



# **UD 5. MECANISMOS DE COMUNICACIÓN ASÍNCRONA.**



## 5.1. INTRODUCCIÓN.

- **Asíncrono:** Suceso o proceso que se produce de forma independiente al tiempo de procesamiento.
- **CORS (Cross Origin Resource Sharing):** Mecanismo que consiste en acceder a información o a datos de otro dominio distinto al del origen de la petición.
- **jQuery:** Biblioteca multipropósito que fue creada por John Resig y que ha sido mejorada en sucesivas versiones. Con esta librería, se puede manipular el DOM, crear animaciones, utilizar AJAX para realizar peticiones asíncronas al servidor, etc.
- **JSON (JavaScript Object Notation):** Formato para el almacenamiento e intercambio de datos.



## 5.1. INTRODUCCIÓN.

La **comunicación asíncrona** es una de las cualidades que confiere a JavaScript su máxima potencia. En una **secuencia síncrona** de larga duración, mientras el programa se está ejecutando, no se puede hacer nada más hasta que no haya terminado dicha operación. Sin embargo, con las **secuencias asíncronas**, podemos seguir realizando operaciones mientras se ejecuta en segundo plano la secuencia principal.

Tenéis más información acerca de las **secuencias asíncronas** en el siguiente enlace:

<https://developer.mozilla.org/es/docs/Learn/JavaScript/Asynchronous/Introducing>.



## 5.2. LOS WEB WORKERS.

Un web worker, como lo define la WC3, es un proceso de JavaScript que se ejecuta en segundo plano, de tal manera que la página web no se vea penalizada ni ralentizada por tener un proceso ejecutándose de forma permanente.

Se puede utilizar web workers para analizar los patrones de navegación o para realizar llamadas asíncronas a un servicio de forma transparente para el cliente. (usuario).

Para ejecutar un web worker, el navegador tiene que soportar HTML5 y estar en una versión no excesivamente antigua.



## 5.2. LOS WEB WORKERS

A continuación, se muestra un ejemplo de web worker en el que se ha creado el mismo, en un fichero aparte llamado *trabajador.js*. El fichero HTML principal tendrá el siguiente aspecto:

## Ejemplo: *web worker*.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo web worker</title>
</head>
<body>
  <p>Secuencia de Finobacci: <output id="result"></output></p>
  <button onclick="startWorker()">Start Worker</button>
  <button onclick="stopWorker()">Stop Worker</button>
```

## Ejemplo: *web worker*.

```
<script>

var w;

function startWorker(){
    if(typeof (Worker) !== "undefined") {
        if(typeof (W) === "undefined") {
            w = new Worker("trabajador.js");
        }
        w.onmessage = function(event){
            document.getElementById( "result").innerHTML = event.data;
        };
    } else{
        document.getElementById( "result").innerHTML = "Fallo al ejecutar el web
worker";
    }
}

function stopWorker(){
    w.terminate();
    w = undefined;
}

</script></body></html>
```

## Ejemplo: **web worker**.

Se ha incluido un botón para iniciar el web worker y otro botón para pararlo. Hay que tener en cuenta los siguientes aspectos del programa:

- El **web worker** se crea en la variable `w` siempre y cuando `w` tenga el tipo *“undefined”*. De lo contrario, devuelve *“Fallo al ejecutar el web worker”*.
- Cuando se crea el web worker, existe un modo de comunicación con la web host, que es el paso del mensaje. Se puede capturar los mensajes mediante un listener *“**w.onmessage = function(event)...**”*.
- El web worker puede enviar mensajes a la página web host mediante la función **postMessage()**.



