UNIDAD

Uso de javascript avanzado

IES MARQUES DE COMARES

ÍNDICE:

IES MARQUES DE COMARES

1.- **INTRODUCCIÓN**

Ajax es la técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano, por lo que es posible realizar cambios sobre las páginas sin necesidad de recargarlas, sino agregando lo que necesitamos de la respuesta del servidor, mejorando la interactividad y velocidad en las aplicaciones.

IES MARQUES DE COMARES

INTRODUCCIÓN

- AJAX es un conjunto de tecnologías para el desarrollo web
- Gracias a estas tecnologías somos capaces de solicitar contenido en nuestro navegador de forma independiente, una vez ya hemos cargado la web
- AJAX es usado, no sólo en navegación de escritorio, también en frameworks

IES MARQUES DE COMARES

2.- Mapas en Javascript

Denominamos mapas en javascript a aquellos objetos que permiten crear estructuras de tipo clave-valor, sabiendo que la clave es única y que tiene asociado un valor que no tiene porqué ser único.

Los mapas se manejarán con los métodos asociados al objeto Map, destacando set, get, has, delete, keys, values

Además, vamos a poder fácilmente, a partir de mapas, crear objetos iterables (manejados con la estructura for …of) y convertirlos en arrays.

IES MARQUES DE COMARES

Para comprender los mapas, nos basaremos como ejemplo un mapa para cada uno de los módulos del 2º curso del ciclo de DAW:

```
$(function(){
//creamos el mapa (objeto Map como array)
const modulos=new Map([[1,"DIW"],[2,"DEW"],[3,"DWEC"],[4,"DWES"]]);
console.log(modulos);
//Usemos el método set para añadir los módulos de empresas y FCT al mapa
modulos.set(5,"EIEM").set(6,"FCT");
console.log(modulos);
//obtener el módulo DWEC del mapa (clave 3) , devolviéndola si existe y error en caso contrario
if (modulos.has(3))
    console.log(modulos.get(3));
else
    console.log("el módulo de clave 3 no se ha encontrado");
//Borrar el módulo de FCT
modulos.delete(6);
console.log(modulos);
//convertir el mapa módulos en el array módulos2 mediante el operador de propagación (poco interesante)
var modulos2=new Array(...modulos);
console.log(modulos2);
//recorrer mapas con el bucle for...of
for (let elemento of modulos){
    console.log(elemento);
//recorrer el mapa diviendo en clave-valor
for (let [clave,valor] of modulos)
    console.log (`Clave: ${clave}, Valor: ${valor}`);
// recorrer claves y valores por separado con los métodos keys y values
for (let clave of modulos.keys())
    console.log(clave);
for (let valor of modulos.values())
    console.log(valor);
}
});
```

3.- Control de errores

Hay que diferenciar entre Error, excepción y warning.

- **Error-> fallo que hace que la aplicación se detenga . No está controlado y provoca situaciones indeseadas.**
- **Excepción-> Error que sí se puede controlar, y por tanto, gestionarlo.**
- **Warning-> Error leve. No impide la ejecución pero avisa al programador de un aviso que debe corregirse para evitar errores complejos posteriores.**

IES MARQUES DE COMARES

Control de errores

Lo mejor en javascript para manejar el control de errores, es utilizarlo con en modo estricto, lo que se consigue con la directiva "use strict" como primera línea del archivo, lo que conlleva a detectar errores que en javascript no estricto no se detectarían. Veamos un ejemplo:

Modo no estricto: no da error

```
// modo no estricto. Usamos variable sin declarar y vemos que no detecta error
a=1;
console.log(a);
```

Modo estricto (error)

```
//modo estricto detectando errores de forma más estricta.
'use strict';
b=2;
console.log(b);
```

Además, podemos usar modo estricto solo para el ámbito de una función:

```
function nombre(parámetros){
'use strict'
....
}
```

IES MARQUES DE COMARES

Lanzando errores propios

Aunque la mayoría de los errores los crea el intérprete de Javascript, podemos crear nuestros propios errores con el objeto correspondiente y lanzarlo con la instrucción "throw". Javascript tiene 6 objetos que sirven para crear errores propios:

TIPO ERROR	USO
Error	Error genérico
EvalError	Error al intentar usar la función eval() de javascript
RangeError	Error numérico o de rango de valores incorrecto
ReferenceError	Referencia a objeto no válido
TypeError	Error por un mal uso de los tipos de datos
URlError	Error en la URL.

