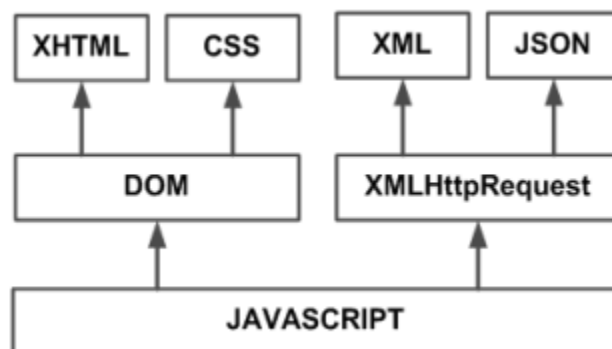




Asynchronous → Asíncrono

JavaScript Asíncrono + XML

Es la Unión de las Siguietes tecnologías:



Para más info, ir a Libros Web:

<http://librosweb.es/ajax/>


HOLA MUNDO CON AJAX:

Muestra en un alert el texto de un archivo.txt que se encuentra en el servidor.

Consta de 4 Pasos:

- (1) Instancia el Objeto XMLHttpRequest.
 - (2) Prepara la función de respuesta.
 - (3) Realiza la petición al servidor.
 - (4) Ejecuta la función de respuesta.
-

```
function descargaArchivo(){  
    //(1) Obtener la instancia del objeto XMLHttpRequest  
    // Navegadores buenos  
    if(window.XMLHttpRequest) {  
        petición_http = new XMLHttpRequest();  
    }  
    // Navegadores Malos ej. I.E.  
    else if(window.ActiveXObject) {  
        petición_http = new ActiveXObject("Microsoft.XMLHTTP");  
    }  
    //(2) Preparar la función de respuesta  
    petición_http.onreadystatechange = muestraContenido;  
    //(3) Realizar petición HTTP  
    petición_http.open('GET', 'http://localhost/holamundo.txt', true);  
    petición_http.send(null);  
    //(4) Se ejecuta automáticamente la función cuando se recibe la respuesta  
    del servidor  
    function muestraContenido() {  
        if(petición_http.readyState == 4) {  
            alert(petición_http.responseText);  
        }  
    }  
}
```



```

    }
    }
    }
    if(peticion_http.status == 200) {
        alert(peticion_http.responseText);
    }
}

window.onload = descargaArchivo;

```

HOLA MUNDO OPTIMIZADO:

```

var READY_STATE_UNINITIALIZED=0;
var READY_STATE_LOADING=1;
var READY_STATE_LOADED=2;
var READY_STATE_INTERACTIVE=3;
var READY_STATE_COMPLETE=4;
var peticion_http;

function cargaContenido(url, metodo, funcion) {
    peticion_http = inicializa_xhr();
    if(peticion_http) {
        peticion_http.onreadystatechange = funcion;
        peticion_http.open(metodo, url, true);
        peticion_http.send(null);
    }
}

```

```

function inicializa_xhr() {
    if(window.XMLHttpRequest) {
        return new XMLHttpRequest();
    }
    else if(window.ActiveXObject) {
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}

function muestraContenido() {
    if(peticion_http.readyState == READY_STATE_COMPLETE) {
        if(peticion_http.status == 200) {
            alert(peticion_http.responseText);
        }
    }
}

function descargaArchivo() {
    cargaContenido("http://localhost/holamundo.txt", "GET", muestraContenido);
}

window.onload = descargaArchivo;

```

PROPIEDADES OBJETO XMLHttpRequest:

readyState: valor numérico entero, que almacena el estado de la petición.

responseText: el contenido de la respuesta del servidor en forma de cadena de texto.

responseXML: contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM.

status: el código de estado HTTP devuelto por el servidor (200 para respuesta correcta, 404 "No encontrado", 500 para error del servidor, etc.).

statusText: el código de estado HTTP devuelto por el servidor en forma de cadena de texto ("OK", "Not Found", "Internal Server Error", etc.).

Valores para readyState:

0: No inicializado (objeto creado, pero no se ha invocado el método open).

1: Cargando (objeto creado, pero no se ha invocado el método send).

2: Cargado (se ha invocado el método send, pero el servidor aún no ha respondido).

3: Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad responseText).

4: Completo (se han recibido todos los datos de la respuesta del servidor).

MÉTODOS OBJETO XMLHttpRequest:

abort(): detiene la petición actual.

getAllResponseHeaders(): devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor.

getResponseHeader("cabecera"): devuelve una cadena de texto con el contenido de la cabecera solicitada.

onreadystatechange: maneja los eventos que se producen. Se invoca cada vez que se produce un cambio en el estado de la petición HTTP.

open("método", "url"): establece los parámetros de la petición que se realiza al servidor.

Los parámetros necesarios son: el método Http y la url destino (se puede indicar de forma relativa o absoluta).

También tiene otros parámetros opcionales:

open(string método, string URL[, Boolean asíncrono, string usuario, string password]);

send(contenido): realiza la petición http al servidor.

setRequestHeader("cabecera", "valor"): Permite establecer cabeceras personalizadas en la petición HTTP. Se debe invocar el método open() antes que setRequestHeader().

HOLA MUNDO 3 SUPER OPTIMIZADO:

Encapsula todo en un objeto para hacerlo re-utilizable.

Adicionalmente, si la url no es válida o el servidor no responde, se muestra el status del error del servidor.


```
var net = new Object();

net.READY_STATE_UNINITIALIZED=0;
net.READY_STATE_LOADING=1;
net.READY_STATE_LOADED=2;
net.READY_STATE_INTERACTIVE=3;
net.READY_STATE_COMPLETE=4;

// Constructor
net.CargadorContenidos = function(url, funcion, funcionError) {
    this.url = url;
    this.req = null;
    this.onload = funcion;
    this.onerror = (funcionError) ? funcionError : this.defaultError;
    this.cargaContenidoXML(url);
}

net.CargadorContenidos.prototype = {
    cargaContenidoXML: function(url) {
        if(window.XMLHttpRequest) {
            this.req = new XMLHttpRequest();
        }
        else if(window.ActiveXObject) {
            this.req = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
}
```





```
if(this.req) {  
    try {  
        var loader = this;  
        this.req.onreadystatechange = function() {  
            loader.onReadyState.call(loader);  
        }  
        this.req.open('GET', url, true);  
        this.req.send(null);  
    } catch(err) {  
        this.onerror.call(this);  
    }  
}  
},  
  
onReadyState: function() {  
    var req = this.req;  
    var ready = req.readyState;  
    if(ready == net.READY_STATE_COMPLETE) {  
        var httpStatus = req.status;  
        if(httpStatus == 200 || httpStatus == 0) {  
            this.onload.call(this);  
        }  
        else {  
            this.onerror.call(this);  
        }  
    }  
},  
},
```



```
↑
    defaultError: function() {
        alert("Se ha producido un error al obtener los datos"
            + "\n\nreadyState:" + this.req.readyState
            + "\nstatus: " + this.req.status
            + "\nheaders: " + this.req.getAllResponseHeaders());
    }
}
```

```
function muestraContenido() {
    alert(this.req.responseText);
}
```

```
function cargaContenidos() {
    var cargador=new net.CargadorContenidos("http://localhost/holamundo.txt",
        muestraContenido);
}
```

```
window.onload = cargaContenidos;
```

//esta es una optimización del hola_mundo2, adicionalmente, muestra estatus de error del servidor, si llega a ocurrir.

Nota: ver capítulo 7.4 Ajax – Libros Web

EJERCICIO NOTICIAS:

La pág. Web, tiene una zona llamada ticker en la que se muestran noticias generadas por el servidor.

El código JavaScript hace lo siguiente:

- 1) De forma periódica cada cierto tiempo (ej. Cada 3 segundos), se realiza una petición al servidor mediante AJAX y se muestra el contenido de la respuesta en la zona reservada para las noticias.
- 2) Además del contenido enviado por el servidor, se debe mostrar la hora en la que se ha recibido la respuesta.
- 3) Cuando se pulsa el botón “Detener”, se detienen las peticiones al servidor. Si se vuelve a pulsar, se reanudan.
- 4) Los botones “Anterior” y “Siguiente”, detienen las peticiones al servidor y permiten mostrar los contenidos anteriores y posteriores al que se muestra en ese momento.
- 5) Cuando se recibe una respuesta del servidor, se resalta visualmente la zona ticker.

Archivo generaContenidos.php

```
<?php
    //Frases de contenido de las que se selecciona aleatoriamente una de ellas
    $frases=array("Primera Frase", "Segunda Frase", "Ultima Frase");

    // Generar un número aleatorio comprendido entre 0 y el número total de
    frases disponibles

    srand((double)microtime()*1000000);

    $numeroAleatorio = rand(0, count($frases)-1);

    // La respuesta de este script consiste en una frase seleccionada
    aleatoriamente

    echo $frases[$numeroAleatorio];

?>
```

Archivo noticias.html

```
<!DOCTYPE html>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<title>Ejercicio 12 - Actualización periódica de contenidos</title>

<style type="text/css">
```

```
body { margin: 0; }

#contenidos { padding: 1em; }

#ticker {
    height: 20px;
    padding: .3em;
    border-bottom: 1px solid #CCC;
    background: #FAFAFA;
    font-family: Arial, Helvetica, sans-serif;
}

#ticker strong { margin-right: 1em; }

#acciones {
    position: absolute;
    top: 3px;
    right: 3px;
}
```

</style>

<script type="text/javascript">

//La variable intervalo se usa para detener las peticiones al servidor.

var intervalo = null;

//El array catalogo_noticias, se usa para guardar en el lado del cliente, las noticias que envía el servidor. Se usa para añadir funcionalidad de anterior y siguiente

var catalogo_noticias = [];

//indice_noticia, sirve para posicionarnos en el array catalogo_noticias

var indice_noticia=null;

//net es el objeto que nos permite conectarnos al servidor

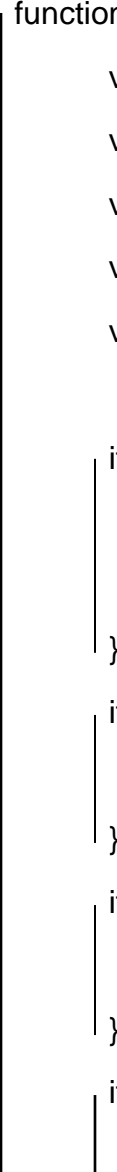
var net = new Object();

...

