Contenido

[¿Qué es JSON, para qué sirve y dónde se usa? 1](#_Toc40037749)

[JSON Nombre/Par de Valores 1](#_Toc40037750)

[Valores Json 1](#_Toc40037751)

[Objetos JSON 1](#_Toc40037752)

[Arrays JSON 2](#_Toc40037753)

[Cómo se forma una cadena de JSON 2](#_Toc40037754)

[Escribir JSON en PHP 5](#_Toc40037755)

[Leer JSON en PHP 5](#_Toc40037756)

[Ajax con JSON y jQuery 6](#_Toc40037757)

# ¿Qué es JSON, para qué sirve y dónde se usa?

**JSON (JavaScript Object Notation),** y aunque su nombre lo diga, no es necesariamente parte de JavaScript, de hecho es un estándar basado en texto plano para el intercambio de información, por lo que se usa en muchos sistemas que requieren mostrar o enviar información para ser interpretada por otros sistemas, la ventaja de JSON al ser un formato que es independiente de cualquier lenguaje de programación, es que los servicios que comparten información por éste método, no necesitan hablar el mismo idioma, es decir, el emisor puede ser Java y el receptor PHP, cada lenguaje tiene su propia librería para codificar y decodificar cadenas de JSON.

* JSON es sólo un formato de datos - contiene sólo propiedades, no métodos.
* JSON requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas.
* Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione. Se debe ser cuidadoso para validar cualquier dato que se utilizar (aunque los JSON generados por computador tienen menos probabilidades de tener errores, mientras el programa generador trabaje adecuadamente). Es posible validar JSON utilizando una aplicación como JSONLint.
* JSON Puede tomar la forma de cualquier tipo de datos que sea válido para ser incluido en un JSON, no sólo arreglos u objetos. Así, por ejemplo, una cadena o un número único podrían ser objetos JSON válidos.
* A diferencia del código JavaScript en que las propiedades del objeto pueden no estar entre comillas, en JSON, sólo las cadenas entre comillas pueden ser utilizadas como propiedades.

# JSON Nombre/Par de Valores

Para asignar a un nombre un valor debemos usar los dos puntos ':' este separador es el equivalente al igual ('=') de cualquier lenguaje.

"Nombre" : "Geeky Theory"

# Valores Json

Los tipos de valores que podemos encontrar en Json son los siguientes:

• Un número (entero o float)

• Un string (entre comillas simples)

• Un booleano (true o false)

• Un array (entre corchetes [] )

• Un objeto (entre llaves {})

• Null

# Objetos JSON

Los objetos JSON se identifican entre llaves, un objeto puede ser, por ejemplo, una fruta y su cantidad disponible:

{ "NombreFruta":"Manzana" , "Cantidad":20 }

# Arrays JSON

En un Json puedes incluir arrays, para ello el contenido del array debe ir entre corchetes [].

En este caso tenemos un array de objetos fruta:

{

"Frutas": [

{ "NombreFruta":"Manzana" , "cantidad":10 },

{ "NombreFruta":"Pera" , "cantidad":20 },

{ "NombreFruta":"Naranja" , "cantidad":30 }

]

}

# Cómo se forma una cadena de JSON

El principio básico, como ya hemos visto, es con pares atributo-valor, éstos deben estar encerrados entre llaves { , } que es lo que definen el inicio y el fin del objeto.

//Ejemplo de JSON para un objeto tipo Persona

{

"nombre":"Fulano Probencio",

"edad":27,

"nacionalidad":"Chileno",

"altura":"172 cm",

"peso":75

}

Hasta ahora un objeto JSON no es gran complicación, pero ahora pasaremos a esa parte ya que como mencioné anteriormente, un valor que puede almacenar JSON son arreglos/arrays e incluso otros objetos, comencemos primero con un ejemplo de un array, continuemos con el objeto anterior de persona, agreguémosle ahora el atributo pasatiempos, el cual será un listado de hobbies que tiene nuestro amigo Fulano.

{

"nombre":"Fulano Probencio",

"edad":27,

"nacionalidad":"Chileno",

"altura":"172 cm",

"peso":75,

"pasatiempos":["Polo","Cricket","Ski","Drafting","Gaming"]

}

El nuevo atributo pasatiempos tiene como valor un arreglo/array, para definir un array, debemos listar sus elementos separados por coma y encerrados entre corchetes [ , ], sus valores pueden ser cualquier valor aceptado en JSON y en el lenguaje emisor/receptor (numéricos, textos, booleanos, incluso otros arreglos y otros objetos).

Con estos principios podemos comenzar a crear nuestras cadenas de JSON, aunque básicas, ya podríamos interpretarlas y armarlas con cierta facilidad. Algunos editores de código como NetBeans, Dreamweaver, Textpad, etc. tienen incorporados plugins (o se les puede descargar) que nos ayudan a dar formato y validar nuestros objetos JSON, aunque si tienen prisa o no cuentan con ese tipo de herramientas pueden optar por soluciones en línea como JSONLint, un validador de JSON en línea que nos da formato y valida los objetos que ingresemos en su área de texto.