```
var anterior = 0;
var actual = 1;
var secuencia = "";
function temporizador() {
    if(!anterior){//invierte el resultado de la expresión.
        secuencia+=" "+actual;
    }else{
        secuencia+=" -"+actual;
    }
    postMessage(secuencia);
    aux = anterior + actual;
    anterior = actual;
    actual = aux;
    setTimeout("temporizador()",500);
}
temporizador();
```



## 5.3. JSON. JavaScript Object Notation.

Es una sintaxis que se emplea para almacenar e intercambiar datos. Como podrá verse a continuación, es una alternativa más fácil de usar a XML.

### 5.3.1. Sintaxis JSON.

La sintaxis JSON en JavaScript tiene en cuenta cinco reglas:



### 5.3.1. Sintaxis JSON.

1. Es un subconjunto de la sintaxis de JavaScript.
2. Los datos aparecen como pares nombre/valor.
3. Los datos se separan con comas.
4. Las llaves { } contienen objetos.
5. Los corchetes [ ] contienen arrays.



### 5.3.1. Sintaxis JSON.

El siguiente ejemplo de JSON define el objeto *empleados* con un **array** que contiene tres registros del tipo *empleado*, el cual es un **objeto**:

```
{
  "empleados": [
    { "nombre": "Gerardo", "apellidos": "García" };
    { "nombre": "Maryna", "apellidos": "Hernández" };
    { "nombre": "Abel", "apellidos": "Pacheco" };
  ]
}
```



### 5.3.1. Sintaxis JSON.

Entre las ventajas de JSON, cabe citar las siguientes:

- ***Formato ligero de intercambio de datos.*** Se prescinde al máximo de elementos que sobrecarguen los datos de información innecesaria.
- ***Independiente del lenguaje.*** JSON utiliza la sintaxis JavaScript, pero el formato es solo texto, al igual que XML. El texto puede leerse y utilizarse como formato de datos por cualquier lenguaje de programación. (ver ejemplo a continuación).
- ***Autodescriptivo y fácil de entender.***

## Ejemplo: *Uso de Datos en formato JSON*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Uso Datos JSON</title>
</head>
<body>
  <h2>Ejemplo de objetos JSON </h2>
  <p id="demo"></p>
  <script>
    var text = '{"nombre":"IES KURSAAL","calle":"Virgen de Europa
4","teléfono":"956 67 07 67"}';
    var obj=JSON.parse(text);
    document.getElementById( "demo").innerHTML=
    obj.nombre + "<br>" + obj.calle + "<br>" + obj.teléfono;
  </script></body></html>
```



### 5.3.1. Sintaxis JSON.

Las **diferencias** entre JSON y XML son las siguientes:

- XML tiene que ser analizado con un analizador XML, mientras que JSON puede ser analizado por una función JavaScript estándar.
- JSON no utiliza etiqueta de fin.
- JSON es más corto.
- JSON es más rápido de leer y escribir.
- JSON puede utilizar matrices.
- JSON es más rápido y sencillo que XML.



### 5.3.1. Sintaxis JSON.

Las **similitudes** entre JSON y XML son las siguientes:

- JSON y XML son autodescriptivos (legibles por humanos).
- JSON y XML son jerárquicos (valores dentro de valores).
- JSON y XML pueden ser analizados y utilizados por muchos lenguajes de programación.
- Tanto JSON como XML se pueden obtener con una llamada *XMLHttpRequest*.

Pincha en el siguiente [enlace](#) para ver un vídeo explicativo acerca de las diferencias y similitudes de ambos lenguajes.





### 5.3.2. Instalación de un servidor JSON. Práctica guiada.

El objetivo de este ejercicio es mostrar cómo se puede instalar un servidor JSON en Node, desde el cual se puedan recuperar datos en este formato para luego ser utilizados en scripts (JavaScript).



## 5.4. Comunicación asíncrona. AJAX.

AJAX es el acrónimo de *Asynchronous JavaScript and XML*. Con ésta técnica, se pueden crear páginas dinámicas de forma rápida que consultan de una forma asíncrona fuentes de datos externas a la página.

Esta técnica permite que una página se actualice sin tener que enviar información otra vez al servidor, y esperar a que el servidor la responda y envíe información de vuelta. Como únicamente se intercambian datos, solo se envía lo que se necesita (no toda la página).



## 5.4. Comunicación asíncrona. AJAX.

Las páginas web antiguas tenían que recargarlas de forma completa. El programador tenía dos opciones, o traerse mucha información durante la carga, lo que ralentizaba mucho la navegación, o realizar muchas recargas de la información.