//igual a hola_mundo 3, se cambia la función muestraContenido y se adapta la función cargaContenidos

```
function obtenerElementoId(identificador){  
    //obtiene un elemento por id  
    return document.getElementById(identificador);  
}
```

```
function mostrarHora(){  
    var fechaHora=new Date();  
    var horas=fechaHora.getHours();  
    var minutos=fechaHora.getMinutes();  
    var segundos=fechaHora.getSeconds();  
    var sufijo="am";  
  
    if(horas>12){  
        horas=horas-12;  
        sufijo="pm";  
    }  
  
    if(horas<10){  
        horas="0"+horas;  
    }  
  
    if(minutos<10){  
        minutos="0"+minutos;  
    }  
  
    if(segundos<10){  
        segundos="0"+segundos;  
    }  
}
```



```

    ↑
    ↑
    }
    return horas+":"+minutos+":"+segundos+sufijo;
}

```

```

function muestraContenido(info) {
    //obtiene el lugar donde se actualizan las noticias
    var zonaNoticia=obtenerElementoId("ticker");

    // Borrar datos anteriores
    zonaNoticia.innerHTML = "";

    //agrega la info a mostrar
    zonaNoticia.innerHTML=info;
    //resalta la zona donde se agrega la info
    resaltaZona(zonaNoticia,'#00BFFF');
    //devuelve al color original a la zona resaltada
    setTimeout(function(){resaltaZona(zonaNoticia,'#FAFAFA');},300);
}

```

```

function resaltaZona(elemento,nuevo_color){
    //cambia el color de fondo de un elemento(tag)
    elemento.style.backgroundColor=nuevo_color;
}

```

```

function recibeServidor(){
    //la noticia devuelta por el server
    var noticia=this.req.responseText;
    ↓

```

```

↑
//el tiempo en que el servidor entrega la noticia en el cliente
var tiempo=mostrarHora();

//arma la info(la noticia y el tiempo de la noticia)
var info="<strong>"+tiempo+"</strong> "+noticia;

//Va armando el catálogo de noticias del cliente, agregándole
elementos
catalogo_noticias.push(info);

//llama a la función que muestra en la pag.
muestraContenido(info);
}

```

```

function cargaContenidos() {
    var cargador=new net.CargadorContenidos(
        "http://localhost/generaContenidos.php", recibeServidor);
}

```

```

function detener_iniciar(){
    if(this.value=="Iniciar"){
        intervalo=setInterval(cargaContenidos,3000);
        this.value="Detener";
        indice_noticia=null;
    }else{
        clearInterval(intervalo);
        this.value="Iniciar";
    }
}
}

```

```
function anterior_siguiente(){  
    //detenemos las peticiones al servidor  
    clearInterval(intervalo);  
    obtenerElementoId("detener").value="Iniciar";  
    //sacamos el tamaño del catálogo de noticias que tiene el cliente  
    var sizeCatalogo=catalogo_noticias.length-1;  
  
    if(indice_noticia==null){  
        //al índice de noticias, le asignamos el tamaño del catálogo de  
        //noticias que tiene el cliente  
        indice_noticia=sizeCatalogo;  
    }  
  
    if(this.id=="siguiente"){  
        if(indice_noticia<sizeCatalogo){  
            //si el índice de noticias es menor que el tamaño del  
            //catálogo, significa que hay más noticias siguientes  
            indice_noticia++;  
        }  
    }  
  
    if(this.id=="anterior"){  
        if(indice_noticia>0){  
            //si el índice de noticias es mayor que la posición del primer  
            //elemento del catálogo(0), significa que hay más noticias  
            //anteriores  
            indice_noticia--;  
        }  
    }  
  
    //sacamos la noticia del catálogo del cliente
```

```

    ↑
    var info=catalogo_noticias[indice_noticia];
    //llama a la función que muestra en la pág.
    muestraContenido(info);
  }

```

```

window.onload =function(){
    //refresca cada 3 seg.
    intervalo=setInterval(cargaContenidos,3000);
    //para el botón detener-iniciar
    obtenerElementoId("detener").onclick=detener_iniciar;
    //para el botón siguiente
    obtenerElementoId("siguiente").onclick=anterior_siguiente;
    //para el botón anterior
    obtenerElementoId("anterior").onclick=anterior_siguiente;
}

```

```

</script>

```

```

</head>

```

```

<body>

```

```

<div id="ticker"></div>

```

```

<div id="acciones">

```

```

  <input type="button" id="detener" value="Detener"/>

```

```

  <input type="button" id="anterior" value="&laquo; Anterior" />

```

```

  <input type="button" id="siguiente" value="Siguiente &raquo;" />

```

```

</div>

```

```

</body>

```

```

</html>

```

Nota: esta es mi solución al ejercicio 12 de Libros web.

ENVIO DE PARÁMETROS AL SERVIDOR:

GET: con este método los parámetros se concatenan a la URL accedida.

Se utiliza cuando se accede a un recurso que depende de la info proporcionada por el usuario.

Ej:

http://localhost/aplicacion?parametro1=valor1¶metro2=valor2¶metro3=valor3

GET, tiene un límite en la cantidad de datos para enviar, si se pasa el límite de 512 bytes, el servidor devuelve el error 414 y mensaje: "Request-URI Too Long" (La URI de la petición es demasiado larga).

POST: los parámetros se envían en el cuerpo de la petición. Este método se utiliza en operaciones que crean, borran o actualizan información.

Ej: Se envía el siguiente formulario:

Un formulario web con tres campos de texto y un botón. Los campos están etiquetados como 'Fecha de nacimiento:', 'Codigo postal:' y 'Telefono:'. El botón está etiquetado como 'Validar datos'.

```
<form>
```

```
  <label for="fecha_nacimiento">Fecha de nacimiento:</label>
```

```
  <input type="text" id="fecha_nacimiento" name="fecha_nacimiento"/><br/>
```

```
  <label for="codigo_postal">Codigo postal:</label>
```

```
  <input type="text" id="codigo_postal" name="codigo_postal"/><br/>
```

```
  <label for="telefono">Telefono:</label>
```

```
  <input type="text" id="telefono" name="telefono"/><br/>
```

```
  <input type="button" value="Validar datos"/>
```

```
</form>
```

```
<div id="respuesta"></div>
```

Codigo JavaScript:

```
var READY_STATE_COMPLETE=4;
```

```
var petición_http=null;
```

```
function inicializa_xhr(){  
    if(window.XMLHttpRequest){  
        return new XMLHttpRequest();  
    }  
    else if(window.ActiveXObject) {  
        return new ActiveXObject("Microsoft.XMLHTTP");  
    }  
}
```

```
function crea_query_string() {  
    var fecha = document.getElementById("fecha_nacimiento");  
    var cp = document.getElementById("codigo_postal");  
    var telefono = document.getElementById("telefono");  
  
    return "fecha_nacimiento=" + encodeURIComponent(fecha.value) +  
        "&codigo_postal=" + encodeURIComponent(cp.value) +  
        "&telefono=" + encodeURIComponent(telefono.value) +  
        "&nocache=" + Math.random();  
}
```

```
function valida() {  
    petición_http = inicializa_xhr();  
    if(petición_http) {  
        petición_http.onreadystatechange = procesaRespuesta;
```



```

↑
↑
    petición_http.open("POST", "http://localhost/validaDatos.php", true);

    petición_http.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

    var query_string = crea_query_string();

    petición_http.send(query_string);
}
}

```

```

function procesaRespuesta() {
    if(petición_http.readyState == READY_STATE_COMPLETE) {
        if(petición_http.status == 200) {
            document.getElementById("respuesta").innerHTML=
            petición_http.responseText;
        }
    }
}
}

```

Content-Type: cabecera necesaria para enviar parámetros por POST.

crea_query_string(): cadena construida manualmente, con variables y sus valores concatenados con &, esta cadena se conoce como query_string, y son los parámetros que se envían.

send(parámetros): envía los parámetros al servidor.

encodeURIComponent(): reemplaza los caracteres que no se pueden utilizar en la URL, por su representación hexadecimal.

- _ . ! ~ * ' () → no se modifican

Espacio en blanco → %20

& → %26

Ej: var cadena="cadena & caracteres problemáticos /:=";

var cadena_segura=encodeURIComponent(cadena);

```
//cadena_segura="cadena%20%26%20caracteres%20problem%C3%A1ticos%20%2F%20%3A%20%3D";
```

decodeURIComponent(): contraria a encode.

encodeURIComponent(): codifica la URL completa, no codifica los caracteres ; / ? : @ & = + \$, #:

decodeURI(): decodifica.

nocache: contiene un # aleatorio creado mediante el método Math.random().