Ejemplo:

```
let miError=new RangeError("Se esperaba un número");
throw miError;
```

IES MARQUES DE COMARES

Gestionando excepciones (Try … Catch)

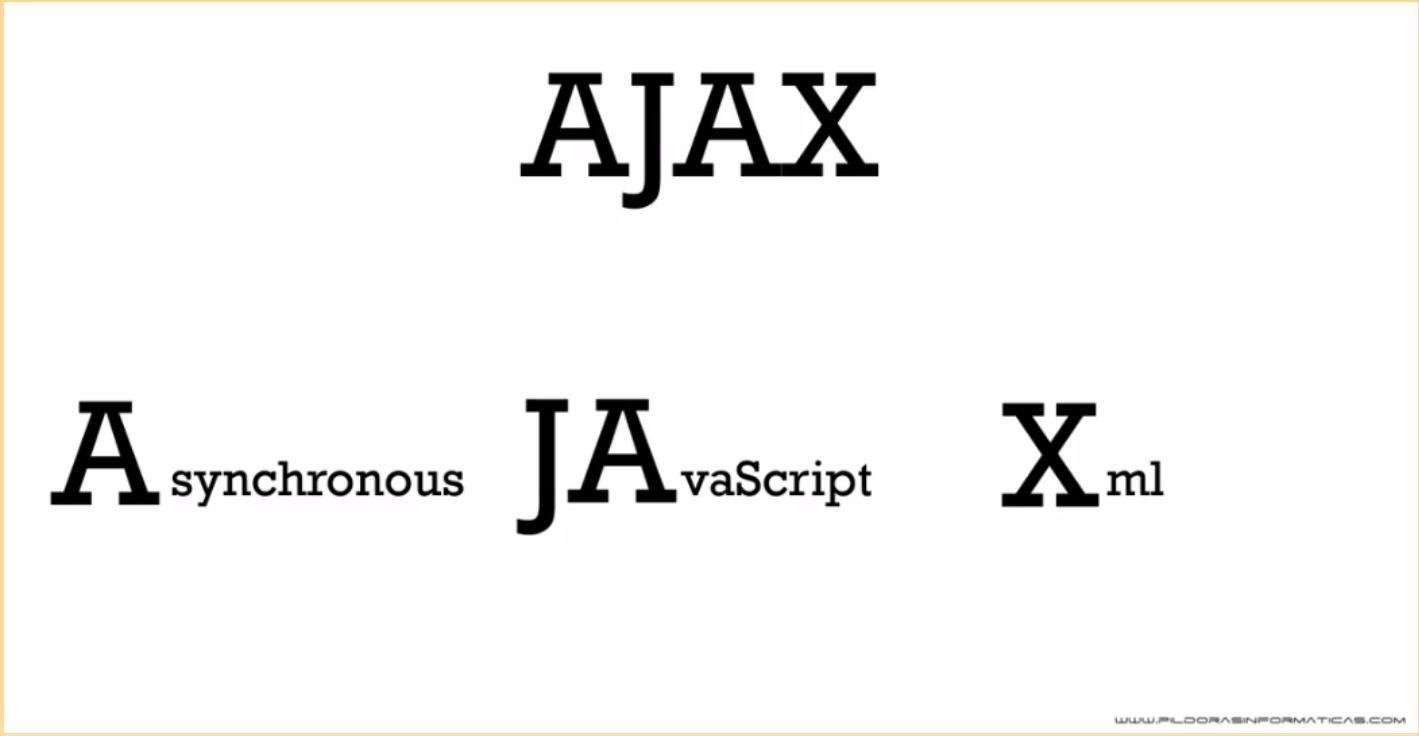
Javascript proporciona el mecanismo Try … Catch para manejar errores. En el bloque Try se coloca el código que puede provocar el error, de forma que la línea que lo provoca pasa la ejecución al bloque catch dejando en suspenso la ejecución del resto de las líneas Try hasta que el error es gestionado.

Además, podemos añadir un bloque finally cuyo contenido se muestra tanto si se produce la excepción como si no.

