["images[2]<br/>tercera imagen"]
```

Cuarto elemento del formulario (botón Submit):

```
document.forms[0].elements[3]
document.getElementById("botón submit")
document.getElementsByTagName("input")[2]
```

Se supone que está así definido el botón de submit:

```
<INPUT id="botón submit" type="submit" >
```


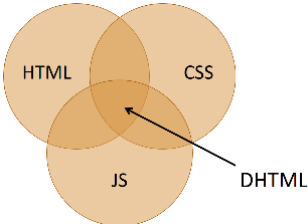
## MODIFICAR EL ASPECTO VISUAL DEL DOCUMENTO

- Atributos *style* y *className*

```
document.getElementById("mi_span").style.color = "#ff0000";
document.getElementById("mi_span").style.backgroundColor = "#00ff00";

document.getElementById("mi_span").className += " seleccionado";
```
- Atributo *classList*


```
document.getElementById("mi_span").classList.add("seleccionado");
```



## ALGUNOS COMENTARIOS SOBRE EVENTOS DOM

- Evento *change* vs. evento *input*

```
<input type="text" id="comida" size=30 value="" oninput="funcion()">
```
- Uso de event listeners
  - <http://stackoverflow.com/questions/6348494/addeventlistener-vs-onclick>





## EJERCICIO DE VALIDACIÓN DE FORMULARIO


- Realizar un documento HTML con un formulario como el de la figura:

Nombre:

Edad:

E-mail:

Enviar consulta
- No se enviarán datos al servidor hasta que:
  - El campo para el nombre tenga un "nombre" válido de al menos 2 caracteres, no permitiéndose caracteres en blanco ni al principio ni al final
  - El campo edad tenga una edad válida
  - El campo e-mail tenga una dirección válida (al menos contenga un carácter "@")
  - Las validaciones se tienen que hacer de forma programática con *JavaScript*, no se darán por válidas validaciones en HTML 5
- El formulario se enviará para su procesamiento a uno de los siguientes servicios <https://postman-echo.com/get> o <https://postman-echo.com/post>, el primero acepta peticiones en modo GET y el segundo en modo POST. Comparar el resultado obtenido en ambos casos te ayudará a comprender la diferencia entre ambos métodos de envío



## LOCALSTORAGE Y SESSIONSTORAGE

- Almacenes de datos en *window*
  - Almacena pares clave/valor: *setItem*, *getItem*, *removeItem* y *clear*
  - Permiten el acceso mediante sintaxis abreviada

```
<HTML>
<HEAD>
  <script type="text/javascript">
    if (! localStorage.visitas) localStorage.visitas = 1;
    else localStorage.visitas = Number(localStorage.visitas) + 1;

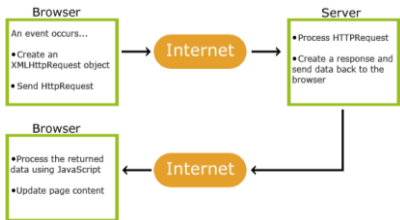
    if (! sessionStorage.visitas) sessionStorage.visitas = 1;
    else sessionStorage.visitas = Number(sessionStorage.visitas) + 1;

    document.write("Visitas totales " + localStorage.visitas + "<br>");
    document.write("Visitas en sesión " + sessionStorage.visitas);
  </script>
</HEAD>
<BODY>
</BODY>
</HTML>
```



# AJAX

- *Asynchronous JavaScript and XML*
- **NO** es un lenguaje de programación, sino una forma de usar tecnología existente en el momento en que se definió
- Conjunto de técnicas para intercambiar información con el servidor en background y actualizar partes de un documento HTML sin recargar la página completa
- Se basa en 4 estándares de Internet:
  - XMLHttpRequest object
  - JavaScript/DOM
  - CSS
  - XML
- Las aplicaciones AJAX no dependen del navegador ni de la plataforma!



# EJEMPLO DE FUNCIONAMIENTO

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <script type="text/javascript">
6          function cargaRespuesta() {
7              var xmlhttp = new XMLHttpRequest();
8              xmlhttp.onreadystatechange = function() {
9                  if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
10                     document.getElementById("respuesta").innerHTML = xmlhttp.responseText;
11                 }
12             }
13             var dir = "respuesta2.php?comida=" + document.getElementById("comida").value;
14             xmlhttp.open("GET", dir, true);
15             xmlhttp.send();
16         }
17     </script>
18 </head>
19 <body>
20     <h1> El crustaceo crugiente - eFood </h1>
21     <form name="miFormulario">
22         Por favor, ingrese el plato su elección
23         <input type="text" id="comida" size=30 value="" onkeyup="cargaRespuesta();">
24         <div id="respuesta" style="margin-top: 100px;">
25     </div>
26 </form>
27 </body>
```