Se usa para evitar problemas de cache con los navegadores. Se le obliga al navegador a relizar siempre la petición al servidor y no usar cache, ya que este parámetro siempre tendrá un valor diferente.

Nota: Las validaciones con AJAX, son validaciones complejas, que requieran el uso de bases de datos, como:

- Comprobar que un nombre de usuario no este previamente registrado.
- Comprobar que la localidad corresponda con el cód. Postal, etc.

HOLA MUNDO 3 CON POST:

Para que sirva con el método POST, se hacen los siguientes cambios:

// Constructor

```
net.CargadorContenidos = function(url, funcion, funcionError, metodo,  
parametros, contentType) {  
    ...  
    this.cargaContenidoXML(url, metodo, parametros, contentType);  
}
```

```
net.CargadorContenidos.prototype = {  
    cargaContenidoXML: function(url, metodo, parametros, contentType) {  
        ...  
        this.req.open(metodo, url, true);
```



```

    ↑
    |
    |   if(contentType) {
    |       this.req.setRequestHeader("Content-Type", contentType);
    |   }
    |
    |   this.req.send(parametros);
    |
    |   ...
    |
    },

```

ENVIO DE PARAMETROS XML:

Volviendo al ejemplo del formulario enviado por el método POST, esta vez, en vez de enviar un query_string, vamos a crear el xml:

```

function crea_xml() {
    var fecha = document.getElementById("fecha_nacimiento");
    var cp = document.getElementById("codigo_postal");
    var telefono = document.getElementById("telefono");

    var xml = "<parametros>";
    xml=xml+ "<fecha_nacimiento>" + fecha.value + "<\ /fecha_nacimiento>";
    xml = xml + "<codigo_postal>" + cp.value + "<\ /codigo_postal>";
    xml = xml + "<telefono>" + telefono.value + "<\ /telefono>";
    xml = xml + "<\ /parametros>";

    return xml;
}

```

Nota: Esto crea una cadena de texto que representa un documento XML.

Se emplea el carácter \ en el cierre de todas las etiquetas XML, para que valide siguiendo el estándar XHTML de forma estricta.

También se puede recibir en el cliente, respuestas del servidor de tipo XML con la propiedad: **responseXML**.

Ej: El servidor envía este documento XML:

```
<respuesta>
  <mensaje>...</mensaje>
  <parametros>
    <telefono>...</telefono>
    <codigo_postal>...</codigo_postal>
    <fecha_nacimiento>...</fecha_nacimiento>
  </parametros>
</respuesta>
```

Para recibir esta respuesta del servidor se usa:

```
function procesaRespuesta() {
  if(peticion_http.readyState == READY_STATE_COMPLETE) {
    if(peticion_http.status == 200) {
      var documento_xml = peticion_http.responseXML;
      var root=documento_xml .getElementsByTagName
        ("respuesta")[0];

      var mensajes = root.getElementsByTagName("mensaje")[0];
      var mensaje = mensajes.firstChild.nodeValue;


      var parametros= root.getElementsByTagName
        ("parametros")[0];

      var telefono=parametros.getElementsByTagName
        ("telefono")[0].firstChild.nodeValue;

      var fecha_nacimiento=parametros.getElementsByTagName
        ("fecha_nacimiento")[0].firstChild.nodeValue;

      var codigo_postal=parametros.getElementsByTagName
        ("codigo_postal")[0].firstChild.nodeValue;
```





```
document.getElementById("respuesta").innerHTML=mensaje+"<br/>" + "Fecha
nacimiento = " + fecha_nacimiento + "<br/>" + "Codigo postal = " + codigo_postal +
"<br/>" + "Telefono = " + telefono;
}
}
}
```

Nota: El ejercicio 13: Nombre Usuario, valida con POST, si el nombre de usuario está disponible (la validación es muy simple, el servidor contesta aleatoriamente Si o No).

El ejercicio 14: Nombre UsuarioXML, realiza el mismo procedimiento del ejercicio 13, pero adicionalmente, si el nombre de usuario no está disponible, recibe del servidor, un documento XML, unas sugerencias de nombres de usuarios disponibles.

/*La respuesta del servidor es:

```
<respuesta>
    <disponible>no</disponible>
    <alternativas>
        <login>...</login>
        <login>...</login>
        ...
        <login>...</login>
    </alternativas>
</respuesta>
*/
```

PARÁMETROS JSON:

En formato JSON, la respuesta que envía el servidor es más concisa:

```
{
    mensaje: "...",
    parametros:{
        telefono: "...",
        codigo_postal: "...",
        fecha_nacimiento: "..."
    }
}
```

Con este formato, la función que procesa la respuesta del servidor quedaría así:

```
function procesaRespuesta() {
    if(http_request.readyState == READY_STATE_COMPLETE) {
        if(http_request.status == 200) {
            var respuesta_json = http_request.responseText;
            var objeto_json = eval("("+respuesta_json+")");
            var mensaje = objeto_json.mensaje;
            var telefono = objeto_json.parametros.telefono;
            var fecha_nacimiento=objeto_json.parametros
                .fecha_nacimiento;
            var codigo_postal=objeto_json.parametros
                .codigo_postal;
            document.getElementById("respuesta").innerHTML=
            mensaje+"<br>"+"Fecha nacimiento =
            " +fecha_nacimiento + "<br>" + "Codigo postal =
            " + codigo_postal + "<br>" + "Telefono = " + telefono;
```


↑
↑
↑ }

}

eval(): como el servidor devuelve una respuesta en forma de cadena de texto, se usa la función eval() para transformarla en objeto JSON.

Nota: el ejercicio 16: **CargarListasDesplegable**, como su nombre lo indica, carga con AJAX, desde un archivo php, en el servidor, una lista de datos (las Provincias de España), y según se selecciona, hace otra petición al servidor, para cargar los datos de los Municipios, de la provincia seleccionada.

- Selecciona - ← Selecciona una provincia

(el usuario selecciona una provincia)

Álava ← Selecciona una provincia

(automáticamente se realiza la petición al servidor y se obtiene la lista de municipios de la provincia)

```

<municipios>
  <municipio>
    <codigo>0014</codigo>
    <nombre>Alegria-Dulantzi</nombre>
  </municipio>
  <municipio>
    <codigo>0029</codigo>
    <nombre>Amurrio</nombre>
  </municipio>
  ...
</municipios>

```

Álava Alegria-Dulantzi 51 municipios

Amurrio
Añana
Aramaio
Armiñón
...

Ejercicio 18: **Teclado Virtual**, carga un teclado en pantalla, en el idioma seleccionado (Español, Alemán, Ruso, Griego y Checo), para ser usado con el mouse.



Ejercicio 19: **Autocompletar**, carga en un div, una serie de sugerencias, devueltas por el servidor, según a lo que se haya escrito en un input. Pero con JQuery, se puede hacer de una forma más fácil y simplificada.

Ejercicio 20: **Monitor Servidores Remotos**, es una consola básica de monitorización de equipos de red. Los servidores que se monitorizan, pertenecen a dominios conocidos de Internet (ej www.google.com) y por tanto, externos a la aplicación JavaScript.

La aplicación hace un ping, a través de la red mediante AJAX para comprobar los equipos que se quieren monitorizar.

Ejercicio 21: **Lector RSS**, es un Lector de Noticias RSS (Really Simple Syndication, un formato XML para syndicar o compartir contenido en la web), en donde se ingresa una URL de una pág. de noticias (ej. <http://www.eltiempo.com/>).

Mediante Ajax, encuentra el canal RSS asociado a dicha dirección, y nos muestra en un menú lateral, los titulares de las noticias más importantes.

Cuando se hace clic en uno de estos titulares, carga en un div central, una pequeña descripción de la noticia, así como nos da la posibilidad de ingresar a los enlaces donde encontramos toda la información de esas noticias.



Este mini tuto, es para la API versión 3 de Google Maps.

Para más info, consultar el API, y ver ejemplos de uso de muchas funciones de los mapas: <https://developers.google.com/maps/documentation/javascript/tutorial>

Para Usar Google Maps:

En la hoja de estilos .CSS, definimos el tamaño del div, donde se mostrara el mapa.

- 1) Llamamos al script de google maps con:

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

```
src="https://maps.googleapis.com/maps/api/js?sensor=false"></script>
```

Se usa para utilizar la api, el parámetro sensor, puede ser true o false, depende de si se usa un sensor (ejemplo localizador gps).

- 2) Cuando la pág. termine de cargar (window.onload), llamamos la función encargada de crear el mapa, que en este caso se llama initialize().

El "window.onload..." hace lo mismo que la siguiente línea:

```
google.maps.event.addDomListener(window, 'load', initialize);
```

- 3) Dentro de la function initialize(), definimos las variables de: **latitud, longitud y zoom**, si queremos que nuestro mapa, recién se carga, aparezca en un lugar en específico. El zoom, es para el acercamiento que queremos que tenga el mapa recién se carga.

También definimos la variable **mapOptions**, en donde podemos especificar las siguientes opciones:

Opciones del Mapa:

center→ es donde estará centrado el mapa.

zoom→ acercamiento o alejamiento, 0 es el mapa de la tierra

disableDefaultUI→ si se pone igual a true, se eliminan los controles de la interfaz de usuario(UI).

mapTypeId→ Se refiere al tipo de mapa.