5.- **¿Qué es AJAX?**

En realidad, el término AJAX es un acrónimo de Asynchronous JavaScript + XML, que se puede traducir como "JavaScript asíncrono + XML“

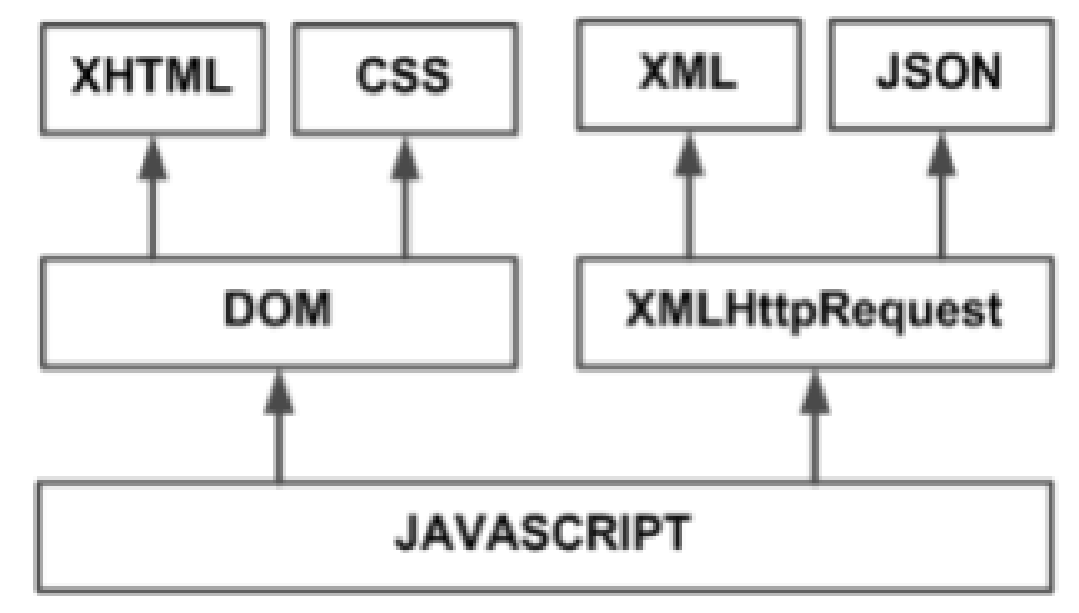
Asíncrono significa que podemos hacer una petición al servidor e inmediatamente hacer una segunda o sucesivas peticiones sin necesidad de esperar la respuesta del servidor a la primera petición o peticiones anteriores.



¿Qué es AJAX?

Las tecnologías que forman AJAX son:

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.



IES MARQUES DE COMARES

6.-VENTAJAS Y DESVENTAJAS DE AJAX

VENTAJAS

- Mostrar nuevo contenido HTML sin recargar la página.
- Enviar un formulario y mostrar la respuesta inmediatamente.
- Hacer login en una página sin abandonar la misma.
- Mostrar resultados automáticos de encuestas en formularios.
- Consultar una BBDD para obtener respuesta automáticamente.
- Mejora la experiencia del usuario
- Puede ser utilizada en cualquier plataforma y navegador
- Menor transferencia de datos cliente/servidor
- Optimización de recursos (tiempo de operaciones)
- Portabilidad y usabilidad (permite realizar una petición de datos al servidor y recibirla sin necesidad de cargar la página entera)

IES MARQUES DE COMARES

VENTAJAS Y DESVENTAJAS DE AJAX

DESVENTAJAS:

- JavaScript debe estar activado en el navegador web para funcionar
- Utilizar AJAX preferentemente en formularios de contacto, validación de correo electrónico y contraseñas para no afectar el posicionamiento web (SEO)
- Tiempos de respuesta del servidor en momentos determinados
- Tiempo de desarrollo (es necesario tener conocimiento medio/alto de las tecnologías que hacen parte de AJAX)
- Algunas funciones a las que estamos acostumbrados en la navegación web pueden no funcionar como esperamos. Por ejemplo, el botón de atrás, guardar marcador o actualizar la página web.

El punto débil de AJAX es que es necesario escribir código para cada uno de los navegadores que utilizaremos, y esto requiere de mucho tiempo.

6.-MECANISMOS ASÍNCRONOS

Son aquellos mecanismos usados en peticiones asíncronas donde hay que esperar a que termine la respuesta de la petición

6.1.- Callback

Un Callback (llamada de vuelta) es una función que se ejecutará **después** de que otra función haya terminado de ejecutarse, de aquí el nombre de Callback.

