Escuela Politécnica de Ingeniería
Grado de Ingeniería Informática en Tecnologías de la Información

Tecnologías Web

Tema 2

Tecnologías web de cliente: Javascrip, Ajax y JQuery

Índice

- Jascript
- Ajax
- JQuery


JavaScript

Introducción (1)

- ¿ Qué es un lenguaje de *script* ?
  - Un lenguaje de *script* permite adaptar, manipular y automatizar tareas de un sistema existente.
    - Extiende las capacidades de la aplicación con la que trabajan.
    - Habitualmente realizan diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario.

Introducción (2)

- ¿ Qué es ECMAScript ?
  - Especificación de lenguaje de programación publicada por *ECMA International* e inicialmente basada en el lenguaje JavaScript desarrollado originalmente por Netscape.
    - El desarrollo de estándar comenzó a finales de 1996 y la primera edición de este estándar es de junio de 1997.
    - Actual: ECMA-262, ECMAScript Language Specification (5.1 Edition / Junio 2011).

Introducción (3)

- JavaScript
  - Lenguaje interpretado débilmente tipado, orientado a objetos y guiado por eventos.
  - Compatible con el estándar ECMAScript (dialecto).
  - Uso habitual en el lado cliente
    - Permite mejorar el interfaz de usuario, personalizando y haciendo más interactivas las páginas web.

Introducción (4)

- JavaScript del lado cliente
  - Los navegadores pueden interpretar el código JavaScript
    - El cliente realizan peticiones al servidor web y reciben tanto código HTML como los *scripts* incluidos en dicho código.
    - El navegador lee el código recibido de arriba a abajo y muestra el código HTML y ejecuta los *scripts* en el orden en que aparecen.

Introducción (5)

- Ubicación de los *scripts*
  - Código JavaScript incluido en el elemento `script` de HTML o en un fichero externo referenciado por dicho elemento.
    - En general dentro del elemento `head` de cabecera (definición de funciones y objetos).
    - En el cuerpo (`body`) del documento HTML si generan una salida.
  - Código JavaScript directamente en atributos de tipo evento de ciertos elementos HTML.
  - Como referencia de un hipervínculo.

Introducción (6)

- Ejemplos

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Ejemplo</title>
<script type="text/javascript">
<!--
      alert("¡ Buenos días !");
-->
</script>
</head>
<body>
</body>
</html>
```

Introducción (7)

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Escritura de datos en documento</title>
</head>
<body>
     <h1>Título principal</h1>
     <script type="text/javascript">
     <!--
          document.write("<p>Fecha de la última modificación: ");
          document.write(document.lastModified + "</p>");
     -->
     </script>
     <p>Resto del documento ...</p>
</body>
</html>
```

JavaScript (1)

- Estructura léxica
  - Conjunto de reglas que especifican cómo escribir programas en este lenguaje.
    - Los programas están escritos utilizando el conjunto de caracteres Unicode.
    - Sensible a la diferencia entre minúsculas y mayúsculas (*case-sensitive*).
    - Se ignoran los espacios entre *tokens*. Los programas se pueden (y deben) formatear e indentar para hacerlos más legibles.

JavaScript (2)

- Comentarios
  - Soporta comentarios de línea, desde `//` hasta el fin de línea.
  - Y comentarios de varias líneas, entre `/*` y `*/`.
- Literales
  - Valores que pueden aparecer directamente en un programa.

JavaScript (3)

Tipo	Valores
	`null`, `undefined` e `Infinity`
Cadenas de caracteres	`"ejemplo 1"`, `'ejemplo 2'`
Numéricos	`120`, `1.5`, `1.3E+6`
Booleanos	`true` y `false`
Arrays	`[1, 2, 3, 4, 5]`
Expresiones regulares	`/(0[1-9])|(1[0-2])/`
Objetos	`{x:1, y:2}`

JavaScript (4)

- Identificadores y palabras reservadas
  - Un **identificador** es un nombre simple. Se utilizan como nombres de variables y de funciones y proporcionan etiquetas para ciertos bucles.
    - Deben comenzar por una letra, un carácter de subrayado (_), o el símbolo $. Los caracteres posteriores pueden ser cualquier carácter alfanumérico, carácter de subrayado o carácter dólar.