Admite los siguientes valores:

MapTypeId.ROADMAP: muestra la vista de mapa de carretera predeterminada.

MapTypeId.SATELLITE: muestra imágenes de satélite de Google Earth.

MapTypeId.HYBRID: muestra una mezcla de vistas normales y de satélite.

MapTypeId.TERRAIN: muestra un mapa físico basado en información del relieve

Ejemplo de uso:

mapTypeId: google.maps.MapTypeId.ROADMAP

Nota: existen muchas otras opciones, para más info. ver el API.

-
- 4) Se crea un nuevo mapa, pasándole como parámetro el elemento que va a contener el mapa(div), y las opciones con las que va a ser creado.

```
var map=new google.maps.Map(document.getElementById("mapa"),
mapOptions);
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
```

```
<style type="text/css">
```

```
html { height: 100% }
```

```
body { height: 100%; margin: 0; padding: 0 }
```

```
#mapa { width: 500px; height: 400px }
```

```
</style>
```

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

```
src="https://maps.googleapis.com/maps/api/js?sensor=false">
```

```
</script>
```

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

```
function initialize(){

    //variables para ubicar un sitio en el mapa

    var latitud = 4.728702;

    var longitud = -74.115899;

    var zoom = 20;


    var mapOptions = {

        center: new google.maps.LatLng(latitud, longitud),

        zoom: zoom,

        //disableDefaultUI: true,

        mapTypeId: google.maps.MapTypeId.SATELLITE

    };


    //se crea un nuevo mapa, pasándole como parámetro el
    //elemento que va a contener el mapa(div)

    var map=new google.maps.Map(document.getElementById("mapa"),
    mapOptions);


    //se añade un manejador de eventos para que cuando se
    //haga click en el mapa, se añada una marca:

    google.maps.event.addListener(map, 'click', function(event) {

        //event.latLng→ obtiene las coordenadas latitud y
        //longitud de donde se hizo clic

        //ponerMarca(event.latLng,map);

        ponerMarcaPersonal(event.latLng,map);

    });

}
```

```

function ponerMarca(sitio,mapa) {
    var marca = new google.maps.Marker({
        position: sitio,
        map: mapa,
        title: "Marca Simple"
    });

    //centra el mapa donde se hizo la marca

    mapa.setCenter(sitio);

    //añadir un evento para que cuando se haga clic se borre la marca:

    //google.maps.event.addListener(marca, 'click',borrarMarca);

    //mostrar una ventana con información de la marca cuando se hace clic

    google.maps.event.addListener(marca, 'click',function(){
        mostrarInfoMarca(mapa,marca);
    });
}

```

//Para crear una marca personalizada:

```

function ponerMarcaPersonal(sitio,mapa){

    // el tamaño de las marcas se expresa como X,Y

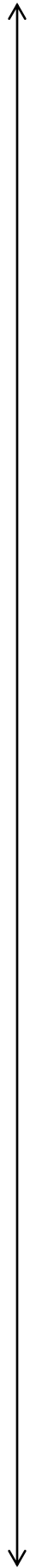
    //el origen de la imagen (0,0), es la parte superior izquierda de la imagen

    // Orígenes, posición del enlace (link, donde se hace clic en la marca) y coordenadas de la marca

    //incrementan en X a la derecha y Y hacia abajo

```



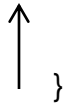


```
var imagen={  
    url: 'narutoIcono.jpg',  
    //Esta marca es de 20 pixeles de ancho y 32 pixeles de  
    alto  
    size: new google.maps.Size(20, 32),  
    // Origen de la imagen (0,0)  
    origin: new google.maps.Point(0,0),  
    // el enlace de la imagen es en (0,32)  
    anchor: new google.maps.Point(0, 32)  
};
```

```
//Shapes define la región clickeable del icono  
  
// el tipo define un área HTML 'poly' que traza un polígono  
como series de puntos (X,Y)  
  
//La coordenada final, se conecta a la primera coordenada,  
cerrando el polígono
```

```
var shape = {  
    coord: [1, 1, 1, 32, 18, 32, 18 , 1],  
    type: 'poly'  
};
```

```
var marca=new google.maps.Marker({  
    position: sitio,  
    map: mapa,  
    icon: imagen,  
    shape: shape,  
    title: "Marca Personalizada",  
});
```



```
}  
  
function borrarMarca(){  
    //se usa el parámetro null para borrarlo del mapa, si en vez de  
    //null, se le pasa el mapa, se dibuja la marca  
    this.setMap(null);  
}
```

```
function posicionMarca(marca){  
    //se obtiene la posición de la marca  
    var posicion_marca=marca.getPosition();  
    //Con la posicion, se sacan Latitud y Longitud  
    var latitud_marca=posicion_marca.lat();  
    var longitud_marca=posicion_marca.lng();  
  
    var mensaje="Latitud= "+latitud_marca+"<br/>Longitud=  
    "+longitud_marca;  
    return mensaje;  
}
```

```
function mostrarInfoMarca(mapa,marca){  
    var informacion=posicionMarca(marca);  
    //crea el objeto de la ventana  
    var infoVentana = new google.maps.InfoWindow({  
        content: informacion,  
        //maxWidth: 300,  
        //maxHeight: 300  
    });
```




```

    ↑
    //abre la ventana
    infoVentana.open(mapa,marca);
  }

  window.onload = function() {
    //se carga el mapa
    initialize();
  }

</script>
</head>
<body>
  <div id="mapa"/>
</body>
</html>

```

Ejercicio 23: **Google Maps Muestra Clima**, es una aplicación que nos muestra la información meteorológica de un lugar en específico (España).

Los puntos (latitud y longitud) de los sitios donde se mostraran iconos de lluvia, sol, nublado, etc., se encuentran en el archivo previsionMetereologica.php, en un array.

Este script php, determina de forma aleatoria para cada punto la condición del clima ("lluvia", "soleado", "nublado", "tormentas", "nieve").

Los iconos del sol, etc. En realidad son marcas personalizadas en el mapa de google.



```

<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
    <style type="text/css">
      html { height: 100% }
      body { height: 100%; margin: 0; padding: 0 }
      #mapa { width: 600px; height: 600px }
    </style>

    <script type="text/javascript"
      src="https://maps.googleapis.com/maps/api/js?sensor=false">
    </script>
    <script type="text/javascript">

      //Variable para el mapa
      var map=null;

      var todos_marcadores=[];

      var net = new Object();

      net.READY_STATE_UNINITIALIZED=0;
      net.READY_STATE_LOADING=1;
      net.READY_STATE_LOADED=2;
      net.READY_STATE_INTERACTIVE=3;
      net.READY_STATE_COMPLETE=4;

      // Constructor
      net.CargadorContenidos = function(url, funcion, funcionError,
      metodo, parametros, contentType, asincrono) {
        this.url = url;
        this.req = null;
        this.onload = funcion;
        this.onerror = (funcionError) ? funcionError : this.defaultError;
        this.cargaContenidoXML(url, metodo, parametros,
        contentType, asincrono);
      }

      net.CargadorContenidos.prototype = {
        cargaContenidoXML: function(url, metodo, parametros,
        contentType, asincrono) {
          if(window.XMLHttpRequest) {
            this.req = new XMLHttpRequest();
          }
          else if(window.ActiveXObject) {
            this.req = new ActiveXObject("Microsoft.XMLHTTP");
          }
        }
      }
    </script>
  </head>
  <body>
    <div id="mapa">
      <img alt="Mapa de Google Maps" data-bbox="192 89 866 931"/>
    </div>
  </body>
</html>

```

```

    ↑
    ↑
    ↑
    }

    if(this.req) {
        try {
            var loader = this;
            this.req.onreadystatechange = function() {
                loader.onReadyState.call(loader);
            }
            this.req.open(metodo, url,
                (asincrono===false)?false:true);
            if(contentType) {
                this.req.setRequestHeader(
                    "Content-Type", contentType);
            }
            this.req.send(parametros);
        } catch(err) {
            this.onerror.call(this);
        }
    }
},

onReadyState: function() {
    var req = this.req;
    var ready = req.readyState;
    if(ready == net.READY_STATE_COMPLETE) {
        var httpStatus = req.status;
        if(httpStatus == 200 || httpStatus == 0) {
            this.onload.call(this);
        }
        else {
            this.onerror.call(this);
        }
    }
},

defaultError: function() {
    alert("Se ha producido un error al obtener los datos"
        + "\n\nreadyState:" + this.req.readyState
        + "\nstatus: " + this.req.status
        + "\nheaders: " + this.req.getAllResponseHeaders());
}
}

function initialize(){
    //variables para ubicar un sitio en el mapa
    var latitud = 40.41558722527384;
    var longitud = -3.6968994140625;
    ↓

```

```

    var zoom = 6;

    var mapOptions = {
        center:      new      google.maps.LatLng(latitud,
        longitud),
        zoom: zoom,
        disableDefaultUI: true,
        mapTypeId:
        google.maps.MapTypeId.SATELLITE
    };

    map = new google.maps.Map
        (document.getElementById("mapa"),
        mapOptions);
    }

function descargaClima(){
    var url='http://localhost/previsionMeteorologica.php
        ?nocache='+Math.random();
    var carga=new net.CargadorContenidos
        (url,recibe_datos,null,"GET");
    }

function recibe_datos(){
    //Se Recibe la respuesta del servidor con:
    //this.req.responseText -->si es Texto
    var respuesta_json=this.req.responseText;

    var clima=eval("(" +respuesta_json+"");

    ponerMarcaPersonal(clima);
}

function ponerMarcaPersonal(locations) {
    //este método con el párametro null, borra las marcas
    anteriores, y nos sirve para evitar que se sobrepongan
    imágenes
    dibujarMarcas(null,todos_marcadores);

    // Crea las marcas y las almacena en el array todas las
    marcas, para luego dibujarlas en el mapa
    todos_marcadores=[];