En JavaScript, las funciones admiten funciones como argumentos. Cualquier función que se pase como argumento se denomina función Callback.

Callback

Para ilustrar esto, veamos un ejemplo (Supongamos la siguiente ejecución de funciones):

```
function antes(){ console.log(“antes”);}
function despues(){ console.log(“después”);}
antes();
despues();
```

¿qué pasaría si la función antes tuviera algún tipo de código que no se puede ejecutar inmediatamente? Por ejemplo, una petición a una API, donde tenemos que enviar una petición y esperar una respuesta (para simular esto usaremos un setTimeout).

```
function antes(){
        // Simular retardo
        setTimeout( function(){console.log(“antes); }, 1000 );
        }
function despues(){ console.log(“después”);}
antes();
despues();
```

En este caso, al haber retrasado la función antes, imprimirá el mensaje “después” delante de “antes”.

Callback

Para ilustrar esto, veamos un ejemplo (Supongamos la siguiente ejecución de funciones):

```
function antes(){ console.log(“antes”);}
function despues(){ console.log(“después”);}
antes();
despues();
```

¿qué pasaría si la función antes tuviera algún tipo de código que no se puede ejecutar inmediatamente? Por ejemplo, una petición a una API, donde tenemos que enviar una petición y esperar una respuesta (para simular esto usaremos un setTimeout).

```
function antes(){
        // Simular retardo
        setTimeout( function(){console.log(“antes); }, 1000 );
        }
function despues(){ console.log(“después”);}
antes();
despues();
```

En este caso, al haber retrasado la función antes, imprimirá el mensaje “después” delante de “antes”.

IES MARQUES DE COMARES

Callback

Pues bien, los Callbacks son un modo de asegurarse de que cierto código no se ejecuta hasta que otro código haya terminado de ejecutarse.

Ejemplo: suma de 2 números usando callback

```
<input type="text" id="num1" />
<input type="text" id="num2" />
<button id="sumar"> Sumar</button>
function Sumar(num1, num2, callback){
   return callback(num1+num2);
}
 document.querySelector("#sumar").addEventListener('click', function(){
   var       num1 = parseInt(document.querySelector("#num1").value),
             num2 = parseInt(document.querySelector("#num2").value);

   Sumar(num1, num2, function(r){ console.log('El resultado es ' + r); })
})
```

6.2.- Promesas

El problema de las callbacks surge cuando debemos encadenar una tras otra porque necesitamos que se ejecute una acción cuando haya terminado la anterior y así sucesivamente. En este caso, otro problema es que para manejar el error, habría que implementarlo en cada Callback, por lo que, para evitar este último problema surgen las promesas.

6.2.- Promesas

Las promesas trabajan con dos recursos principales: resolve (éxito) y reject (error). Por tanto, las promesas podemos verlas como in if-then-else, donde si se cumple la petición se ejecutaría el resolve y si la petición falla se ejecutaría el reject.

Las promesas tienen dos métodos para ejecutar la sincronía:

- Método then (se ejecuta cuando se ha ejecutado el código de la promesa con éxito y recibe lo que devuelva resolve). Podemos tener tantos métodos then como queramos, de forma que cada método then trabaja con la respuesta del then anterior.
- Método catch (error, si no se cumple la promesa).

Además, de esta forma podemos tratar todos los errores unificados en el catch.

IES MARQUES DE COMARES

Ejemplo anterior de sumar dos números con promesas:

```
function sumarPromise(num1,num2) {
                if (typeof num1 !== "number" || typeof num2 !== "number") {
                    //devolvemos mensaje de error
                return Promise.reject("Error, alguno de los valores no son un número");
                }
                return new Promise((resolve, reject) => {
                      //vamos a devolver un objeto con los valores a sumar y el resultado
                resolve({
                    num1: num1,
                    num2: num2,
                    suma: num1+num2
                    });
                });
            }

sumarPromise(1,2)
                .then(objeto => {
                console.log(objeto);
                console.log('Inicio Promise');
                console.log(`Promise: ${objeto.num1}, ${objeto.num2},${objeto.suma}`);
                return sumarPromise(3,4);
                })
                .then(objeto => {
                    console.log(`Promise: ${objeto.num1}, ${objeto.num2},${objeto.suma}`);
                    console.log(" Fin promesa");
                })
                .catch(error => console.error(error));
```

6.3.- Funciones asíncronas

Son aquellas que esperan a que se cumpla algo para ejecutarse. Las funciones asíncronas no vienen a reemplazar las promesas, sino que pueden trabajar en colaboración con ellas.