JavaScript (5)

Palabras Reservadas

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	nul	try	

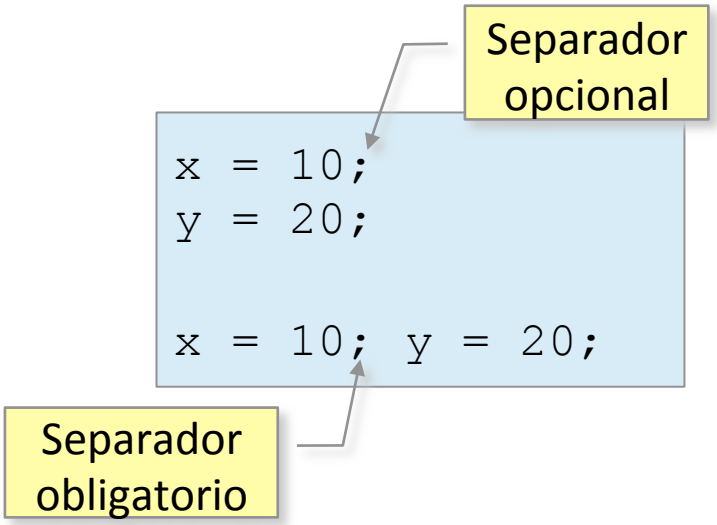
JavaScript (6)

- Separación de sentencias
  - Al igual que otros lenguajes de programación en JavaScript se utiliza el punto y coma (;) como separador de sentencias.
    - En JavaScript se puede omitir el carácter ; entre sentencias que están en líneas distintas.

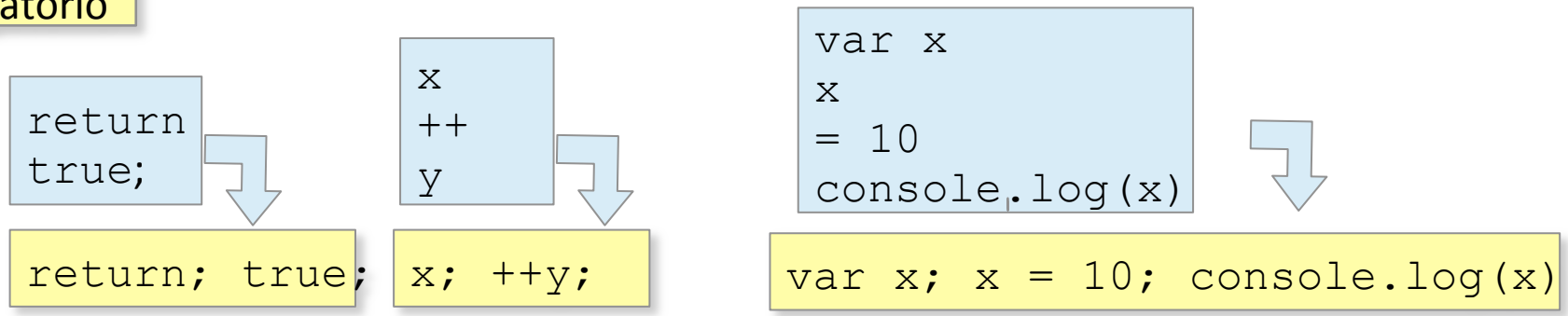
Su omisión puede provocar errores en el código.

Mejor opción: incluir el separador aunque no sea necesario.

JavaScript (7)



- JavaScript no trata cada salto de línea como si hubiera un separador (;). Sólo es así cuando no puede interpretar el código sin éste (con un par de excepciones).



JavaScript (8)

- Tipos de datos
  - *Primitivos*: `Undefined`, `Null`, `Boolean`, `Number` y `String`.
  - *Objetos*: *`Global`*, `Object`, `Function`, `Array`, `String`, `Boolean`, `Math`, `Date`, `RegExp` y objetos de error.
    - Los tipos primitivos son no modificables, mientras que los tipos objeto son modificables.

JavaScript (9)

- Números
  - En JavaScript no se establece diferencia entre valores enteros y de punto flotante.
  - Se representan utilizando el formato de punto flotante de 64 bits (estándar IEEE 754).
  - Literales enteros
    - JavaScript reconoce valores decimales (1234) y hexadecimales (0xff).

JavaScript (10)

- Literales de punto flotante
  - Con punto decimal (2.45) o con formato exponencial (1.473E-12).
- Aritmética
  - En JavaScript pueden utilizarse los operadores aritméticos habituales: *suma* (+), *resta* (-), multiplicación (*), división (/) y resto de la división (%).
  - También soporta operaciones matemáticas más complejas mediante funciones y constantes definidas como propiedades del objeto `Math`.

