## **What is JSON?**

* JSON stands for **J**ava**S**cript **O**bject **N**otation
* JSON is a lightweight data-interchange format
* JSON is "self-describing" and easy to understand
* JSON is language independent

\*  
JSON uses JavaScript syntax, but the JSON format is text only.  
Text can be read and used as a data format by any programming language.

## **Why use JSON?**

Since the JSON format is text only, it can easily be sent to and from a server, and used as a data format by any programming language.

JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects:

JSON.parse()

So, if you receive data from a server, in JSON format, you can use it like any other JavaScript object.

## **JSON Syntax Rules**

JSON syntax is derived from JavaScript object notation syntax:

* Data is in name/value pairs
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

## **JSON - Evaluates to JavaScript Objects**

The JSON format is almost identical to JavaScript objects.

In JSON, keys must be strings, written with double quotes:

### JSON

{ "name":"John" }

In JavaScript, keys can be strings, numbers, or identifier names:

### JavaScript

{ name:"John" }

## **JSON Values**

In **JSON**, values must be one of the following data types:

* a string
* a number
* an object (JSON object)
* an array
* a boolean
* null

In **JavaScript** values can be all of the above, plus any other valid JavaScript expression, including:

* a function
* a date
* undefined

***JSON values cannot be one of the following data types:***

* ***a function***
* ***a date***
* ***undefined***

In JSON, *string values* must be written with double quotes:

JSON

{ "name":"John" }

In JavaScript, you can write string values with double *or* single quotes:

JavaScript

{ name:'John' }

## **JSON Uses JavaScript Syntax**

Because JSON syntax is derived from JavaScript object notation, very little extra software is needed to work with JSON within JavaScript.

With JavaScript you can create an object and assign data to it, like this:

### Example

var person = { name: "John", age: 31, city: "New York" };

You can access a JavaScript object like this:

### Example

// returns John  
person.name;

It can also be accessed like this:

### Example

// returns John  
person["name"];

Data can be modified like this:

### Example

person.name = "Gilbert";

It can also be modified like this:

### Example

person["name"] = "Gilbert";

You will learn how to convert JavaScript objects into JSON later in this tutorial.

## **JavaScript Arrays as JSON**

The same way JavaScript objects can be used as JSON, JavaScript arrays can also be used as JSON.

You will learn more about arrays as JSON later in this tutorial.

## **JSON Files**

* The file type for JSON files is ".json"
* The MIME type for JSON text is "application/json"

# **JSON vs XML**

Both JSON and XML can be used to receive data from a web server.

The following JSON and XML examples both define an employees object, with an array of 3 employees:

### JSON Example

{"employees":[  
  { "firstName":"John", "lastName":"Doe" },  
  { "firstName":"Anna", "lastName":"Smith" },  
  { "firstName":"Peter", "lastName":"Jones" }  
]}

### XML Example

<employees>  
  <employee>  
    <firstName>John</firstName> <lastName>Doe</lastName>  
  </employee>  
  <employee>  
    <firstName>Anna</firstName> <lastName>Smith</lastName>  
  </employee>  
  <employee>  
    <firstName>Peter</firstName> <lastName>Jones</lastName>  
  </employee>  
</employees>

## **JSON is Like XML Because**

* Both JSON and XML are "self describing" (human readable)
* Both JSON and XML are hierarchical (values within values)
* Both JSON and XML can be parsed and used by lots of programming languages
* Both JSON and XML can be fetched with an XMLHttpRequest

## **JSON is Unlike XML Because**

* JSON doesn't use end tag
* JSON is shorter
* JSON is quicker to read and write
* JSON can use arrays

The biggest difference is:

 XML has to be parsed with an XML parser. JSON can be parsed by a standard JavaScript function.

## **Why JSON is Better Than XML**

XML is much more difficult to parse than JSON.  
JSON is parsed into a ready-to-use JavaScript object.

For AJAX applications, JSON is faster and easier than XML:

Using XML

* Fetch an XML document
* Use the XML DOM to loop through the document
* Extract values and store in variables

Using JSON

* Fetch a JSON string
* JSON.Parse the JSON string

## **JSON Strings**

Strings in JSON must be written in double quotes.

### Example

{ "name":"John" }

## **JSON Numbers**

Numbers in JSON must be an integer or a floating point.

### Example

{ "age":30 }

## **JSON Objects**

Values in JSON can be objects.

### Example

{  
"employee":{ "name":"John", "age":30, "city":"New York" }  
}

## **JSON Arrays**

Values in JSON can be arrays.

### Example

{  
"employees":[ "John", "Anna", "Peter" ]  
}

## **JSON Booleans**

Values in JSON can be true/false.

### Example

{ "sale":true }

## **JSON null**

Values in JSON can be null.

### Example

{ "middlename":null }

**JSON.parse()**

인자로 전달된 문자열을 자바스크립트의 데이터로 변환한다.

# **JSON.parse()**

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

**Parse the data with JSON.parse(), and the data becomes a JavaScript object.**

## **Example - Parsing JSON**

Imagine we received this text from a web server:

'{ "name":"John", "age":30, "city":"New York"}'

Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

var obj = JSON.parse('{ "name":"John", "age":30, "city":"New York"}');

Make sure the text is written in JSON format, or else you will get a syntax error.