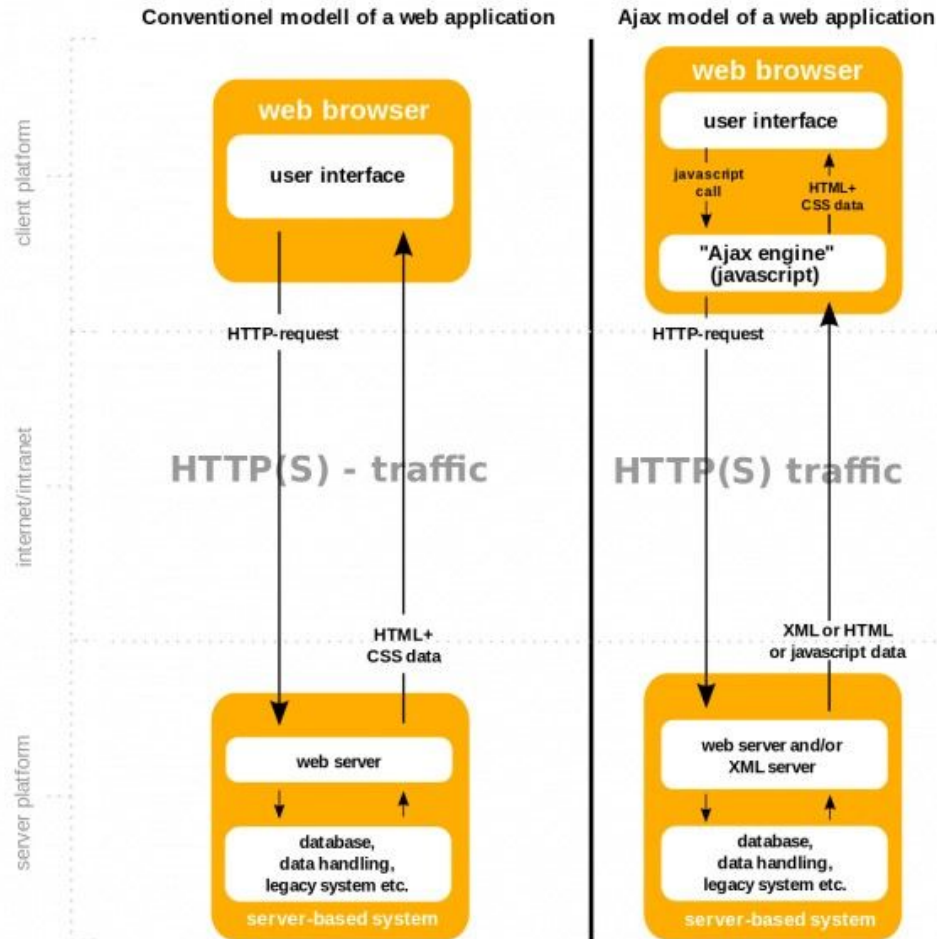
Cuando apareció la comunicación asíncrona, muchas aplicaciones comenzaron a utilizar AJAX como Google Maps, Gmail, YouTube, Facebook Tabs, etc.



## 5.4. Comunicación asíncrona. AJAX.

AJAX se basa en el objeto XMLHttpRequest para intercambiar datos con el servidor de forma asíncrona.

El esquema de funcionamiento de una web que utilice AJAX es el siguiente:



## Modelo convencional.

- 1) Se envía una solicitud desde el navegador web al servidor.
- 2) El servidor recibe y, posteriormente, recupera datos.
- 3) El servidor envía los datos solicitados al navegador web.
- 4) El navegador web recibe los datos y vuelve a cargar la página para que aparezcan los datos.

Durante este proceso, los usuarios no tienen más remedio que esperar hasta que se complete todo el proceso. No solo consume mucho tiempo, sino que también supone una carga innecesaria en el servidor.

## Modelo AJAX.

- 1) El navegador crea una llamada de JavaScript que luego activará XMLHttpRequest.
- 2) En segundo plano, el navegador web crea una solicitud HTTP al servidor.
- 3) El servidor recibe, recupera y envía los datos al navegador web.
- 4) El navegador web recibe los datos solicitados que aparecerán directamente en la página. No se necesita recargar.