```

```

    var shape = {
        coord: [1, 1, 1, 32, 18, 32, 18, 1],
        type: 'poly'
    };
    for (var i=0;i<locations.length;i++){

        var punto=locations[i];
        var prediccion=punto.prediccion;
        var punto_lat=punto.latlon[0];
        var punto_lon=punto.latlon[1];

        var imagen = {
            url: 'imagenes/'+prediccion+'.png',
            //Esta marca es de 32 pixeles de ancho y
            // 32 pixeles de alto
            size: new google.maps.Size(32, 32),
            // Origen de la imagen (0,0)
            origin: new google.maps.Point(0,0),
            // el enlace de la imagen es en (0,32)
            anchor: new google.maps.Point(0, 32)
        };

        var marcaLatLng = new google.maps.LatLng
            (punto_lat, punto_lon);
        var marca = new google.maps.Marker({
            position: marcaLatLng,
            icon: imagen,
            shape: shape,
            title: prediccion
        });

        todos_marcadores.push(marca);
    }
    dibujarMarcas(map,todos_marcadores);
}

function dibujarMarcas(mapa,marcadores) {
    for (var i=0;i<marcadores.length;i++){
        marcadores[i].setMap(mapa);
    }
}

window.onload = function() {
    //se carga el mapa
    initialize();
}

```

```

    }
    ↑
    //se piden los datos al server
    setInterval(descargaClima,3000);
  }
</script>
</head>
<body>
  <div id="mapa"/>
</body>
</html>

```

FRAMEWORK PROTOTYPE:

Facilita el desarrollo de aplicaciones con JavaScript y Ajax.

Funciones y Métodos básicos:

\$() → atajo de document.getElementById()

Si se le pasa un parámetro, devuelve un objeto, si se le pasan varios parámetros devuelve un array simple con todos los objetos.

// Con JavaScript

```

var elemento1 = document.getElementById('primero');
var elemento2 = document.getElementById('segundo');

```

// Con Prototype

```

var elemento = $('primero');
var elementos = $('primero', 'segundo');

```

\$F() → obtiene el valor de los campos del formulario.

Ej.1: <input id="municipio" />

// Con JavaScript

```

document.getElementById("municipio").value

```

// Con Prototype

```

$F("municipio")

```

Ej.2: <select id="municipio">

<option>...</option>

</select>

// Con JavaScript

document.getElementById("municipio").options[document.getElementById("municipio").selectedIndex].value

// Con Prototype

\$F("municipio")

\$\$() → permite seleccionar elementos de la pág. utilizando selectores CSS.

<div id="principal">

<p>Primer párrafo</p>

<p>Segundo párrafo</p>

</div>

<p>Tercer párrafo</p>

var todosParrafos = \$\$('p');

var parrafosInteriores = \$\$('#principal p');

\$A() → convierte en array cualquier cosa que se parezca a un array.

getElementsByTagName() → devuelve un objeto de tipo NodeList o HTMLCollection, que no son arrays pero pueden recorrerse como tales.

<select id="lista">

<option value="1">Primer valor</option>

<option value="2">Segundo valor</option>

<option value="3">Tercer valor</option>

</select>

// 'lista_nodos' es una variable de tipo NodeList

```
var lista_nodos = $('lista').getElementsByName('option');
```

// 'nodos' es una variable de tipo array

```
var nodos = $A(lista_nodos);
```

```
// nodos = [ objeto_html_opcion1  
            , objeto_html_opcion2  
            , objeto_html_opcion3]
```

\$H() → crea arrays asociativos (llamados hash).

```
var usuarios = { usuario1: "password1",  
                usuario2: "password2",  
                usuario3: "password3" };
```

```
var hash_usuarios = $H(usuarios);
```

```
var logins = hash_usuarios.keys();
```

```
// logins = ["usuario1", "usuario2", "usuario3"]
```

```
var passwords = hash_usuarios.values();
```

```
// passwords = ["password1", "password2", "password3"]
```

```
var queryString = hash_usuarios.toQueryString();
```

```
// queryString= "usuario1=password1&usuario2=password2&usuario3=password3"
```

```
var debug = hash_usuarios.inspect();
```

```
// #<Hash:{'usuario1':'password1'  
          , 'usuario2':'password2',  
          'usuario3':'password3'}>
```

\$R() → crea rangos de valores, desde el valor del primer argumento, hasta el Segundo argumento. El tercer argumento indica si se excluye o no.


```
var rango = $R(0, 100, false);  
    // rango = [0, 1, 2, 3, ..., 100]
```

```
var rango = $R(0, 100);  
    // rango = [0, 1, 2, 3, ..., 100]
```

```
var rango = $R(0, 100, true);  
    // rango = [0, 1, 2, 3, ..., 99]
```

```
var rango = $R(100, 0);  
    // rango = [100]
```

```
var rango = $R(0, 100);
```

```
var incluido = rango.include(400);  
    // incluido = false
```

```
var rango = $R('a', 'k');  
    // rango = ['a', 'b', 'c', ..., 'k']
```

```
var rango = $R('aa', 'ak');  
    // rango = ['aa', 'ab', 'ac', ..., 'ak']
```

inspect() → devuelve una cadena de texto que es una representación de los contenidos del objeto.

Permite visualizar el contenido de variables complejas.

Se puede utilizar con cadenas de texto, objetos y arrays de cualquier tipo.

Funciones de Prototype para Cadenas de Texto:

stripTags() → elimina las etiquetas HTML y XML.

stripScripts() → elimina todos los bloques <script></script>

escapeHTML() → transforma los caracteres problemáticos en Html.

< → se transforma en <

& → &

`<p> → <p>`

unescapeHTML() → inversa a escapeHTML

```
var cadena = "<p>Prueba de texto & caracteres como  
&ntilde;</p>".unescapeHTML();
```

```
// cadena = "Prueba de texto & caracteres como ñ"
```

extractScripts() → devuelve un array con todos los bloques `<script></script>` de la cadena de texto.

evalScripts() → ejecuta c/u de los bloques `<script></script>`

toQueryParams() → convierte una cadena de texto de tipo querystring en un array asociativo hash de pares parámetro/valor.

```
var cadena = "parametro1=valor1&parametro2=valor2&parametro3=valor3";
```

```
var parametros = cadena.toQueryParams();
```

```
// $H(parametros).inspect() = #<Hash:{'parametro1':'valor1', 'parametro2':'valor2',  
parametro3':'valor3'}>
```

toArray() → convierte la cadena de texto en un array que contiene sus letras.

camelize() → convierte la cadena de texto separada por guiones en una cadena con notación camelCase.

```
var cadena="cualquier-cosa-asi".camelize();
```

```
//cadena="cualquierCosaAsi"
```

underscore() → contraria a camelize, pero con guiones bajos.

```
var cadena="AlgoEsAsi".underscore();
```

```
//cadena="algo_es_asi"
```

dasherize() → convierte los guiones bajos por guiones medios.

```
var cadena="esto_tambien".dasherize();
```

```
//cadena="esto-tambien"
```

Nota: combinando camelize(), underscore() y dasherize(), se pueden obtener el nombre DOM de cada propiedad CSS.

```
var cadena = 'borderTopStyle'.underscore().dasherize();
```

```
// cadena = 'border-top-style'
```

```
var cadena = 'border-top-style'.camelize();
```

```
// cadena = 'borderTopStyle'
```

Funciones de Prototype para Elementos:

Los elementos obtenidos mediante \$(), pueden usar las siguientes funciones:

Element.visible() → devuelve true/false si el elemento es visible/oculto (devuelve true para los campos tipo hidden).

Element.show() y **Element.hide()** → muestra y oculta el elemento.

Element.toggle() → si el elemento es visible, lo oculta y si está oculto, lo muestra.

Element.scrollTo() → baja o sube el scroll de la página hasta la posición del elemento indicado.

Element.getStyle() y **Element.setStyle()** → obtiene/establece el valor del estilo CSS.

Element.classListNames() → obtiene los class del elemento.

Element.hasClassName() → devuelve true/false si incluye un determinado class.

Element.addClassName() → añade un class al elemento.

Element.removeClassName() → elimina el class al elemento.

Nota: todas las funciones anteriores se pueden invocar de dos formas diferentes:

// Las dos instrucciones son equivalentes

```
Element.toggle('principal');
```

```
$('principal').toggle()
```

Funciones de Prototype para Formularios:

Field.clear() → borra el valor de cada campo que se le pasa (admite uno o más parámetros).

Field.present() → devuelve true si los campos que se le pasan han sido llenados por el usuario (valores no vacíos).

Field.focus() → le pone foco al campo del formulario.

Field.select() → selecciona el valor del campo.

Field.activate() → combina en una única función los métodos focus() y select()

Ej: Las funciones se pueden invocar de 3 formas:

// Las 3 instrucciones son equivalentes

```
Form.Element.focus('id_elemento');
```

```
Field.focus('id_elemento')
```

```
$('#id_elemento').focus()
```

Las siguientes funciones sirven para el formulario completo:

Form.serialize() → devuelve una cadena de texto de tipo "query string" con el valor de todos los campos del formulario ("campo1=valor1&campo2=valor2&campo3=valor3").