JavaScript (11)

- En JavaScript no se producen errores en caso de desbordamiento o división por cero. En estos casos el resultado es ±∞ y se representa con `Infinity` o `-Infinity`.
- En casos de indeterminación como, por ejemplo 0/0 o ∞/∞, el resultado es `NaN` (*not a number*).
- El objeto `Number` define propiedades adicionales: `MAX_VALUE`, etc.

JavaScript (12)

- Texto
  - Un *string* es una secuencia ordenada de valores de 16 bits (Unicode) no modificable.
  - Un *string* literal se proporciona encerrando la secuencia de caracteres entre comillas simple o entre comillas dobles.
    - El literal puede expresarse en varias líneas finalizando cada línea con el carácter `\`.
    - El carácter `\`, también se utiliza para escapar caracteres especiales (`\\`, `\'`, `\”`, `\n`, etc.).

JavaScript (13)

- Manipulación de *strings*
  - El operador + se utiliza como operador de concatenación.
  - Otras operaciones están proporcionadas como funciones del objeto `String`. Adicionalmente, la propiedad `length` de éste determina el tamaño de un *string*.
- Reconocimiento de patrones
  - En JavaScript se pueden crear objetos que representan patrones de texto (`RegExp`). Estos patrones están descritos con *expresiones regulares*.

JavaScript (14)

- Una expresiones regular literal se denota encerrando ésta entre `/`.
- El reconocimiento de patrones se puede realizar tanto con funciones de `String` como con las funciones expuestas por el objeto `RegExp`.

```
s = "Hola mundo.";
s[0];                          // 'H'
s.charAt(s.length-1);    // '.'
s.toUpperCase();               // "HOLA MUNDO."
```

```
var text = "test: 1, 2, 3";
var pattern = /\d+/g;
pattern.test(text);            // true
text.search(pattern);    // 6
text.match(pattern);           // ['1', '2', '3']
```

JavaScript (15)

- Booleanos
  - Valores booleanos: `true` y `false`.
  - Los valores `undefined`, `null`, `0`, `NaN` y "" se pueden convertir al valor booleano `false`. Cualquier otro valor (incluidos objetos y *arrays*) se convierten como `true`.

JavaScript (16)

- El objeto `Global`
  - Las propiedades y funciones de este objeto son símbolos definidos globalmente que están disponibles para los programas JavaScript.

Ejemplos

- `undefined`, `Infinity` y `NaN`
- `isNaN()`, `parseInt()`, `eval()`
- `String()`, `RegExp()`, `Array()`
- `Math`, `JSON`

  - Las variables y funciones de usuario definidas en el nivel superior también son propiedades y funciones del objeto.

JavaScript (17)

- Objetos envoltorio (*wrapper objects*)
  - Aunque un *string* literal (o una variable al que se le asigne éste) no es un objeto, se puede acceder a las propiedades e invocar las funciones del objeto (**métodos**) mediante la notación '.'.
  - Para ello, números, booleanos, expresiones regulares y *strings* **se convierten temporalmente** en objetos creados mediante los constructores correspondientes (*wrapper objects*).

JavaScript (18)

- Valores primitivos y objetos
  - Como ya se ha indicado los valores primitivos no son modificables los objetos si lo son.
  - Al comparar en los primeros se comparan valores en los segundos se comparan referencias.
    - Comparación de valores: operador ==
    - Comparación de referencias: operador ===

JavaScript (19)

```
var s = new String("Hola mundo.");
var t = new String("Hola mundo.");
s.len==t.len;
      // true
s===t;
      // false
```

- Conversiones de tipos
  - JavaScript es muy flexible en los tipos de valores requeridos (p.e., la

```
10 + "dato";        // convierte 10 a string
"3" * "6";          // convierte los strings en números
```

JavaScript (20)

Valor	Conversión			
	String	Number	Boolean	Object
undefined	"undefined"	NaN	false	*Throws TypeError*
null	"null"	0	false	
true	"true"	1		new Boolean(true)
false	"false"	0		new Boolean(false)
""		0	false	new String("")
"1.5"		1.5	true	new String("1.5")
"abc"		NaN	true	new String("abc")
0	"0"		false	new Number(0)
NaN	"NaN"		false	new Number(NaN)
Infinity	"Infinity"		true	new Number(Infinity)
-Infinity	"-Infinity"		true	new Number(-Infinity)
1	"1"		true	new Number(1)

JavaScript (21)

- Declaración de variables
  - Las variables deben declararse antes de utilizarlas mediante la palabra reservada `var`.
    - Se pueden declarar varias variables con el separador ',' y se pueden inicializar.

```
var msg="hola", i=0;
var num;
```

  - Ámbito
    - El ámbito de una variable es la región del código en el que se define.