### 5.4.1. Práctica guiada: sistema localización ciudades.

Se va a crear una pequeña página web en la que se pueda introducir el nombre de una ciudad y el sistema ofrezca sugerencias con ciudades que comiencen por las letras que se vayan introduciendo en un campo de texto. (Cómo el texto predictivo de Google).

El sistema estará compuesto de dos partes: en una, se tiene el código HTML (index.html) con los script JavaScript integrados y la pertinente llamada ***HttpRequest*** y, en otra, un fichero php llamado ***getSugerencia.php***, el cual recibe la petición y envía de vuelta el nombre de las ciudades que ha encontrado en una lista creada previamente.

```
<script>
    function muestraSugerencia(str){
        if(str.length == 0){
            document.getElementById( "txtSugerencia" ).innerHTML="";
            return;
        } else {
            var xmlhttp = new XMLHttpRequest();
            xmlhttp.onreadystatechange = function(){
                if(this.readyState == 4 && this.status == 200){
                    document.getElementById( "txtSugerencia" ).innerHTML=this.responseText;
                }
            };
            xmlhttp.open( "GET", "http://localhost/getSugerencia.php?param=" + str,
                true);
            xmlhttp.send();
        }
    }
</script>
```



## Práctica guiada: *Localización de ciudades.*

```
<body>
  <p><b>Escribe el nombre de alguna ciudad a donde quieras ir:</b></p>
  <form>
    Ciudad:<input type="text" onkeyup="muestraSugerencia(this.value)">
  </form>
  <p>Sugerencias: <span id="txtSugerencia"></span></p>
</body>
```

El script anterior va a realizar las llamadas:

- ***var xmlhttp = new XMLHttpRequest()***. Para crear un objeto del tipo XMLHttpRequest.
- ***xmlhttp.open()***. Para llamar al servicio o página web donde se van a recibir los datos.
- ***xmlhttp.send()***. para enviar la petición.

Como se va a hacer una petición asíncrona, el programador tiene que decir al sistema qué código tiene que ejecutarse cuando ocurra algo con la petición que ha realizado. Cada vez que ocurra algo en la petición, se ejecutará la función asociada al evento o método ***onreadystatechange*** del objeto XMLHttpRequest creado previamente:

- ***xmlhttp.onreadystatechange = function(){....}***

Una vez se esté dentro de esta función, se recoge el valor recibido y, en este caso, se le asignará a un elemento HTML creado para tal fin, pero ¿por qué se espera que se cumpla la condición “***this.readyState == 4 && this.status == 200***”? La respuesta es porque cuando:

- ***this.readyState == 4***: significa que la petición ha terminado y que la respuesta, por lo tanto está lista.
- ***this.status == 200***: significa que la petición se ha completado OK. Otros valores típicos de este atributo pueden ser el famoso 404 (not found) o el 403 (forbidden).

Valores del atributo *readyState*:

- 0 (no inicializada)
- 1 (leyendo).
- 2 (leído).
- 3 (interactiva).
- 4 (completo).

Ahora quedaría la parte servidora, que procesaría la petición AJAX. En este caso sería un script php colocado en el servidor de nombre *getSugerencia.php*:

```
<?php
```

```
//Array de nombres de ciudades
```

```
$arr[] = "Casares";
```

```
$arr[] = "Barcelona";
```

```
$arr[] = "Alcorcón";
```

```
$arr[] = "Málaga";
```

```
$arr[] = "Fuengirola";
```

```
$arr[] = "Marbella";
```

```
$arr[] = "Río de Janeiro";
```

```
$arr[] = "Gijón";
```

```
$arr[] = "Oviedo";
```

```
$arr[] = "Cádiz";
```

```
$arr[] = "Algeciras";
```

```
.
```

```
.
```

```
.
```