Por último, una nota, cuando se manda o recibe un objeto JSON, siempre se maneja un sólo objeto a la vez, es decir, no podemos manejar más de un objeto pegado con otro, aunque los separemos por comas, pero sí es posible enviar/recibir un gran objeto JSON que tenga un valor con muchos objetos en su interior, por ejemplo, si quisiera enviar varios objetos persona de una sola vez, haría lo siguiente

{

    "personas": [

        {

            "nombre": "Fulano Probencio",

            "edad": 27,

            "nacionalidad": "Chileno",

            "altura": "172 cm",

            "peso": 75,

            "pasatiempos": [

                "Polo",

                "Cricket",

                "Ski",

                "Drafting",

                "Gaming"

            ],

            "soltero": true,

            "direccion": {

                "calle": "Ave. Siempre Viva",

                "numero": "123",

                "pais": "México"

            }

        },

        {

            "nombre": "Otro Fulano",

            "edad": 22,

            "nacionalidad": "Argentino",

            "altura": "178 cm",

            "peso": 72,

            "pasatiempos": [

                "Pintar",

                "Programar",

                "Scuba Diving"

            ],

            "soltero": true,

            "direccion": {

                "calle": "Ave. Siempre Viva",

                "numero": "123",

                "pais": "México"

            }

        }

        .. .. .. .. ..

    ]

}

**Ejercicio:**

Imaginemos que tenemos una *frutería* y que queremos obtener el *nombre* y la *cantidad* de *fruta* y *verdura* que tenemos. En un principio vamos a suponer que tenemos lo siguiente:

 - Fruta:

* 10 manzanas
* 20 Peras
* 30 Naranjas

 - Verduras

* 80 lechugas
* 15 tomates
* 50 pepinos

Solución:

{"Fruteria":

[

{"Fruta":

[

{"Nombre":"Manzana","Cantidad":10},

{"Nombre":"Pera","Cantidad":20},

{"Nombre":"Naranja","Cantidad":30}

]

},

{"Verdura":

[

{"Nombre":"Lechuga","Cantidad":80},

{"Nombre":"Tomate","Cantidad":15},

{"Nombre":"Pepino","Cantidad":50}

]

}

]

}

Como podemos observar, hemos creado un objeto llamado frutería y, dentro de ese objeto hemos almacenado un array de dos elementos. El primer elemento del array contiene un objeto llamado fruta y el segundo elemento del array contiene otro objeto llamado verdura. Estos objetos a su vez contienen un array cuyo contenido es el nombre y la cantidad de cada fruta o verdura.

Imaginemos que nos gustaría saber la cantidad de manzanas que tenemos. El path de este array sería el siguiente:

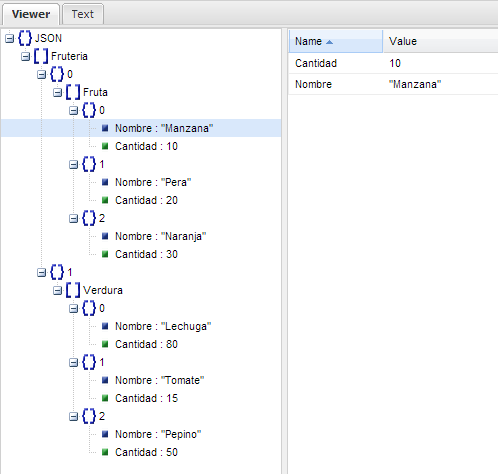
Path: json['Fruteria'][0]['Fruta'][0]['Cantidad']

Observamos que la cantidad de manzanas se almacena dentro del primer elemento del array que contiene el objeto Frutería, y a su vez dentro del primer elemento del array que contiene el objeto Fruta.

Todo esto parece un poco confuso, pero una vez que se domina veremos que tenemos ante nosotros una gran herramienta de desarrollo.

Existen herramientas online que ayudan a visualizar mejor un JSON. Una de las mejores herramientas que he visto es la página JSON Viewer. Si introducimos nuestro ejemplo observamos lo siguiente:

<http://jsonviewer.stack.hu/>



# Escribir JSON en PHP

En cualquier instalación estándar de PHP, desde la versión 5.2.0, se incorpora de forma predeterminada la extensión JSON y es muy fácil pasar los datos de un array a notación en JSON. Aunque si no se cuenta con esta extensión se puede construir la notación JSON de forma manual, es recomendable tenerla si vas a trabajar de forma frecuente con datos en formato JSON.

Entre las funciones de la extensión JSON para PHP que nos interesan, la más imprescindible es la función json\_encode(). Con esta función podemos pasar nuestros datos a formado JSON rápida y fácilmente. Sólo es necesario tener los datos en forma de array u objeto.