JavaScript (22)

- Una variable *global* tiene un ámbito global. Es una propiedad del objeto `Global`.
- Una variable declarada dentro en una función sólo está definida en el cuerpo de la función, es una variable *local*. Las variables utilizadas en una función y no declaradas son globales.

```
var global1 ="global";
function ejemplo () {
            var local = "local";

            global2 = local;
}
global1;  // “global”
global2;  // “local”
```

JavaScript (23)

- Operadores
  - Aritméticos
    - +, −, *, /, %, ++, −−
  - Operadores de comparación
    - ==, !=, ===, !==, <, <=, >, >=
  - Operadores lógicos
    - !, &&, ||

JavaScript (24)

- Operadores de manipulación de bits
  - `&`, `|`, `^`, `~`, `<<`, `>>`
- Otros
  - `:?`, `in`, `instanceof`, `new`, `delete`, `void`, `typeof`

- Evaluación de expresiones
  - JavaScript puede interpretar una cadena de caracteres como código fuente, evaluarlo y producir un valor.

JavaScript (25)

- Sentencias
  - Asignación: `n=0`
  - Bloque: `{ x=2.1; y=x*x }`
  - Declaración:
    - `var n=0;`
    -
```
function fact (n) {
    if (n<=1) return 1;
    return n*fact(n-1);
}
```

JavaScript (26)

- Condicionales
  - Alternativa simple y compuesta
    - ```
      if (x == undefined)
          x = 100;
      ```
    - ```
      if (x>0)
          y = x;
      else
          y = -x;
      ```
  - Alternativa múltiple
    - ```
      switch (expresión) {
          sentencias
      }
      ```
      ```

JavaScript (27)

- Bucles
  - `while`, `do-while`, `for`
    - `for (prop in object)`
      `sentencia;`
    - ECMAScript no especifica el orden en el que se enumeran las propiedades de un objeto con el bucle `for-in`, pero en la práctica las implementaciones JavaScript de los principales navegadores las enumeran en el orden en que fueron creadas.
- Saltos
  - `break`, `continue`, `return`, `throw` y bloques `try-catch`

JavaScript (28)

- Otras sentencias
  - `with`
    -
```
with (documents.forms[0]) {
    name.value = “”;
    email.value = “”:
}
```
  - `debugger`

JavaScript – Objetos (1)

- Objetos JavaScript
  - Colección de propiedades no ordenada, cada una de las cuales consta de un nombre y un valor.
    - El valor puede ser cualquier valor JavaScript o el resultado de una función de acceso (*getter*).
  - Propiedades y objetos tienen asociados ciertos valores denominados *atributos de propiedades* y *atributos de objeto*, respectivamente.

JavaScript-Objetos (2)

- Atributos de propiedades con valor *(data descriptor)*
  - `writable`. Especifica cuando se puede poner el valor de la propiedad.
  - `enumerable`. Especifica cuando se puede obtener el nombre de la propiedad en un bucle `for-in`.
  - `configurable`. Especifica cuando se puede eliminar la propiedad y cuando se pueden alterar sus atributos.

JavaScript-Objetos (3)

- Atributos de propiedades con funciones de acceso *(accessor descriptor)*
  - En este caso, además de los atributos `enumerable` y `configurable` vistos anteriormente:
    - `get`. Especifica la función que retorna el valor de la propiedad.
    - `set`. Especifica la función que pone el valor de la propiedad.

JavaScript-Objetos (4)

  - Una propiedad puede tener *atributos de valor* o *atributos de acceso* pero no ambos.

- Atributos de objeto
  - `prototype`. Referencia a otro objeto del cual hereda sus propiedades.
  - `class`. Un *string* que categoriza el tipo de un objeto.
  - `extensible`. Especifica cuando se pueden añadir nuevas propiedades al objeto.

JavaScript-Objetos (5)

- Categorías de objeto
  - *Objeto nativo*. Es un objeto o clase de objetos definido por la especificación ECMAScripts (p.e., array, functions, dates, etc.).
  - *Objeto anfitrión*. Es un objeto definido por el entorno en el que el intérprete JavaScript está alojado (p.e., objetos `HTMLElement` que representan la estructura de una página web).
  - *Objeto definido por el usuario*. Es un objeto creado por la ejcución del código JavaScript.