Práctica guiada: *Localización de ciudades.*

```
//Primero recorremos el parámetro URL
$param = $_REQUEST[ "param" ];
$sugerencia = "";

//Mira se coincide el parámetro con alguna de las ciudades
if ($param != ""){
    $param = strtolower($param); //convierte el parámetro recibido a minúscula.
    $len = strlen($param); //determina la longitud del parámetro recibido
    foreach($arr as $nombre){
        //comparamos si el texto recibido coincide con el comienzo de alguna ciudad
        if (stristr($param, substr($nombre, 0, $len))){
            if($sugerencia == ""){
                $sugerencia = $nombre; //primera sugerencia
            } else {
                $sugerencia = ", $nombre"; //segunda y siguientes sugerencias
            }
        }
    }
}

//En el caso de no encontrar ninguna sugerencia se le hace saber al usuario.
//En caso contrario devuelve las sugerencias encontradas.
echo $sugerencia == "" ? "ninguna sugerencia" : $sugerencia;

?>
```

## Práctica guiada: *Localización de ciudades.*

En un bucle, se va **recorriendo el array** y seleccionando aquellas posiciones del array cuyas primeras letras coinciden con el texto que el usuario ha tecleado en la página web.



## 5.4.2. Ejemplo con AJAX y JSON.

Se explicará con un ejemplo cómo se trata la información recibida en formato JSON por un script JavaScript. Vamos a tener un archivo HTML llamado ***json.html*** y un archivo de texto ***mywebs.txt*** con información en formato JSON, más concretamente, un array de tres objetos en los que se almacena tanto el nombre como la dirección de tres páginas web.

```
<body>
  <div id="lasWebs"></div>
  <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "mywebs.txt";
    xmlhttp.onreadystatechange = function() {
      if(this.readyState == 4 && this.status == 200){
        var listaWebs = JSON.parse(this.responseText);
        getWebs(listaWebs);
      }
    };
    xmlhttp.open("GET", url, true);
    xmlhttp.send();
```

Al recibir los datos se usa **parse** del objeto JSON para interpretar la información obtenida.

Esta llamada creará un array de objetos, el cual se envía como parámetro a la función **getWebs()**, que lo único que hace es recorrerlo y mostrar el contenido en el elemento **<div>** de la página web con el ID igual a **"lasWebs"**.



```
function getWebs(arr) {  
    var out = "";  
    var i;  
    for(i = 0; i < arr.length; i++){  
        out += '<a href=" ' + arr[i].url + '>' + arr[i].display +  
'</a><br>';  
    }  
    document.getElementById("lasWebs").innerHTML = out;  
}  
</script></body></html>
```

El contenido del fichero ***mywebs.txt***:

```
[
  {
    "display": "Web de IES Kursaal", "url":"http://ieskursaal.es/"
  },
  {
    "display": "Web de Ayto. Algeciras", "url":"http://www.algeciras.es/es/index.html"
  },
  {
    "display": "Web de Diputación Cádiz", "url":"https://www.dipucadiz.es/"
  },
]
```

Es un array de tres posiciones en las que tiene tres objetos, cada uno con dos campos: display y url.



## Tarea 5.1.

Verifica si el código expuesto en este apartado funciona correctamente en un navegador.  
Como característica especial, en vez de páginas web, utiliza razas de animales.



### 5.4.3. Ejemplo con AJAX y XML.

Según el W3C, “el lenguaje XML (Extensible Markup Language) es un formato de texto simple y muy flexible derivado de SGML (ISO 8879). Se diseñó originalmente para la publicación electrónica a gran escala y en la actualidad desempeña un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y en otros lugares”.

```
<concesionario>
<coche combustible="D">
    <marca>Volkswagen</marca>
    <modelo>Passat</modelo>
    <color>Azul</color>
    <año>2005</año>
    <precio>7900.00</precio>
</coche>
<coche combustible="G">
    <marca>Volkswagen</marca>
    <modelo>Touran</modelo>
    <color>Azul</color>
    <año>2007</año>
    <precio>8200.00</precio>
</coche>
...
</concesionario>
```

Este fichero se denominará ***coches.xml***, se situará en el mismo directorio que el archivo HTML y será el fichero que servirá para realizar la consulta.