## EL OBJETO XMLHttpRequest

- Se usa para realizar peticiones HTTP al servidor
- Soportado por todos los navegadores modernos
  - Hace unos años era necesario comprobar si el navegador lo soportaba
- Envío de la petición HTTP:
  - `open(tipo, url, asíncrona)`
    - Configuración de la petición  $\equiv$  definición de un formulario
    - tipo: "GET" o "POST"  $\equiv$  form.method
    - url: destino de la petición  $\equiv$  form.action
    - asíncrona: true o false. Indicador de asincronismo!!!
  - `send([query_string])`
    - Envía la petición (después de configurarla)  $\equiv$  submit
    - Petición GET: sin parámetros
    - Petición POST: recibe el query string (p.ej., "param1=valor1&param2=valor2")

```
xmlhttp.setRequestHeader("Content-type",  
                           "application/x-www-form-urlencoded")
```





## EL OBJETO XMLHttpRequest

- Gestión del ciclo de vida de la petición:
  - `readyState`
    - 0: la petición aún no se ha inicializado
    - 1: se ha establecido la conexión con el servidor
    - 2: se ha recibido la petición
    - 3: se está procesando la petición
    - 4: la petición ha terminado de procesarse y la respuesta está disponible
- Recepción de la respuesta:
  - Llamadas síncronas  $\rightarrow$  después del `send`
  - Llamadas asíncronas  $\rightarrow$  evento `readystatechange`
  - `status`: código de respuesta de la petición (p.ej. 200, 401, 403, 404 o 500)
  - `responseXML`: datos enviados por el servidor como isla de datos XML
  - `responseText`: datos enviados por el servidor en forma de string



# JQUERY

- Biblioteca que trata de simplificar la programación de la lógica *JavaScript* en el lado del cliente:
  - Acceso a la estructura del documento con una sintaxis similar a la de CSS
  - Fácil manejo de conjuntos de elementos
    - En las últimas versiones de *JavaScript* la diferencia no es muy significativa
  - Tratamiento específico de eventos
  - Comunicación con el servidor
- Debe ser importada:
  - Desde el propio servidor de la app web que estamos desarrollando
  - Desde un servidor público (CDN)
- Es una biblioteca muy liviana que desde servidores rápidos se descarga muy rápidamente, agregando poca sobrecarga al tiempo de descarga de la página
- Normalmente, una vez la página inicial es descargada, la biblioteca se cachea en alguna caché intermedia o la del propio navegador