JavaScript-Objetos (6)

- Categorías de propiedades
  - *Propiedades propias del objeto*. Son las propiedades definidas directamente sobre un objeto.
  - *Propiedades heredadas*. Son las propiedades definidas por el *objeto prototipo* de un objeto.
- Creación de objetos
  - Objetos literales. Lista de pares `nombre:valor` separados por comas y encerrados entre llaves.

JavaScript-Objetos (7)

```
var empty = { }; // objeto vacío
var libro = { "un titulo":"abc" , paginas:10 };
var segmento = {
     izq: { x:0, y:0},
     der: { x:0, y:-2}
};
```

- Mediante el operador `new`
  - El operador `new` crea e inicializa un nuevo objeto y debe estar seguido de la invocación a una función. La función, usualmente denominada *constructor*, sirve para inicializar el nuevo objeto creado.

JavaScript-Objetos (8)

```
var obj = new Object();          // igual que { }
var a = new Array();             // igual que []
var d = new Date();              // crea un objeto fecha
var r = new RegExp(/\d{1,4}/);
```

- También se pueden utilizar funciones definidas por el usuario.

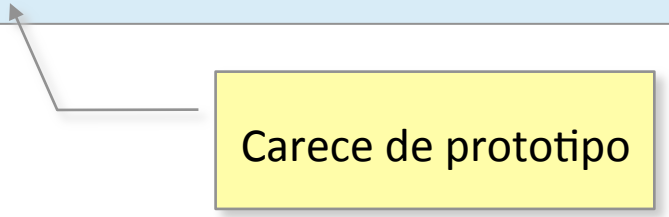
```
function Range (from, to) {
     this.from = from;
     this.to = to;
}

var r = new Range(10, 20);
```

JavaScript-Objetos (9)

- Mediante el método `Object.create()`
  - El método `Object.create()` crea un nuevo objeto pasando como primer argumento el objeto prototipo.

```
var o1 = Object.create({x:1, y:2});       // hereda x e y
var o2 = Object.create(Object.prototype); // como {}
```



JavaScript-Objetos (10)

```
function inherit (p) {
      if (p == null) throw TypeError();
      // ECMAScript 5
      if (Object.create)
            return Object.create(p);
      // ECMAScript 3
      var t = typeof p;
      if (t !== "object" && t !== "function")
            throw TypeError();
      function f () {};
      f.prototype = p;
      return new f();
}
```

JavaScript-Objetos (11)

- Acceso a las propiedades de un objetos
  - Mediante el operador .
  - Mediante el operador [ ]
    - También se puede utilizar este operador porque el objeto se puede ver como un *array* asociativo donde la clave es el nombre de la propiedad.

JavaScript-Objetos (12)

- Otras operaciones sobre propiedades
  - Eliminación: operador `delete`
  - Test: `obj.hasOwnProperty("prop")`
  - Enumeración: bucle `for-in`

Falso si la propiedad es heredada

- Definición de atributos de propiedades
  - Por defecto las propiedades definidas por el usuario tienen valor y son modificables, configurables y se pueden eliminar.
    - Para establecer otros atributos se utiliza una sintaxis similar a la indicada para los objetos literales con el método `Object.defineProperty`

JavaScript-Objetos (13)

```
Object.defineProperty(Objeto.prototype, "valor",
      {
      value: 0,
        configurable: false,
        enumerable: false,
        writable: true
      });
Object.defineProperty(Objeto.prototype, "num",
      {
        configurable: false,
        enumerable: false,
        get: function () {return this.valor; },
        set: function (newValue) { this.valor = newValue; }
      });
function Objeto (str) {
      this.base = new String(str);
      this.object = "obj_" + Objeto.prototype.num++
      eval(this.object + "= this");
}
```

JavaScript-Objetos (14)

- Clases y prototipos
  - Clase: conjunto de objetos que heredan propiedades de un mismo objeto prototipo.
  - Habitualmente se define una función constructora para crear e inicializar los objetos. La función proporciona el *prototipo*.

Toda definición de función es un objeto (`Function`) que dispone del campo `prototype`.

```
var F = function() {};
F === F.prototype.constructor;  // return true
```

El objeto prototipo incluye el campo `constructor` que referencia la función constructora.