El siguiente paso será crear el archivo cohes.html que contenga el script que lleve a cabo un filtrado sobre un determinado tipo de nodos del fichero coches.xml.

```
<body>
  <p id="listado"></p>
  <script>
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if(this.readyState == 4 && this.status == 200) {
        recuperaDatos(this);
      }
    };
    xhttp.open("GET", "coches.xml", true);
    xhttp.send();
    function recuperaDatos(xml) {
      var datos = "";
      var xmlDatos = xml.responseXML;
      array = xmlDatos.getElementsByTagName("marca");
      for (i = 0; i < array.length; i++) {
        datos = datos + "marca: " + array[i].childNodes[0].nodeValue + "<br>";
      }
      document.getElementById("listado").innerHTML = datos;
    }
  </script></body>
```

No se usa **responseTXT**, dado que el formato de los datos es XML.

La variable array contiene los nodos que van a ser seleccionados, para lo cual se utiliza el método **getElementsByTagName** seleccionando los nodos de tipo **"marca"**.



## Tarea 5.2.

Basándote en el código expuesto en este apartado, carga los datos de un fichero XML con la temática que desees.





## 5.5. Comunicación asíncrona con el servidor. AJAX y jQuery.

AJAX, como ya se ha visto, es la forma que tiene JavaScript de comunicarse de forma asíncrona con un servidor. Utilizar jQuery es otra manera más de realizar peticiones por GET o POST a un servidor e incorporar el resultado a un campo de una página web.

A veces, AJAX puede cambiar, dependiendo del navegador en el que se esté utilizando. En ocasiones, el programador tiene que comprobar el tipo de navegador antes de ejecutar un código u otro.

Enlace a la web de [jQuery](#) para más información.



### 5.5.1. Carga simple de datos load().

Esta función load() permite cargar datos directamente del servidor en un campo HTML que el programador decida. El formato de esta función es el siguiente:

```
jQuery.load( url [,datos][,la_función])
```


- **url** es un string que contiene la página web que se va a invocar.
- **datos** es un objeto o un string con la información que se quiere enviar al servidor.
- **la\_función** es una función que se desea invocar cuando la petición se comparte.



### 5.5.1. Carga simple de datos load().

Este es el método más sencillo de obtener datos de un servidor utilizando jQuery. Una vez que la petición se complete, la función **load()** cargará los datos recibidos en el campo que el programador decida.

```
$("#lista").load("datos.html");
```



Los datos se cargan en un campo con identificador **id=lista**. Y los datos se reciben de la página web **datos.html**.

## Ejemplo: **AJAX-jQuery**

```
<!DOCTYPE html>
<html>
<head>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js" ></script>
  <script>
    $(document).ready( function () {
      $( "#btnDatos" ).click( function () {
        $( "#lista" ).load( "datos.html" );
      });
    });
  </script>
</head>
<body>
  <h3>Asignaturas</h3>
  <ul id="lista"></ul>
  <button id="btnDatos">Carga las asignaturas</button>
</body>
</html>
```

El fichero ***datos.html*** contendrá el siguiente código HTML:

```
<!DOCTYPE html>
<html>
  <li>Desarrollo web en entorno cliente</li>
  <li>Desarrollo web en entorno servidor</li>
  <li>Programación</li>
  <li>Entornos de desarrollo</li>
  <li>Sistemas informáticos</li>
</html>
```



## Tarea 5.3.

Crea un programa basado en el código expuesto en este apartado y verifica que funciona correctamente en un navegador.