Form.findFirstElement() → devuelve el primer campo activo del formulario.

Form.getElements() → devuelve un array con todos los campos del formulario (incluyendo los elementos ocultos).

Form.getInputs() → devuelve un array con todos los elementos <input>. Se puede filtrar los resultados pasándole como segundo parámetro el tipo de <input> que se quiere obtener y el tercer parámetro indica el nombre del elemento <input>.

Form.disable() → deshabilita todo el formulario.

Form.enable() → habilita todo el formulario.

Form.focusFirstElement() → pone el foco del formulario en el primer campo que sea visible y esté habilitado.

Form.reset() → resetea el formulario. Es equivalente al método reset() de JavaScript.

Funciones de Prototype para Arrays:

clear() → vacía los contenidos el array y lo devuelve.

compact() → devuelve el array sin elementos null o undefined.

first() → devuelve el primer elemento del array.

flatten() → convierte cualquier array, en un array unidimensional. Se realiza un proceso recursivo que va "aplanando" el array:

```
var array_original = ["1", "2", 3,  
    ["a", "b", "c",  
        ["A", "B", "C"]  
    ]  
];  
  
var array_plano = array_original.flatten();  
  
    // array_plano = ["1", "2", 3, "a", "b", "c", "A", "B", "C"]
```

indexOf(value) → devuelve el valor de la posición del elemento en el array ó -1 si no lo encuentra.

```
array_original.indexOf(3); // 2  
array_original.indexOf("C"); // -1
```

last() → devuelve el último elemento del array.

reverse() → devuelve el array en sentido inverso.

shift() → elimina y devuelve el primer elemento.

without() → devuelve el array sin los elementos que se le pasan como parámetros.

```
var array = [12, 15, 16, 3, 40].without(16, 12)  
  
// array = [15, 3, 40]
```

Funciones de Prototype para Objetos Enumerables:

Algunos objetos en JavaScript se comportan como colecciones de valores, también llamados "enumeraciones" de valores.

En Prototype, se definen utilidades para estos objetos con Enumerable.

Cuando se obtienen objetos con las funciones de Prototype, ya se incluyen los métodos de Enumerable.

Si se quiere añadir esos métodos a un objeto propio, se pueden usar las utilidades de Prototype para crear objetos y extenderlos.

```
var miObjeto=Class.create();
```

```
Object.extend(miObjeto.prototype, Enumerable);
```

Recorrido Arrays:

```
var vocales = ["a", "e", "i", "o", "u"];
```

```
// Recorrer el array con JavaScript
```

```
for(var i=0; i<vocales.length; i++) {  
    alert("Vocal " + vocales[i] + " está en la posición " + i);  
}
```

```
// Recorrer el array con Prototype:
```

```
vocales.each(function(elemento, indice) {  
    alert("Vocal " + elemento + " está en la posición " + indice);  
});
```

select() → es un alias del método **findAll()** → permite filtrar los contenidos de un array.

```
var numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
```

```
resultado = numeros.findAll(function(elemento) {  
    return elemento > 5;  
});
```

```
// resultado = [6, 7, 8, 9, 10]
```

pluck() → obtiene los valores de una misma propiedad para todos los elementos de la colección.

```
var numLetras = ['hola', 'mundo', 'que', 'bien', 'funciona', 'Prototype'].pluck('length');
```

```
// numLetras = [4, 5, 3, 4, 8, 9]
```

partition() → divide una colección en 2 grupos, el de los elementos de tipo true y el de los elementos de tipo false (valores como null, undefined).

```
var valores = ['nombreElemento', 12, null, 2, true, , false].partition();
```

```
// valores = [['nombreElemento', 12, 2, true], [null, undefined, false]]
```

Este método también permite asignar una función propia para determinar si un elemento es true o false.

Ej:

```
var letras= $R('a', 'k').partition(function(n) {  
    return ['a', 'e', 'i', 'o', 'u'].include(n);  
})
```

```
// letras = [['a', 'e', 'i'], ['b', 'c', 'd', 'f', 'g', 'h', 'j', 'k']]
```

invoke() → permite ejecutar una función para todos los elementos de la colección.

```
var palabras = ['hola', 'mundo'].invoke('toUpperCase');
```

```
// palabras = ['HOLA', 'MUNDO']
```

Nota: en la documentación de Enumerable hay muchos otros métodos útiles.

Otras Funciones Útiles de Prototype:

Try.these() → permite probar varias funciones de forma consecutiva, hasta que una funcione.

Es muy útil para las aplicaciones que deben funcionar correctamente en varios navegadores diferentes.

Ej: se usa Try.these(), para obtener el objeto encargado de hacer las peticiones AJAX.

```
var Ajax = {  
    getTransport: function() {  
        return Try.these(  
            function() {return new XMLHttpRequest();},  
            function() {return new ActiveXObject('Msxml2.XMLHTTP')},  
            function() {return new ActiveXObject('Microsoft.XMLHTTP')}  
        ) || false;  
    }  
};
```

```

    ↑
    ↑
    },
    activeRequestCount: 0
  }

```

Class.create() → permite crear clases de una forma elegante y sencilla.

```
MiClase = Class.create();
```

```

MiClase.prototype = {
  initialize: function(a, b) {
    this.a = a;
    this.b = b;
  }
}

```

```
var miClase = new MiClase("primer_valor", "segundo_valor");
```

Object.extend() → añade o sobre escribe las propiedades de un objeto en otro. Es una forma muy básica y primitiva de herencia entre clases.

En la llamada a la función, el primer objeto es el destino en el que se copian las propiedades del segundo objeto.

```
Object.extend(objetoDestino, objetoOrigen);
```

Esta función es muy útil para que las aplicaciones definan una serie de opciones por defecto y puedan tener en cuenta las opciones establecidas por cada usuario.

Ej: //El array “opciones” guarda las opciones por defecto de la aplicación

```
var opciones = {campo: “usuario”, orden: “ASC”};
```

```
// El usuario establece sus propias opciones
```

```
var opciones_usuario = {orden: “DESC”, tipoBusqueda: “libre”};
```

```
// Se mezclan los dos arrays de opciones, dando prioridad a las opciones del usuario
```

```
Object.extend(opciones, opciones_usuario);
```

```
// Ahora, opciones.orden = “DESC”
```


Funciones de Prototype para AJAX:

- **Ajax.Request()** → realiza las peticiones AJAX y procesa sus resultados.

`new Ajax.Request(url,opciones);`

Opciones, es un parámetro opcional, y se indican en forma de array asociativo, si no se indica nada, se toman los valores por defecto.

```
Ej: new Ajax.Request('/ruta/hasta/pagina.php', {  
    method: 'post',  
    asynchronous: true,  
    postBody: 'parametro1=valor1&parametro2=valor2',  
    onSuccess: procesaRespuesta,  
    onFailure: muestraError  
});  
  
function procesaRespuesta(respuesta) {  
    alert(respuesta.responseText);  
}
```

Todas las opciones de `Ajax.Request()` son:

method → el método de la petición HTTP, por defecto es POST.

parameters → lista de valores que se envían junto con la petición. Deben estar en formato queryString. Ej: parametro1=valor1¶metro2=valor2

encoding → indica la codificación de los datos enviados en la petición. Por defecto es UTF-8.

asynchronous → tipo de petición, por defecto es true (asíncrona).

postBody → contenido que se envía en el cuerpo de la petición de tipo POST.

contentType → valor de la cabecera Content-Type, utilizada para hacer la petición. Por defecto: application/x-www-form-urlencoded

requestHeaders → array con todas las cabeceras propias que se quieran enviar.

onComplete, onLoad, on404, on500 → permiten asignar funciones para el manejo de las distintas faces de la petición.

Se pueden indicar funciones para todos los códigos de estado validos de HTTP.

onSuccess → invoca la función que procesa las respuestas correctas del servidor.

onFailure → invoca la función que se ejecuta cuando la respuesta es incorrecta.

onException → invoca la función encargada de manejar las peticiones erróneas, en las que la respuesta del servidor no es válida, los argumentos incluidos en la petición no son válidos, etc.

- **Ajax.Updater()** → versión especial de Ajax.Request(). Actualiza el contenido HTML de un elemento de la pág. con la respuesta del servidor.

Ej: <div id="info"></div>

new Ajax.Updater('info', '/ruta/pagina.php');

Si la respuesta del servidor es

```
<ul>
  <li>Algo</li>
  <li>Otra Cosa</li>
  <li>Más info</li>
</ul>
```

Después de realizar la petición Ajax.Updater(), dentro del div con id="info", se muestra la respuesta del servidor.

new Ajax.Updater(elemento,url,opciones);

Las opciones son las mismas de Ajax.Request(), y adicionalmente:

insertion → indica cómo se inserta el contenido HTML en el elemento indicado, puede ser:

Insertion.Before

Insertion.Top

Insertion.Bottom

Insertion.After

evalScripts → si la respuesta del servidor incluye scripts en su contenido, permite indicar si se ejecutan o no. Por defecto es false (no se ejecuta).

- **Ajax.PeriodicalUpdater()** → ejecuta de forma repetitiva una llamada a Ajax.Updater().