JavaScript-Objetos (15)

- El nombre del constructor identifica el nombre de la clase de objetos que crea.

El operador `instanceof` permite comprobar si un objeto es de una cierta clase.

```
var obj = new F();
obj instanceof F;  // return true
```

- Los métodos de los objetos son propiedades del objeto prototipo cuyo valor son funciones.
- Las propiedades que se añadan al constructor sirven como campos y métodos de clase.
- Las instancias de la clase pueden tener sus métodos y propiedades propias.

JavaScript-Objetos (16)

```
function Range (from, to) {
      this.from = from;
      this.to = to;
}

Range.prototype.includes =
      function (x) {
            return this.from <= x && x <= this.to;
      };
Range.prototype.foreach =
      function (f) {
            for (var x = Math.ceil(this.from); x <= this.to; x++)
                  f(x);
      };
Range.prototype.toString =
      function () {
            return "(" + this.from + "..." + this.to + ")";
      };
```

```
var r = new Range(1, 5);
r.includes(3);              // return true
r instanceof Range; // return true
```

JavaScript-Objetos (17)

- La herencia basada en prototipos de JavaScript es dinámica: todos los objetos heredan las propiedades de sus prototipos, incluso si éstos cambian después de haber sido creados.
  - Es posible extender una clase en cualquier momento sin más que añadir nuevas propiedades al objeto prototipo.

JavaScript-Objetos (18)

- Definición de subclases
  - Para definir una subclase el objeto prototipo de la subclase debe heredar las propiedades del objeto prototipo de

```
function RangeSubclass (to) {
      this.str = "";
      Range.call(this, 0, to);
};

RangeSubclass.prototype = new Range();
RangeSubclass.prototype.constructor = RangeSubclass;
```

JavaScript-Objetos (19)

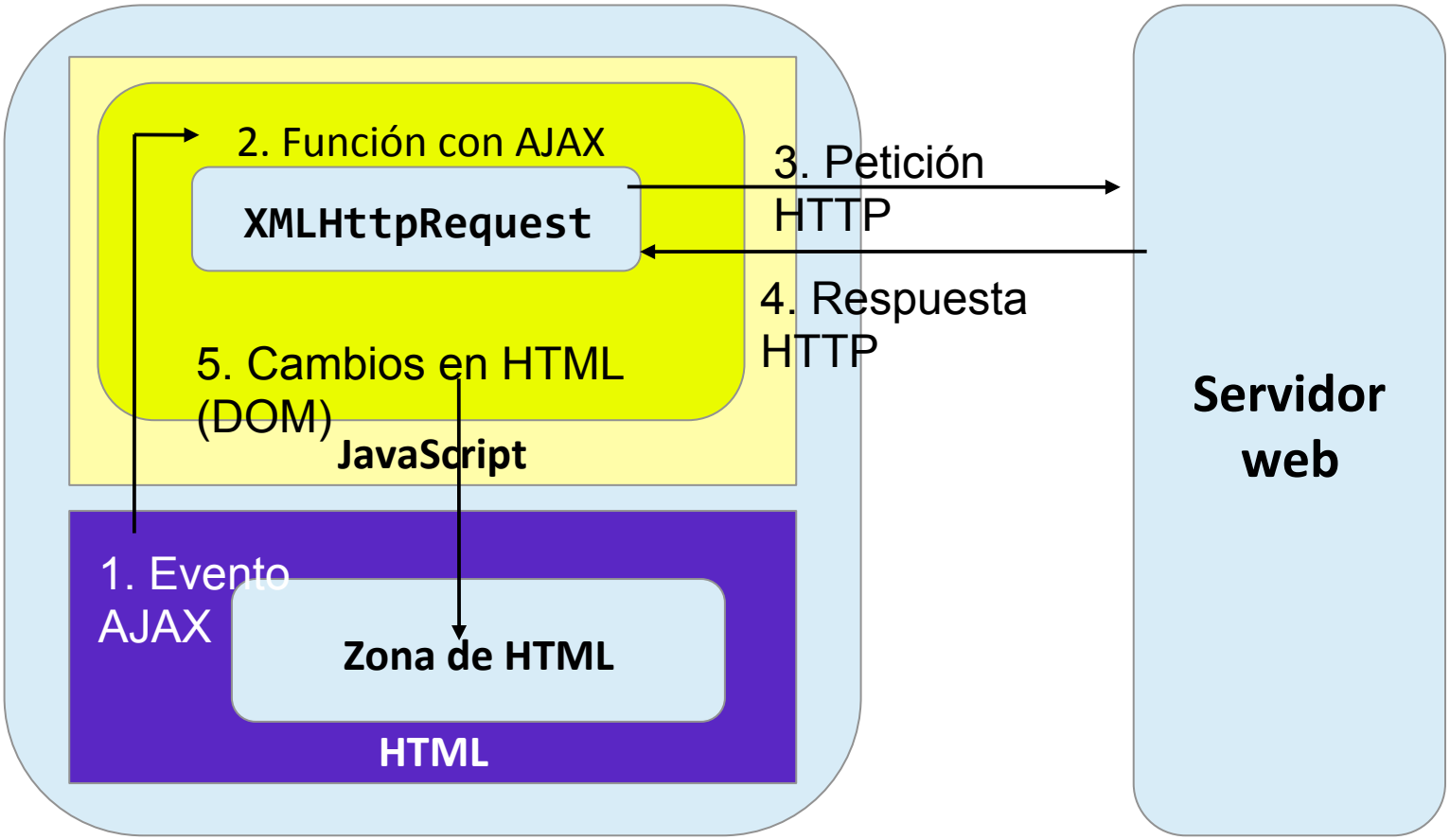
- Serialización de objetos
  - El proceso de serialización permite convertir el estado de un objeto a un *string* desde el cual se puede restaurar dicho estado.
    - `JSON.stringify()` realiza la conversión a *string*.
    - `JSON.parse()` restaura el estado del objeto.
    - Estas dos funciones utilizan el formato de intercambio de datos JSON (*JavaScript Object Notation*).