Use the JavaScript object in your page:

### Example

<p id="demo"></p>   
  
<script>  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;   
</script>

<!DOCTYPE html>

<html>

<body>

<h2>Create Object from JSON String</h2>

<p id="demo"></p>

<script>

var txt = '{"name":"John", "age":30, "city":"New York"}' //JSON 문자를 Javascript 데이터로 바꿈

var obj = JSON.parse(txt);

document.getElementById("demo").innerHTML = obj.name + ", age " + obj.age + " lives in " + obj.city;

</script>

</body>

</html>

## **Exceptions**

### Parsing Dates

Date objects are not allowed in JSON.

If you need to include a date, write it as a string.

You can convert it back into a date object later:

### Example

Convert a string into a date:

var text = '{ "name":"John", "birth":"1986-12-14", "city":"New York"}';  
var obj = JSON.parse(text);

obj.birth = new Date(obj.birth); //Date 라는 기능으로 문자를 날짜로 바꿈  
  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;

Parsing Functions

Functions are not allowed in JSON.

If you need to include a function, write it as a string.

You can convert it back into a function later:

Example

Convert a string into a function:

var text = '{ "name":"John", "age":"function () {return 30;}", "city":"New York"}';  
var obj = JSON.parse(text);  
obj.age = eval("(" + obj.age + ")"); // 괄호  
  
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();

You should avoid using functions in JSON, the functions will lose their scope, and you would have to use eval() to convert them back into functions.

**JSON.stringify()**

인자로 전달된 자바스크립트의 데이터를 문자열로 변환한다.

A common use of JSON is to exchange data to/from a web server.

When sending data to a web server, the data has to be a string.

Convert a JavaScript object into a string with JSON.stringify().

## **Stringify a JavaScript Object**

Imagine we have this object in JavaScript:

var obj = { name: "John", age: 30, city: "New York" };

Use the JavaScript function JSON.stringify() to convert it into a string.

var myJSON = JSON.stringify(obj);

The result will be a string following the JSON notation.

myJSON is now a string, and ready to be sent to a server:

### Example

var obj = { name: "John", age: 30, city: "New York" };  
var myJSON = JSON.stringify(obj);  
document.getElementById("demo").innerHTML = myJSON;

## **Stringify a JavaScript Array**

It is also possible to stringify JavaScript arrays:

Imagine we have this array in JavaScript:

var arr = [ "John", "Peter", "Sally", "Jane" ];

Use the JavaScript function JSON.stringify() to convert it into a string.

var myJSON = JSON.stringify(arr);

The result will be a string following the JSON notation.

myJSON is now a string, and ready to be sent to a server:

### Example

var arr = [ "John", "Peter", "Sally", "Jane" ];  
var myJSON = JSON.stringify(arr);  
document.getElementById("demo").innerHTML = myJSON;

## **Exceptions**

### Stringify Dates

In JSON, date objects are not allowed. The JSON.stringify() function will convert any dates into strings.

### Example

var obj = { name: "John", today: new Date(), city : "New York" };  
var myJSON = JSON.stringify(obj);  
  
document.getElementById("demo").innerHTML = myJSON;

## **Object Syntax**

### Example

{ "name":"John", "age":30, "car":null }

JSON objects are surrounded by curly braces {}.

JSON objects are written in key/value pairs.

Keys must be strings, and values must be a valid JSON data type (string, number, object, array, boolean or null).

Keys and values are separated by a colon.

Each key/value pair is separated by a comma.

## **Accessing Object Values**

You can access the object values by using dot (.) notation:

### Example

myObj = { "name":"John", "age":30, "car":null };  
x = myObj.name;

You can also access the object values by using bracket ([]) notation:

### Example

myObj = { "name":"John", "age":30, "car":null };  
x = myObj["name"]; // [ ] -> index value result will be “John”

## **Looping an Object**

You can loop through object properties by using the for-in loop:

### Example

myObj = { "name":"John", "age":30, "car":null };  
for (x in myObj) {  
  document.getElementById("demo").innerHTML += x;  
}

In a for-in loop, use the bracket notation to access the property values:

### Example

myObj = { "name":"John", "age":30, "car":null };  
for (x in myObj) {  
  document.getElementById("demo").innerHTML += myObj[x];  
}

## **Nested JSON Objects**

Values in a JSON object can be another JSON object.

### Example

myObj = {  
  "name":"John",  
  "age":30,  
  "cars": {  
    "car1":"Ford",  
    "car2":"BMW",  
    "car3":"Fiat"  
  }  
 }

You can access nested JSON objects by using the dot notation or bracket notation:

### Example

x = myObj.cars.car2;  
// or:  
x = myObj.cars["car2"];

## **Modify Values**

You can use the dot notation to modify any value in a JSON object:

### Example

myObj.cars.car2 = "Mercedes";

You can also use the bracket notation to modify a value in a JSON object:

### Example

myObj.cars["car2"] = "Mercedes";

## **Delete Object Properties**

Use the delete keyword to delete properties from a JSON object:

### Example

delete myObj.cars.car2;

## **Arrays as JSON Objects**

### Example

[ "Ford", "BMW", "Fiat" ]

Arrays in JSON are almost the same as arrays in JavaScript.

In JSON, array values must be of type string, number, object, array, boolean or null.

In JavaScript, array values can be all of the above, plus any other valid JavaScript expression, including functions, dates, and undefined.

## **Arrays in JSON Objects**

Arrays can be values of an object property:

### Example

{  
"name":"John",  
"age":30,  
"cars":[ "Ford", "BMW", "Fiat" ]  
}

## **Accessing Array Values**

You access the array values by using the index number:

### Example

x = myObj.cars[0];