# EJEMPLO JQUERY

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>jQuery demo</title>
</head>
<body>
  <a href="http://jquery.com/">jQuery</a>
  <script src=http://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js>
  </script>
  $(document).ready(function() {
    $("a").click(function(event) {
      alert("As you can see, the link no longer took you to jquery.com");
      event.preventDefault();
    });
  });
</script>
</body>
</html>
```



## FILOSOFÍA JQUERY

- Trabaja con estructuras similares a un array que permiten aplicar acciones sobre varios elementos en una misma sentencia
- Unifica la sintaxis para prácticamente cualquier acción *JavaScript*:
  1. Seleccionar el elemento o elementos HTML sobre los que queremos trabajar (de ahí el nombre *jQuery*)
  2. Invocar a una función que realiza una acción sobre ellos
  3. La función devuelve una lista de elementos → se pueden enlazar llamadas, incluso aunque la lista no contenga ningún elemento
- Sintaxis básica: `jQuery(selector).accion1().accion2()...`
- Sintaxis abreviada: `$(selector).accion1().accion2()...`
  - `$`: indica que lo que viene a continuación es una expresión *jQuery*
  - `selector`: idéntica los elementos HTML sobre los que trabajar reutilizando y ampliando la sintaxis de selectores CSS
  - `accionX()`: método *jQuery* al que invocar sobre los elementos seleccionados. Puede hacer referencia a la lectura/escritura de una propiedad del elemento, a un efecto visual, a una modificación DHTML, a la invocación o asignación de código de procesamiento de un eventos...



```
$("#div#contenedor_lista input:checked").parent().hide()
```

## ACCESO A LA ESTRUCTURA DEL DOCUMENTO

- Selector de elemento:

```
$('td');
```

- Selector *id*:

```
$('#nombre');
```

- Selector *class*:

```
$('.nombre-clase');
```

- Combinaciones de selectores:

```
$('div ul#nombre');
```

- Selector atributo:

```
$('[atributoN]');  
$('tipo[nombre-atributo]');  
$('tipo[nombre-atributo=valor]');  
$('tipo[nombre-atributo!=valor]');
```

- Pseudo-selectores:

```
$('tbody tr:even');  
$('section:first');  
$('tr:gt(0)');
```



<https://api.jquery.com/category/selectors/>  
[https://www.w3schools.com/jquery/jquery\\_ref\\_selectors.asp](https://www.w3schools.com/jquery/jquery_ref_selectors.asp)

## RECORRIENDO EL ÁRBOL DOM

- Funciones que facilitan recorrer la jerarquía. **Ejemplos:**

```
// todos los tr dentro de contactScreen
$('#contactScreen').find('tr');

// padres de los párrafos
$('p').parent();

// form antecesor de todos los input
$(':input').parents('form');

// labels hermanas de inputs con atributo required
$(':input[required]').siblings('label');

// todos los inputs + todos los labels del documento
$(':input').add('label');
```



## ITERACIÓN SOBRE CONJUNTOS DE ELEMENTOS

- Como la selección y las funciones *jQuery* devuelven algo que imita a un array, se puede utilizar cualquiera de los bucles estándar proporcionados por *JavaScript*
- *jQuery* ofrece además su propio bucle, más eficiente y con una sintaxis más sencilla:

```
$('#form :input').each(function(index, element) {})
```

- Los parámetros *index* y *element* de la función son, respectivamente, el índice y el valor de cada elemento sobre los que se itera
- En las versiones más recientes de *JavaScript* los arrays incorporan un método `forEach()` análogo al de *jQuery*
  - [https://www.w3schools.com/jsref/jsref\\_foreach.asp](https://www.w3schools.com/jsref/jsref_foreach.asp)



## TRATAMIENTO DE EVENTOS

- jQuery gestiona los eventos DOM estándar y añade alguno adicional
  - <https://api.jquery.com/category/events/>
- Para asignar el código que debe procesar un evento se sigue la misma sintaxis general de cualquier sentencia jQuery

```
$(selector).evento([codigoJS])
```

- *evento* es el método que representa el evento que se quiere procesar (*click*, *keypress*, *load*, etc)
- Si no se pasa el parámetro *codigoJS* se dispara el evento correspondiente
- Un evento ampliamente utilizado es el *ready* del *document*

```
$(document).ready(function() {  
    // Tu código irá aquí  
})
```

```
$(function() {  
    // Tu código irá aquí  
})
```



## MANIPULACIÓN DINÁMICA DEL DOCUMENTO HTML

- Texto contenido de un elemento:

```
$('#contactDetails h2').text('CONTACT DETAILS');
```

- HTML de un elemento:

```
$('#contactDetails h2').html('<span>Contact Details</span>');
```

- Valor de un campo de formulario:

```
$('[name="contactName"]').val('testing 123');
```

- Valor de un atributo:

```
$('#textarea').attr('maxlength', 200);
```

- Quitar valor de un atributo:

```
$('#textarea').removeAttr('maxlength');
```

- Ocultar un elemento:

```
$('#loading').hide();
```



MANIPULACIÓN DINÁMICA DEL DOCUMENTO HTML

```
<a href="" onclick="alert('Hello world')">Link</a>
```

```
$(function() {  
  $("a").click(function() {  
    alert("Hello world!");  
  });  
});
```



JS

↑

Pero tiene que ponerse en todos y cada uno de los enlaces del documento!!



MANIPULACIÓN DINÁMICA DEL DOCUMENTO HTML

- Supongamos que queremos agregar un \* rojo a la izquierda de cada input "required" del documento


Contact name

\*

- Esto se puede conseguir con una única sentencia jQuery:

```
$(':input[required]').siblings('label').  
  append($('<span>').text('*')).  
  addClass('requiredMarker');
```


```
.requiredMarker {  
  color: red;  
  padding-left: 7px;  
}
```






## PETICIONES AL SERVIDOR

- `ajax({parametro:valor, parametro:valor, ... })`
  - `url`: url de la petición
  - `async`: el valor por defecto es `true`
  - `type`: "GET" o "POST"
  - `data`: datos que se van a mandar al servidor asociados a la petición
  - `contentType`: el valor por defecto es "application/x-www-form-urlencoded"
  - `dataType`: tipo de datos esperado en la respuesta (p.ej., "xml", "html", "json")
  - `context`: indica cuál va a ser el valor de `this` para las funciones callback
  - `success`: función o array de funciones callback que ejecutar cuando la petición finaliza de forma satisfactoria
  - `error`: función o array de funciones callback que ejecutar en caso de error en la petición
  - `timeout`: timeout de la petición expresado en milisegundos
- `get(url [,data] [,success] [,dataType])`
- `getJSON(url [, data] [, success])`
- `getJSON(url [, data] [, success])`
- `getScript(url [, success])`
- `post(url [, data] [, success] [, dataType] )`
- `load(url [, data] [, complete])`

<https://api.jquery.com/category/ajax/>






## COMENTARIO FINAL SOBRE FRAMEWORKS


- "Real-world programming" rara vez se hace completamente desde 0
  - Sólo con HTML + CSS + JS "desnudos"
- Existen múltiples frameworks (e incluso CMSs) que ayudan en la tarea
- Dos aspectos importantes:
  - Ninguna herramienta es la "bala de plata" que va a solucionar todos los problemas del mundo
  - Mientras que la tecnología base es estable a medio plazo (HTML 1991, CSS 1996, JS 1995), los frameworks y tecnología similar son mucho más efímeros y cambiantes
- Moraleja: poner especial cuidado a la hora de elegir un framework o biblioteca de apoyo al desarrollo



## ALGUNOS FRAMEWORKS POPULARES

- Bootstrap
  - Bootstrap is the most popular HTML, CSS, and JS framework for developing **responsive**, mobile first projects on the web
- ZURB Foundation
  - The most advanced **responsive** front-end framework in the world
- HTML5 Boilerplate
  - The web's most popular front-end template
- AngularJS
  - HTML enhanced for web apps!
- ReactJS
  - A JavaScript library for building user interfaces





## DESARROLLOS INTERESANTES

- Meteor
  - <https://www.meteor.com/>
  - "Meteor is a complete open source platform for building web and mobile apps in pure JavaScript"
  - Claves:
    - El mismo código se va a ejecutar en todos los clientes (navegadores y dispositivos móviles) y en el servidor
    - Cambios en los datos se reflejan inmediatamente en todas ls visualizaciones
    - Un comando envía la app a producción y actualiza todos los dispositivos conectados (sin necesidad de pasar por la app store de turno)