Ej: <div id="titulares"></div>

```
new Ajax.PeriodicalUpdater('titulares', '/ruta/pagina.php', { frequency:30 });
```

→ Actualiza cada 30 seg. el contenido del div con la respuesta del servidor.

También cuenta con las opciones anteriores y adicionalmente con:

frequency → # de segundos entre peticiones. Por defecto: 2 seg.

decay → indica el factor que se aplica a la frecuencia de actualización, cuando la última respuesta del servidor es la misma.

Ej: si la frecuencia es 10 seg. y el decay vale 3, cuando una respuesta del servidor sea igual a la anterior, la siguiente petición se hará $3 * 10 = 30$ segundos después de la última petición.

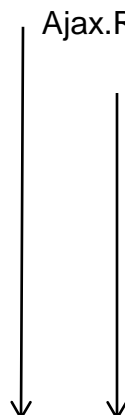
- **Ajax.Responders** → asigna de forma global las funciones encargadas de responder a eventos AJAX.

Sirve para indicarle al usuario si se está ejecutando o realizando alguna petición AJAX.

Sus métodos principales son register() y unregister().

Se les pasa como parámetros un objeto de tipo array asociativo, que incluye las funciones que responden a cada evento.

```
Ajax.Responders.register({
  onCreate: function() {
    if($('info') && Ajax.activeRequestCount > 0) {
      $('info').innerHTML = Ajax.activeRequestCount + "peticiones pendientes";
    }
  }
})
```



```

    },
    onComplete: function() {
        if($('info') && Ajax.activeRequestCount > 0) {
            $('info').innerHTML = Ajax.activeRequestCount + "peticiones
            pendientes";
        }
    }
}
});

```

LIBRERIA SCRIPTACULOUS:

Basada en Prototype, facilita el desarrollo de aplicaciones JavaScript.

Tiene diversos módulos:

- Efectos: se aplican a los elementos de la pág.
Ej: parpadeo, movimiento rápido, aparecer/desaparecer, aumentar/disminuir de tamaño, desplegarse, etc.
- Controles: arrastrar y soltar, autocompletar, editor de contenidos.
- Utilidades: builder, utilizado para crear fácilmente nodos y fragmentos complejos de DOM.

Ej: Formulario de Autocompletar con Scriptaculous:

```

window.onload = function() {

    // Crear elemento de tipo <div> para mostrar las sugerencias del servidor
    var elDiv = Builder.node('div', {id:'sugerencias'});

    document.body.appendChild(elDiv);

    new Ajax.Autocompleter('municipio', 'sugerencias',
    'http://localhost/autocompletaMunicipios.php?modo=ul',
    {paramName: 'municipio'}

    );
}

```

La sintaxis del control Ajax.Autocompleter() es la siguiente:

```
new Ajax.Autocompleter(idCuadroTexto, idDivResultados, url, opciones);
```

idCuadroTexto → donde el usuario escribe las letras que se van a autocompletar.

idDivResultados → donde se muestra la respuesta del servidor.

url → La url del archivo del servidor que recibe lo que el usuario escribe y devuelve las sugerencias.

opciones → permite modificar las opciones por defecto del control de autocompletar.

Las opciones son:

paramName: parámetro enviado al servidor con el texto que escribió el usuario.

Por defecto es igual al atributo name del cuadro de texto utilizado para autocompletar.

tokens: permite autocompletar más de un valor en un mismo cuadro de texto.

```
Ej: new Ajax.Autocompleter('municipio', 'sugerencias',  
    'http://localhost/autocompletaMunicipios.php?modo=ul',  
    { paramName: 'municipio', tokens: ',' }  
);
```

La opción tokens, indica que el carácter (,) separa los elementos dentro del mismo cuadro de texto.

Así, después de autocompletar una palabra, se escribe el carácter (,) y el script autocompletará la siguiente palabra.

minChars → # mínimo de caracteres que el usuario debe escribir, antes de que se realice la petición al servidor. Por defecto= 1 carácter.

indicator → elemento que se muestra mientras se realiza la petición al servidor y se vuelve a ocultar cuando recibe respuesta.

Ej: gif de cargando.

updateElement → se ejecuta cuando el usuario selecciona una opción del div de autocompletar. Lo que hace es mostrar el valor seleccionado en el cuadro de texto y ocultar la lista de sugerencias. También se puede indicar una función propia para evitar este comportamiento por defecto.

afterUpdateElement → se le indica una función, para que se ejecute después de updateElement.

AJAX CON JQUERY:

\$.ajax(opciones) → realiza la petición Ajax.

Ej: \$.ajax({
 url: '/ruta/hasta/pagina.php',
 type: 'POST',
 async: true,
 data: 'parametro1=valor1¶metro2=valor2',
 success: procesaRespuesta,
 error: muestraError
});

Las opciones son:

async → indica con true, si es asíncrona.

beforeSend → permite indicar una función que modifique el objeto XMLHttpRequest antes de realizar la petición. El propio objeto XMLHttpRequest se pasa como único argumento de la función.

complete → se le indica la función que se ejecuta cuando la petición se ha completado (y después de ejecutar, si se han establecido, las funciones de success o error).

La función recibe el objeto XMLHttpRequest, como primer parametro y el resultado de la petición como segundo parámetro.

contentType → indica el valor de la cabecera Content-Type utilizada para hacer la petición.

Por defecto es: application/x-www-form-urlencoded

data → info que se envía al servidor. Si no es una cadena de texto, se convierte en un queryString (parametro1=valor1¶metro2=valor2).

También se puede indicar un array asociativo (clave/valor), que se convierte en queryString.

dataType → el tipo de dato que se espera como respuesta. Si no se indica ningún valor, jQuery lo deduce a partir de las cabeceras de respuesta.

Los valores son:

- "xml" → se devuelve un documento XML (responseXML).
- "html" → se devuelve la respuesta del servidor (responseText)
- "script" → se evalúa la respuesta como si fuera JavaScript y se devuelve el resultado.
- "json" → se evalúa la respuesta JSON y se devuelve el objeto JavaScript generado.

error → indica la función que se ejecuta cuando se produce un error durante la petición.

Recibe el objeto XMLHttpRequest, como primer parámetro, una cadena de texto que indica el error como segundo parámetro, y un objeto con la excepción producida como tercer parámetro.

ifModified → permite considerar como correcta la petición, solamente si la respuesta recibida es diferente a la anterior respuesta. Por defecto su valor es false.

processData → indica si se transforman los datos de la opción data, en cadena de texto. Si se pone false, no se hace la conversión automática.

success → se establece la función que se ejecuta cuando la petición se ha completado correctamente.

Recibe como primer parámetro los datos recibidos del servidor, previamente formateados según la opción dataType.

timeout → tiempo máximo, en milisegundos, que la petición espera la respuesta del servidor antes de anularse.

type → el tipo de petición que se realiza. Su valor por defecto es GET.

url → la URL del servidor al que se realiza la petición.

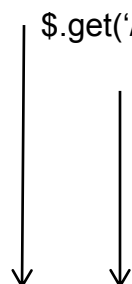
\$.get(url, datos, funcionManejadora); → realiza una petición Ajax de tipo GET.

Ej: //petición simple:

```
$.get('/ruta/pagina.php');
```

// petición GET, con envío de parámetros y función que procesa la respuesta:

```
$.get('/ruta/pagina.php',  
  {      //datos enviados al servidor  
    dato1: "valor1",  
    dato2: "valor2"
```



```

↑
↑
    },
    //función que maneja la respuesta
    function(data){
        $( "#resultado" ).html( data );
        //alert("Respuesta: "+data);
    }
    );

```

\$.post() → Se hace de la misma forma que \$.get(), la única diferencia es que los parámetros que envía al servidor, los envía por POST (en las cabeceras del HTTP).

\$.load() → igual a Ajax.Updater() del framework prototype. Inserta el contenido de la respuesta del servidor en un elemento Html (ej: div).

```
<div id="info"></div>
```

// Con Prototype

```
new Ajax.Updater('info', '/ruta/pagina.php');
```

// Con jQuery

```
$('#info').load('/ruta/pagina.php');
```

\$.getJSON() y **\$.getScript()** → ambos métodos envían los parámetros al servidor por método GET, luego cargan y ejecutan, una respuesta de tipo JSON y una respuesta con código JavaScript, respectivamente.

Nota: Para más información de los métodos de Ajax en JQuery:
<http://www.desarrolloweb.com/articulos/entendiendo-ajax-jquery.html>

OTROS FRAMEWORKS:

- Dojo.
 - Mootools.
-

DETENER PETICIONES HTTP ERRÓNEAS:

Se usa la función setTimeout(), para hacer una cuenta regresiva cuando se hace una petición al servidor.

Si el servidor responde antes de acabarse la cuenta regresiva, se elimina la cuenta atrás y se continúa con la ejecución normal de la aplicación.

Si el servidor no responde, y acaba la cuenta regresiva, se ejecuta una función encargada de detener la petición, reintentarla, mostrar un mensaje al usuario, etc.