Las funciones asíncronas se declaran con la palabra async delante el nombre de la función.

Ej: async funcionAsincronaDeclarada(){ }.

También podemos usar una función asíncrona como una función expresada.

Ej: const funcionAsincronaExpresada = async ( ) => { }

Para el manejo de errores dentro de las funciones asíncronas podemos utilizar el try-catch.

IES MARQUES DE COMARES

Ejemplo anterior de sumar dos números con promesas:

```
function sumarPromise(num1,num2) {
                if (typeof num1 !== "number" || typeof num2 !== "number") {
                    //devolvemos mensaje de error
                return Promise.reject("Error, alguno de los valores no son un número");
                }
                return new Promise((resolve, reject) => {
                      //vamos a devolver un objeto con los valores a sumar y el resultado
                resolve({
                    num1: num1,
                    num2: num2,
                    suma: num1+num2
                    });
                });
            }
    async function funcionAsincronaDeclarada() {
      try {
        console.log('Inicio función asíncrona');

        let objeto = await sumarPromise(1,2);
        console.log(`Función asíncrona: ${objeto.num1},${objeto.num2}, ${objeto.suma}`);

        objeto = await sumarPromise(3,4);
        console.log(`Función asíncrona: ${objeto.num1},${objeto.num2}, ${objeto.suma}`);
        console.log('Fin función asíncrona');
      } catch (err) {
        console.error(`se ha producido el error ${err}`);
      }
    }
    funcionAsincronaDeclarada();
```

7.- EJEMPLOS PETICIONES ASÍNCRONAS

(XHMLHttpRequest, Fetch, Axios y API REST)

Descargar el código fuente ejemplos_asíncronos.zip de la plataforma moodle

7.- EJEMPLOS de CRUD

Descargar el código fuente ejemplos_crud.zip de la plataforma moodle

Propiedades del objeto XHMLHttpRequest(XHR)

Propiedad	Descripción	Observaciones / Ejemplo
onreadystatechange	Sirve para definir una función que será llamada automáticamente cada vez que cambie la propiedad readyState del objeto.	Puede definirse directamente como función anónima: xmlhttp.onreadystatechange = function() { …} O indicando el nombre de función a ejecutar: xmlhttp.onreadystatechange = ejecutarRespuesta;
readyState	Contiene un valor numérico entero que representa la situación del intercambio de datos a través del objeto.	if (xmlhttp.readyState==4) indica que se ha recibido la información solicitada del servidor. Puede tomar los siguientes valores: **0: no inicializado.** Indica que no se ha abierto la conexión con el servidor (no se ha llamado a open) **1: conexión con servidor establecida.** Indica que se ha abierto la conexión pero todavía no se ha enviado la petición (no se ha llamado a send) **2: recibida petición en servidor.** Indica que el servidor ya ha recibido la petición (se ha llamado a send) **3: enviando información.** Se está enviando la información por parte del servidor, todavía no se ha completado la recepción. **4: completado.** Se ha recibido la información del servidor y está lista para operar con ella.

IES MARQUES DE COMARES

Propiedades del objeto XHMLHttpRequest(XHR)

Propiedad	Descripción	Observaciones / Ejemplo
responseText	Una vez completada la comunicación, responseText contiene la respuesta del servidor en forma de cadena de texto	contenidosRecibidos = xmlhttp.responseText.split(","); En este ejemplo la respuesta en forma de texto separado por comas se traslada a los elementos de un array.
responseXML	Una vez completada la comunicación, responseText contiene la respuesta del servidor en formato XML	Si el servidor no ha respondido en formato XML al tratar de recuperar responseXML obtendremos null. La respuesta es un documento o nodo con estructura DOM. Veremos ejemplos más adelante.
status	Código numérico entero enviado por el servidor que indica el tipo de respuesta dada a la petición. Puede tomar valores como: 200: respuesta correcta. 404: no encontrado. 500: error interno del servidor.	if (xmlhttp.readyState==4 && xmlhttp.status==200) El código anterior es un condicional asociado a si se ha completado el intercambio de información y si el servidor ha devuelto un código de respuesta correcta.
statusText	Equivalente a status pero en forma de cadena de texto "OK": respuesta correcta. "Not found": no encontrado.	alert ('Respuesta server: ' + xmlhttp.statusText);