AJAX
Asynchronous Javascript And XML

AJAX (1)

- Nueva capa entre en cliente y el servidor
  - Se usa un objeto del DOM llamado `XMLHttpRequest` por medio del cual se realiza la comunicación con el servidor.
    - Permite obtener información del servidor y actualizar la interfaz de usuario
    - La respuesta que se obtiene del servidor puede estar en varios formatos: Texto plano, HTML, XML, JSON.

AJAX (2)


2. Función con AJAX
XMLHttpRequest
3. Petición HTTP
4. Respuesta HTTP
5. Cambios en HTML (DOM)
JavaScript
1. Evento AJAX
Zona de HTML
HTML
Servidor web

AJAX (3)

- Funcionamiento
  - La mayor parte del tráfico se produce al iniciar la comunicación.
  - Durante la interacción del usuario se envía la menor cantidad posible de información para agilizar la aplicación.
  - La página no espera a la respuesta del servidor:
    - Se generan transacciones asíncronas mientras la aplicación sigue funcionando.

AJAX (4)

- XMLHttpRequest
  - Métodos
    - `abort`
    - `getAllResponseHeaders()`
    - `getResponseHeader(etiqueta)`
    - `open(method, url, asinc, user, password)`:
      - Métodos: `GET`, `POST`, `HEAD`, `POST` o `DELETE`.
    - `send(contenido)`
    - `setRequestHeader(etiqueta, valor)`

AJAX (5)

- Propiedades:
  - `Onreadystatechange`
    - Asignación de manejador al evento de cambio de estado.
  - `readyState`: Estado de la conexión
    - `0`: no inicializada
    - `1`: cargando
    - `2`: cargada
    - `3`: interactiva
    - `4`: completada

AJAX (6)

- `responseText`: La respuesta en forma de texto plano.
- `responseXML`: La respuesta en formato XML.
- `status`: Código de respuesta del servidor.
  - Códigos HTTP. *200*: Transacción completada con éxito.
- `statusText`: Mensaje asociado a la respuesta.

AJAX (7)

- Ciclo de vida de una transferencia
  - Se crea el objeto `XMLHttpRequest`.
  - Se le asigna un manejador del evento de cambio de estado.
  - Se invoca la petición en función de la interacción del usuario.
  - Al recibir un cambio de estado, se gestiona la respuesta.
  - Si la respuesta es correcta, se obtienen los datos y se muestran al cliente.

AJAX (8)

- Creación del objeto `XMLHttpRequest`

```
if (window.XMLHttpRequest)
      objAJAX = new XMLHttpResques();
else
      if (window.ActiveXObject)
            objAJAX = new ActiveXObject("Microsoft.XMLHTTP");
      else
            alert("El navegador no soporta AJAX");
```

-

```
objAJAX.onreadystatechange = funcionAInvocar;
```

AJAX (9)

- Realizar la petición
  - Prepararla con el método open

```
objAJAX.open('GET', 'pruebaAJAX.php', true);
```

Si la petición no es asíncrona se espera por la respuesta.