Por ejemplo:

<?php

$jsondata = array();

if(isset($\_GET['param']) ) {

if( $\_GET['param'] == 'valor' ) {

$jsondata['success'] = true;

$jsondata['message'] = 'Hola! El valor recibido es correcto.';

} else {

$jsondata['success'] = false;

$jsondata['message'] = 'Hola! El valor recibido no es correcto.';

}

//Aunque el content-type no sea un problema en la mayoría de casos, es

// recomendable especificarlo

header('Content-type: application/json; charset=utf-8');

echo json\_encode($jsondata);

exit();

}

# Leer JSON en PHP

A la hora de leer un JSON en PHP normalmente lo recibiremos como un string y tendremos que hacerle la conversión a objetos, a no ser que usemos una función que directamente no hagas esa conversión y podremos trabajar entonces directamente con el objeto.

Funciones útiles para lectura de JSON:

**php://input:** esta es una secuencia de solo lectura que nos permite leer datos sin procesar del cuerpo de la solicitud. Básicamente recoge los datos que se hayan enviado en la petición a nuestro archivo php y los guarda en una variable.

Este código habría que ponerlo en el archivo de recepción de PHP.

Es útil cuando desde un archivo queremos recoger el JSON que nos han enviado.

**file\_get\_contents():** esta función en PHP se usa para leer un archivo en una cadena.

**json\_decode():** esta función toma una cadena JSON y la convierte en una variable PHP que puede ser una matriz o un objeto.

Ejemplo:

$json = file\_get\_contents('php://input'); //leemos JSON

$data = json\_decode($json); // pasamos de string a object

# Usando AJAX con JSON y jQuery

Uno de los argumentos del método $.ajax() es dataType y aquí podemos especificar que vamos a utilizar datos JSON en la respuesta. Si no especificamos nada en dataType, jQuery intentará detectar de forma automática el formato de los datos recibidos. Si en la respuesta desde el servidor se especifica, como hicimos en el ejemplo anterior, el tipo de contenido será más fácil para jQuery tratar los datos correctamente.

Utilizando $.ajax() tendría un aspecto similar a:

$.ajax({

// En data puedes utilizar un objeto JSON, un array o un query string

data: {"parametro1": "valor1", "parametro2": "valor2"},

//Cambiar a type: POST si necesario

type: "GET",

// Formato de datos que se espera en la respuesta

dataType: "json",

// URL a la que se enviará la solicitud Ajax

url: "script.php",

})

.done(function( data, textStatus, jqXHR ) {

if ( console && console.log ) {

console.log( "La solicitud se ha completado correctamente." );

}

})

.fail(function( jqXHR, textStatus, errorThrown ) {

if ( console && console.log ) {

console.log( "La solicitud a fallado: " + textStatus);

}

});

Además del método anterior, podemos utilizar el atajo getJSON(), que quedaría de esta forma para hacer exactamente lo mismo:

$.getJSON( "script.php", { "parametro1" : "valor1", "parametro2" : "valor2" } )

.done(function( data, textStatus, jqXHR ) {

if ( console && console.log ) {

console.log( "La solicitud se ha completado correctamente." );

}

})

.fail(function( jqXHR, textStatus, errorThrown ) {

if ( console && console.log ) {

console.log( "Algo ha fallado: " + textStatus" );

}

});

Ten en cuenta que .getJSON(), como su nombre indica, siempre hace una solicitud con el método GET, si necesitas enviar la solicitud mediante método POST tendrás que utilizar otras funciones de jQuery como .ajax() con el argumento type puesto a POST o su alias .post().

Por ejemplo:

$.ajax({

data: {"parametro1" : "valor1", "parametro2" : "valor2"},

type: "POST",

dataType: "json",

url: "script.php",

})

.done(function( data, textStatus, jqXHR ) {

if ( console && console.log ) {

console.log( "La solicitud se ha completado correctamente." );

}

})

.fail(function( jqXHR, textStatus, errorThrown ) {

if ( console && console.log ) {

console.log( "La solicitud a fallado: " + textStatus);

}

});

//Equivalente a lo anterior

$.post( "script.php", { "parametro1" : "valor1", "parametro2" : "valor2" }, null, "json" )

.done(function( data, textStatus, jqXHR ) {

if ( console && console.log ) {

console.log( "La solicitud se ha completado correctamente." );

}

})

.fail(function( jqXHR, textStatus, errorThrown ) {

if ( console && console.log ) {

console.log( "La solicitud a fallado: " + textStatus);

}

});

En el ejemplo PHP anterior, en el objeto JSON había dos miembros: success y message. Un posible objeto JSON recibido podría ser el siguiente:

{"success" : true, "message" : "Hola! El valor recibido es correcto."}

Este objeto es recibido en el método .done() a través del argumento data y podemos acceder al valor de cada miembro del objeto JSON del siguiente modo:

.done( function(data) {

data.success;

data.message;

});