## 5.5.2. Llamada asíncrona utilizando POST.

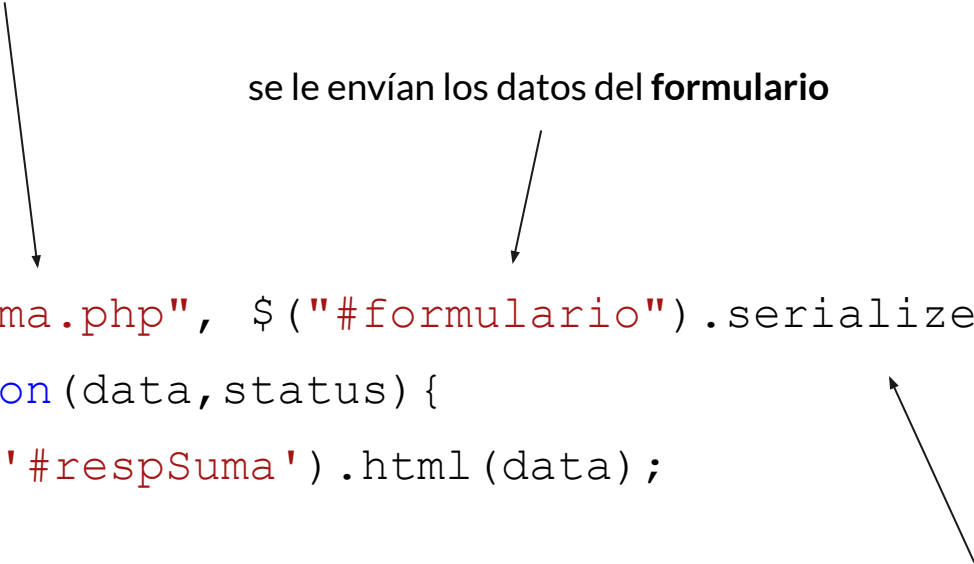
A continuación, se va a mostrar un ejemplo de llamada utilizando POST. La función POST tiene el siguiente formato:

```
jQuery.post( url [, datos] [, la_función] [, dataType])
```

- **url** es un string que contiene la página web que se va a invocar.
- **datos** es un objeto o un string con la información que se quiere enviar al servidor.
- **la\_función** es una función que se desea invocar cuando la petición se complete.
- **dataType** es el formato de datos que se espera recibir del servidor (XML, JSON, script, text o HTML).

Llamada a la página web *suma.php*

se le envían los datos del **formulario**



```
$.post("suma.php", $("#formulario").serialize(),  
function(data,status){  
    $('#respSuma').html(data);  
});
```

The diagram consists of three arrows. One arrow points from the text 'Llamada a la página web suma.php' to the string 'suma.php' in the code. A second arrow points from the text 'se le envían los datos del formulario' to the 'serialize()' method call. A third arrow points from the explanatory text block to the 'serialize()' method call.

Método ***serialize()*** para crear un string en formato URL-encoded con los datos del formulario y que pueda recibirlos correctamente el script en el lado del servidor.



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Suma dos números con AJAX y jQuery </title>
  <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js" ></script>
  <script>
    $(document).ready( function () {
      $( "#btnSuma" ).click( function () {
        $.post( "suma.php",
        $( "#formulario" ).serialize(),
        function (data,status) {
          $( '#respSuma' ).html (data);
        });
      });
    });
  </script>
</head>
```

```
<body>
  <form method="post" id="formulario">
    Introduzca dos valores para calcular la suma:<br>
    <input type="number" name="valor1" placeholder="valor1" autofocus/>
    <input type="number" name="valor2" placeholder="valor2" autofocus/>
    <input type="button" id="btnSuma" value="Suma"/>
  </form>
  <br><br>
  <div id="respSuma"></div>
</body>
</html>
```

```
<?php
```

```
    $valor1 = $_POST['valor1'];
```

```
    $valor2 = $_POST['valor2'];
```

```
    $suma = $valor1+$valor2;
```

```
    echo "La suma es: ".$suma;
```

```
?>
```

archivo: ***“suma.php”***



## Tarea 5.4.

¿Serías capaz de conseguir que el ejercicio expuesto en este apartado pudiese realizar la suma o la resta?



## 5.6. Práctica guiada: encuestas interactivas con AJAX.

En este ejemplo, se va a crear una encuesta interactiva utilizando AJAX. La encuesta hará una pregunta al usuario y mostrará el resultado total de todas las preguntas realizadas hasta el momento.

La página web consta de tres partes:

- Un fichero de texto (*resultados.txt*). Donde se almacenan los resultados.
- Una página html (*encuesta.html*). Donde se genera la encuesta.
- Una página php (*encuesta\_voto.php*). Recoge el voto del usuario y calcula los tantos por ciento teniendo en cuenta los datos anteriores, además de publicar el resultado.



## 5.6. Práctica guiada: encuestas interactivas con AJAX.

### 1. El fichero de texto (*resultados.txt*).

Es donde se van a almacenar los resultados de la encuesta. Se va a llamar resultados.txt y se va a iniciar con la siguiente línea de texto:

```
0 || 0 || 0 || 0
```

Cada uno de esos campos separados por || serán los votos que obtendrá cada una de las opciones. Conforme los usuarios vayan votando, el contenido del fichero irá cambiando:

```
19 || 3 || 6 || 0
```



## 5.6. Práctica guiada: encuestas interactivas con AJAX.

### 2. La página web (*encuesta.html*).

La página web para la encuesta es sumamente sencilla. Consta de cuatro **input** de tipo radio que, al ser seleccionados, ejecutarán la función **getvoto()**. Esta función realiza una llamada AJAX a la página web php *encuesta\_voto.php* pasándole como parámetro por **GET** el valor del voto emitido por el usuario.

```
<!DOCTYPE html>

<html>

<head>

    <title>Encuesta</title>

    <script>

        function getvoto(int){

            xmlhttp = new XMLHttpRequest();

            xmlhttp.onreadystatechange=function(){

                if(this.readyState==4 && this.status== 200){

document.getElementById("encuesta").innerHTML=this.responseText;

                }

            }

            xmlhttp.open("GET", "encuesta_voto.php?voto="+int,true);

            xmlhttp.send();

        }

    </script>
```



```
<body>
  <div id="encuesta">
    <h3>¿Qué equipo crees que va a ganar la liga este año? </h3>
    <form>
      Real Madrid:
        <input type="radio" name="voto" value="0"
onclick="getvoto(this.value)" ><br>
      Barcelona:
        <input type="radio" name="voto" value="1"
onclick="getvoto(this.value)" ><br>
      Atlético de Madrid:
        <input type="radio" name="voto" value="2"
onclick="getvoto(this.value)" ><br>
      Sevilla:
        <input type="radio" name="voto" value="3"
onclick="getvoto(this.value)" ><br>
    </form>
  </div>
```

<?php

Ejemplo: *encuesta\_voto.php*

```
//Se abre el fichero donde están almacenados los datos.
$fichero = "resultado.txt";
$contenido = file($fichero);
//colocamos el contenido en un array y lo almacenamos en cuatro
variables
//por equipos
$array = explode("||", $contenido[0]);
$real = $array[0];
$bar = $array[1];
$atl = $array[2];
$sev = $array[3];
//extraemos el voto de los participantes
$voto = $_GET['voto'];
```

//actualizamos los votos añadiendo el recibido a los anteriores

Ejemplo: *encuesta\_voto.php*

```
if ($voto == 0){
    $real = $real + 1;
}
if ($voto == 1){
    $bar = $bar + 1;
}
if ($voto == 2){
    $atl = $atl + 1;
}
if ($voto == 3){
    $sev = $sev + 1;
}

//creamos la cadena que se va a insertar en el fichero
$insertvoto = $real."||".$bar."||".$atl."||".$sev;

//se abre el fichero como escritura y se escriben los votos actualizados
$fp = fopen($fichero, "w");
fputs($fp,$insertvoto);
fclose($fp);
```

```
//se calcula el % del voto de cada uno de los equipos

$denominador=(int) $real+(int) $bar+(int) $atl+(int) $sev;
$tantoMadrid=100*round($real/$denominador,2);
$tantoBarcelona=100*round($bar/$denominador,2);
$tantoAtletico=100*round($atl/$denominador,2);
$tantoSevilla=100*round($sev/$denominador,2);

?>
<h2>Resultado:</h2>
    <table>
        <tr>
            <td>Real Madrid:</td>
            <td>
                "
height="20"><?php echo($tantoMadrid); ?>%
            </td>
        </tr>
```

```
<tr>
    <td>Barcelona:</td>
    <td>
        " height="20"><?php
echo($tantoBarcelona); ?>%
    </td>
</tr>
<tr>
    <td>Atlético de Madrid:</td>
    <td>
        " height="20"><?php
echo($tantoAtletico); ?>%
    </td>
</tr>
```

```
<tr>
  <td>Sevilla:</td>
  <td>
    "
height="20"><?php echo($tantoSevilla); ?>%
    </td>
  </tr>
</table>
```