  - Enviarla con el método send

```
objAJAX.send(null);
```

Si se utilizara el método POST, aquí irían los parámetros de la petición.

AJAX (10)

- Procesar la respuesta
  - El manejador se invoca al recibir un cambio de estado de la transacción.
    - `0 … 4`
  - Parte del código del manejador

```
if (this.readyState == 4 && this.status == 200) {
     // procesar la petición
}
else {
     // petición pendiente
}
```

AJAX (11)

- Respuesta del servidor
  - Propiedades `responseText`, `responseXML` y `responseJSON`.
- Ejemplo de AJAX
  - Mostrar la hora del servidor. La hora se obtiene mediante el *script* `hora.php`.

hora.php

```php
<?php echo date("H:i:s"); ?>
```

Fragmento de **hora.html**

```html
<p>
      <input id="id_hora" type="text" size="8">
      <input type="button" value= "Obtener hora" onclick="hora()">
</p>
```

AJAX (12)

```
function hora() {
      // crea el objeto XMLHttpRequest
      var xhrObject = new XMLHttpRequest();
      // prepara la petición
      xhrObject.open('GET', "hora.php", true);
      // método onreadystatechange
      xhrObject.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                  var objInput = document.getElementById("id_hora"
                  objInput.value = this.responseText;

            }
      }
      // envio de la petición
      xhrObject.send(null);
}
```

AJAX (13)

- Ejemplo AJAX
  - Al pulsar un botón solicita al servidor un documento XML con enlaces, `links.xml`, que se cargan en el documento HTML.

AJAX (14)

```
function requestLinks () {
	if (window.XMLHttpRequest) {
		// crea el objeto XMLHttpRequest
		var xhrObject = new window.XMLHttpRequest();
		// prepara la petición
		xhrObject.open('GET', 'links.xml', true);
		// método onreadystatechange
		xhrObject.onreadystatechange = function() {
			if (this.readyState == 4 && this.status == 200)
				var objDiv = document.getElementById('content');
				var listLinks = enlaces(this.responseXML);

				if (listLinks)
					objDiv.innerHTML = listLinks;
			}
		// envío de la petición
		xhrObject.send(null);
		}
	else
		alert("Error cargando los enlaces.");
}
```

AJAX (15)

```
function enlaces(xmlDoc) {
        if (xmlDoc != null) { // Lista de enlaces
                var linksTag = xmlDoc.getElementsByTagName('lista')[0].
                                        getElementsByTagName('link');
                var str = "";
                if (linksTag.length > 0) str = "<ul>";
                for ( var i = 0; i < linksTag.length; i++) {
                // título del link
                var titulo = linksTag[i].getElementsByTagName('titulo')[0].
                                childNodes[0].nodeValue;
                // Referencia del link
                var href = linksTag[i].getElementsByTagName('href')[0].
                        childNodes[0].nodeValue;
                // Añadir el link al contenedor ul
                str += "<li><a href=\"" + href + "\">" + titulo + "</li>";
                }
                if (str) str += "</ul>";
                return str;
        }
        return null;
}
```

Alternativas (1)

- Problemas de JavaScript
  - Ciertas carencias del lenguaje
  - Incompatibilidades entre navegadores (en teoría HTML 5 va a eliminar éstas).
  - Falta de productividad. El DOM es potente pero tedioso de usar.

Alternativas (2)

- Librerías de JavaScript (*frameworks*)
  - Aportan mayor productividad simplificando las tareas más comunes
    - Manipulación del DOM
    - AJAX
    - Incorporan *widgets* o componentes gráficos
  - Encapsulan las incompatibilidades entre navegadores
  - Inconveniente: código dependiente de APIs no estándar.

Alternativas (3)

- Algunos *framenworks*
  - Closure, jQuery, MooTools, Prototype, …
  - *Widgets*
    - Dojo, ExtJS, jQuery UI
  - Animaciones
    - Scriptaculous, jQuery UI

Alternativas (4)

- *Framenworks* avanzados
  - Persiguen aplicaciones web con interfaces similares a las aplicaciones de escritorio.
    - **GWT** (*Google Web Toolkit*). Se programa en Java y el resultado se traduce a JavaScript y CSS.
    - **Capuccino** (*Objective-J*)
    - **SproutCore** (*Apple*).