Propiedades del objeto XHMLHttpRequest(XHR)

Propiedad	Descripción	Observaciones / Ejemplo
bort()	Cancela la petición en curso	Puede usarse para abortar el proceso que se está produciendo en segundo plano. Por ejemplo, si el proceso es la subida de un fichero al servidor y el usuario pulsa un botón de cancelar ejecutaríamos: xmlhttp.abort();
open (método, url)	Realiza una petición de apertura de comunicación con un método que puede ser principalmente GET o POST.	Con GET los parámetros de la petición van incluidos en la url. Con POST los parámetros de la petición van en las cabeceras de HTTP. url puede ser una ruta relativa o completa. xmlhttp.open("GET","datos.php?pais=spain");
open (método, url, async, userName, password)	Los parámetros async, userName y password son opcionales. El parámetro async indica si la petición debe ser gestionada asíncronamente. Un valor true implica que el procesamiento del script continuará después de invocarse send() sin bloquear la navegación aún no disponiendo de la respuesta completa del servidor.	El valor por defecto para async es true. Esto indica “procesar en segundo plano sin bloquear la navegación”. Si se establece a false, se bloquearía la navegación hasta obtener respuesta. Esto en general es indeseable, pero puede ser necesario por ejemplo si se requiere usuario y contraseña que han de verificarse antes de proseguir. El userName y password serían nombre de usuario y password necesarios en cada caso para acceder al recurso solicitado. xmlhttp.open("GET","datos.php?pais=spain",

IES MARQUES DE COMARES

Propiedades del objeto XHMLHttpRequest(XHR)

Propiedad	Descripción	Observaciones / Ejemplo
abort()	Cancela la petición en curso	Puede usarse para abortar el proceso que se está produciendo en segundo plano. Por ejemplo, si el proceso es la subida de un fichero al servidor y el usuario pulsa un botón de cancelar ejecutaríamos: xmlhttp.abort();
open (método, url)	Realiza una petición de apertura de comunicación con un método que puede ser principalmente GET o POST.	Con GET los parámetros de la petición van incluidos en la url. Con POST los parámetros de la petición van en las cabeceras de HTTP. url puede ser una ruta relativa o completa. xmlhttp.open("GET","datos.php?pais=spain");
open (método, url, async, userName, password)	Los parámetros async, userName y password son opcionales. El parámetro async indica si la petición debe ser gestionada asíncronamente. Un valor true implica que el procesamiento del script continuará después de invocarse send() sin bloquear la navegación aún no disponiendo de la respuesta completa del servidor.	El valor por defecto para async es true. Esto indica “procesar en segundo plano sin bloquear la navegación”. Si se establece a false, se bloquearía la navegación hasta obtener respuesta. Esto en general es indeseable, pero puede ser necesario por ejemplo si se requiere usuario y contraseña que han de verificarse antes de proseguir. El userName y password serían nombre de usuario y password necesarios en cada caso para acceder al recurso solicitado. xmlhttp.open("GET","datos.php?pais=spain", false, "admin", "pswd");

IES MARQUES DE COMARES

Propiedades del objeto XHMLHttpRequest(XHR)

Propiedad	Descripción	Observaciones / Ejemplo
send()	Envía la petición al servidor	xmlhttp.send();
send(cadena)	Envía la petición al servidor incluyendo datos en una cadena de texto, normalmente asociado al envío de datos mediante POST	xmlhttp.send("nombre=Juan&edad=33");
setRequestHeader (cabecera, valor)	Añade un par cabecera – valor a la cabecera HTTP. Necesario para pasar datos por POST.	xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
getRespondeHeader (cabecera)	Devuelve una cadena con el contenido de la cabecera solicitada	var contType = xmlhttp.getResponseHeader ("Content-Type");
getAllResponseHeaders()	Devuelve una cadena de texto con todas las cabeceras de la respuesta del servidor	var headers = xmlhttp.getAllResponseHeaders ();

IES MARQUES DE COMARES

13.-AJAX. Objetos JSON

Ya hemos visto antes que cuando hacemos una petición asíncrona al servidor, éste nos tiene que devolver esos datos que necesitamos en algún formato determinado como texto, xml, json,etc.

En este caso, vamos a ver cómo javascript procesa los datos cuando el servidor los devuelve en formato JSON