// Variable global que almacena el identificador de la cuenta atrás

var cuentaAtras = null;

var tiempoMaximo = 5000; // 5 segundos

```
function cargaContenido(url, metodo, funcion) {  
    peticion_http = inicializa_xhr();  
    if(peticion_http) {  
        // Establecer la cuenta atrás al realizar la petición HTTP  
        cuentaAtras = setTimeout(expirada, tiempoMaximo);  
        peticion_http.onreadystatechange = funcion;  
        peticion_http.open(metodo, url, true);  
        peticion_http.send(null);  
    }  
}
```

```
function muestraMensaje() {  
    ...  
    if(peticion_http.readyState == READY_STATE_COMPLETE) {  
        if(peticion_http.status == 200) {  
            // Si se ha recibido la respuesta del servidor, eliminar la cuenta atrás  
            clearTimeout(cuentaAtras);  
            ...  
        }  
    }  
}
```



```

↑
|
}
↑
|
}

```

```

function expirada() {
    // La cuenta atrás se ha cumplido, detener la petición HTTP pendiente
    peticion_http.abort();

    alert("Se ha producido un error en la comunicación con el servidor.
    Inténtalo un poco más adelante.");
}

```

Nota: una aplicación AJAX, debe estar preparada para otro tipo de respuestas por parte del servidor.

Estas respuestas se obtienen con el valor del atributo **status**, del objeto XMLHttpRequest.

Códigos De estado devueltos por el Servidor:

Códigos de información

status	statusText	Explicación
100	Continue	Una parte de la petición (normalmente la primera) se ha recibido sin problemas y se puede enviar el resto de la petición
101	Switching protocols	El servidor va a cambiar el protocolo con el que se envía la información de la respuesta. En la cabecera Upgrade indica el nuevo protocolo

Códigos de petición y respuesta correctas

status	statusText	Explicación
200	OK	La petición se ha recibido correctamente y se está enviando la respuesta. Este código es con mucha diferencia el que mas devuelven los servidores

status	statusText	Explicación
201	Created	Se ha creado un nuevo recurso (por ejemplo una página web o un archivo) como parte de la respuesta
202	Accepted	La petición se ha recibido correctamente y se va a responder, pero no de forma inmediata
203	Non-Authoritative Information	La respuesta que se envía la ha generado un servidor externo. A efectos prácticos, es muy parecido al código 200
204	No Content	La petición se ha recibido de forma correcta pero no es necesaria una respuesta
205	Reset Content	El servidor solicita al navegador que inicialice el documento desde el que se realizó la petición, como por ejemplo un formulario
206	Partial Content	La respuesta contiene sólo la parte concreta del documento que se ha solicitado en la petición

Códigos de redirección

status	statusText	Explicación
300	Multiple Choices	El contenido original ha cambiado de sitio y se devuelve una lista con varias direcciones alternativas en las que se puede encontrar el contenido
301	Moved Permanently	El contenido original ha cambiado de sitio y el servidor devuelve la nueva URL del contenido. La próxima vez que solicite el contenido, el navegador utiliza la nueva URL
302	Found	El contenido original ha cambiado de sitio de forma temporal. El servidor devuelve la nueva URL, pero el navegador debe seguir utilizando la URL original en las próximas peticiones

status	statusText	Explicación
303	See Other	El contenido solicitado se puede obtener en la URL alternativa devuelta por el servidor. Este código no implica que el contenido original ha cambiado de sitio
304	Not Modified	Normalmente, el navegador guarda en su caché los contenidos accedidos frecuentemente. Cuando el navegador solicita esos contenidos, incluye la condición de que no hayan cambiado desde la última vez que los recibió. Si el contenido no ha cambiado, el servidor devuelve este código para indicar que la respuesta sería la misma que la última vez
305	Use Proxy	El recurso solicitado sólo se puede obtener a través de un proxy, cuyos datos se incluyen en la respuesta
307	Temporary Redirect	Se trata de un código muy similar al 302, ya que indica que el recurso solicitado se encuentra de forma temporal en otra URL

Códigos de error del navegador

status	statusText	Explicación
400	Bad Request	El servidor no entiende la petición porque no ha sido creada de forma correcta
401	Unauthorized	El recurso solicitado requiere autorización previa
402	Payment Required	Código reservado para su uso futuro
403	Forbidden	No se puede acceder al recurso solicitado por falta de permisos o porque el usuario y contraseña indicados no son correctos
404	Not Found	El recurso solicitado no se encuentra en la URL indicada. Se trata de uno de los códigos más

status	statusText	Explicación
		utilizados y responsable de los típicos errores de <i>Página no encontrada</i>
405	Method Not Allowed	El servidor no permite el uso del método utilizado por la petición, por ejemplo por utilizar el método GET cuando el servidor sólo permite el método POST
406	Not Acceptable	El tipo de contenido solicitado por el navegador no se encuentra entre la lista de tipos de contenidos que admite, por lo que no se envía en la respuesta
407	Proxy Authentication Required	Similar al código 401, indica que el navegador debe obtener autorización del proxy antes de que se le pueda enviar el contenido solicitado
408	Request Timeout	El navegador ha tardado demasiado tiempo en realizar la petición, por lo que el servidor la descarta
409	Conflict	El navegador no puede procesar la petición, ya que implica realizar una operación no permitida (como por ejemplo crear, modificar o borrar un archivo)
410	Gone	Similar al código 404. Indica que el recurso solicitado ha cambiado para siempre su localización, pero no se proporciona su nueva URL
411	Length Required	El servidor no procesa la petición porque no se ha indicado de forma explícita el tamaño del contenido de la petición
412	Precondition Failed	No se cumple una de las condiciones bajo las que se realizó la petición
413	Request Entity Too Large	La petición incluye más datos de los que el servidor es capaz de procesar. Normalmente este error se produce cuando se adjunta en la petición un archivo

status	statusText	Explicación
		con un tamaño demasiado grande
414	Request-URI Too Long	La URL de la petición es demasiado grande, como cuando se incluyen más de 512 bytes en una petición realizada con el método GET
415	Unsupported Media Type	Al menos una parte de la petición incluye un formato que el servidor no es capaz procesar
416	Requested Range Not Suitable	El trozo de documento solicitado no está disponible, como por ejemplo cuando se solicitan bytes que están por encima del tamaño total del contenido
417	Expectation Failed	El servidor no puede procesar la petición porque al menos uno de los valores incluidos en la cabecera Expect no se pueden cumplir

Códigos de error del servidor

status	statusText	Explicación
500	Internal Server Error	Se ha producido algún error en el servidor que impide procesar la petición
501	Not Implemented	Procesar la respuesta requiere ciertas características no soportadas por el servidor
502	Bad Gateway	El servidor está actuando de proxy entre el navegador y un servidor externo del que ha obtenido una respuesta no válida
503	Service Unavailable	El servidor está sobrecargado de peticiones y no puede procesar la petición realizada
504	Gateway Timeout	El servidor está actuando de proxy entre el navegador y un servidor externo que ha tardado

status	statusText	Explicación
		demasiado tiempo en responder
505	HTTP Version Not Supported	El servidor no es capaz de procesar la versión HTTP utilizada en la petición. La respuesta indica las versiones de HTTP que soporta el servidor

Ejercicio 19: AUTOCOMPLETAR CON JQUERY:

```
<link rel="stylesheet"
href="http://code.jquery.com/ui/1.10.4/themes/smoothness/jquery-ui.css">
```

```
<script src="http://code.jquery.com/jquery-1.9.1.js"></script>
```

```
<script src="http://code.jquery.com/ui/1.10.4/jquery-ui.js"></script>
```

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

```
/*La respuesta del servidor es:
```

```
{
```

```
  "a": ["Ababuj", "Abades", "Abadía", "Abadín", "Abadiño", "Abáigar", "Abajas",
"Abaltzisketa", "Abánades", "Abanilla", "Abanto y Ciérvana-Abanto Zierbena",
"Abanto", "Abarán", "Abarca de Campos", "Abárzuza", "Abaurregaina/Abaurrea
Alta", "Abaurrepea/Abaurrea Baja", "Abegondo", "Abejar", "Abejuela", "Abella de la
Conca"],
```

```
  "al": ["Alacón", "Aladrén", "Alaejos", "Alagón", "Alaior", "Alájar", "Alajeró",
"Alameda de Gardón (La)", "Alameda de la Sagra", "Alameda del Valle",
"Alameda", "Alamedilla (La)", "Alamedilla", "Alamillo", "Alaminos", "Alamús (Els)",
"Alange", "Alanís", "Alaquàs", "Alar del Rey", "Alaraz"],
```

```
  ...
```

```
}
```

```
*/
```

```
function todo(){
    $("#municipio").autocomplete({
        source: "http://localhost/autocompletaMunicipiosJquery.php"
    });
}
```

```
$(document).on('ready', function() {
    //obtener el cuadro de texto
    var texto_input=$("#municipio");
    //pedir info al server cada vez que se escribe algo
    texto_input.on('keyup',todo);
    //darle foco
    texto_input.focus();
});
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<form>
```

```
    Digite el Municipio(España):
```

```
    <input type="text" id="municipio"/>
```

```
</form>
```

```
</body>
```

Nota: la funcionalidad autocomplete(), toma lo escrito por el usuario en el input, y lo envía por método **GET**, dentro de una variable llamada **term**.

Ej: Si el usuario escribe: "pro", se invoca al script php así:

http://localhost/autocompletaMunicipiosJquery.php?**term**=pro

RECURSOS ÚTILES:

Ajaxload: utilidad para construir los pequeños iconos animados (gif) que las aplicaciones AJAX suelen utilizar para indicar al usuario que se está realizando una petición.

<http://www.ajaxload.info/>

MiniAjax: decenas de ejemplos reales de aplicaciones AJAX listas para descargar (galerías de imágenes, reflejos para imágenes, formularios, tablas reordenables, efecto lupa sobre las imágenes, etc.)

<http://miniajax.com/>

