What is JSON

JSON is an acronym for JavaScript Object Notation, is an open standard format, which is lightweight and text-based, designed explicitly for human-readable data interchange. It is a language-independent data format. It supports almost every kind of language, framework, and library.

In the early 2000s, JSON was initially specified by Douglas Crockford. In 2013, JSON was standardized as ECMA-404, and RCF 8259 was published in 2017.

JSON is an open standard for exchanging data on the web. It supports data structures like objects and arrays. So, it is easy to write and read data from JSON.

In JSON, data is represented in key-value pairs, and curly braces hold objects, where a colon is followed after each name. The comma is used to separate key-value pairs. Square brackets are used to hold arrays, where each value is comma-separated.

What is JSON

- JSON stands for JavaScript Object Notation.
- JSON is an open standard data-interchange format.
- JSON is lightweight and self-describing.
- JSON originated from JavaScript.
- JSON is easy to read and write.
- JSON is language independent.
- JSON supports data structures such as arrays and objects.
- JSON is plain text written in JavaScript object notation
- JSON is used to send data between computers

The JSON syntax is derived from JavaScript object notation, but the JSON format is text only.

Code for reading and generating JSON exists in many programming languages.

The JSON format was originally specified by Douglas Crockford.

Why Use JSON?

The JSON format is syntactically similar to the code for creating JavaScript objects. Because of this, a JavaScript program can easily convert JSON data into JavaScript objects.

Since the format is text only, JSON data can easily be sent between computers, and used by any programming language.

JavaScript has a built in function for converting JSON strings into JavaScript objects:

JSON.parse()

JavaScript also has a built in function for converting an object into a JSON string:

JSON.stringify()

You can receive pure text from a server and use it as a JavaScript object.

You can send a JavaScript object to a server in pure text format.

You can work with data as JavaScript objects, with no complicated parsing and translations.

Storing Data

When storing data, the data has to be a certain format, and regardless of where you choose to store it, *text* is always one of the legal formats.

JSON makes it possible to store JavaScript objects as text.

Features of JSON

- Simplicity
- Openness
- Self-Describing
- Internationalization
- Extensibility
- Interoperability

Why do we use JSON?

Since JSON is an easy-to-use, lightweight language data interchange format in comparison to other available options, it can be used for API integration. Following are the advantages of JSON:

- **Less Verbose**: In contrast to XML, JSON follows a compact style to improve its users' readability. While working with a complex system, JSON tends to make substantial enhancements.
- **Faster**: The JSON parsing process is faster than that of the XML because the DOM manipulation library in XML requires extra memory for handling large XML files. However, JSON requires less data that ultimately results in reducing the cost and increasing the parsing speed.

- **Readable**: The JSON structure is easily readable and straightforward. Regardless of the programming language that you are using, you can easily map the domain objects.
- **Structured Data**: In JSON, a map data structure is used, whereas XML follows a tree structure. The key-value pairs limit the task but facilitate the predictive and easily understandable model.

JSON vs XML

Before knowing about the differences between JSON and XML, we should be aware of the definition of json and xml.

What is json?

JSON stands for **JavaScript object notation**. JSON has been derived from javascript, where javascript is a programming language. It was originally created to hold the structured data that could be used in javascript. JSON became so popular that it is used for data for all kinds of applications. It is the most popular way of sending the data for Web APIs.

Basic data types supported by json are:

- **Strings**: Characters that are enclosed in single or double quotation marks.
- **Number**: A number could be integer or decimal, positive or negative.
- **Booleans**: The Boolean value could be either true or false without any quotation marks.
- **Null**: Here, null means nothing without any quotation marks.

In addition to basic data types, json has arrays and objects.

Arrays

Arrays are the lists that are represented by the square brackets, and the values have commas in between them. They can contain mix data types, i.e., a single array can have strings, Boolean, numbers.

For example:

Example 1: [1, 2, 7.8, 5, 9, 10];

Example 2: ["red", "yellow", "green"];

Example 3: [8, "hello", null, true];

In the above, example 1 is an array of numbers, example 2 is an array of strings, and example 3 is an array of mix data types.

Objects

Objects are JSON dictionaries that are enclosed in curly brackets. In objects, keys and values are separated by a colon ':', pairs are separated by comma. Keys and values can be of any type, but the most common type for the keys is a string.

For example: {"red" : 1, "yellow" : 2, "green" : 3};

Nesting

Nesting involves keeping the arrays and objects inside of each other. We can put the arrays inside objects, objects inside arrays, arrays inside arrays, etc. We can say that json file is a big object with lots of objects and arrays inside.

Example:

```
{
 "song" :
        {
           "title" :  "Hey Dude";
            "artist": "The Beatles";
             "musicians": ["John Lennon", "Paul McCratney", "Ringo Starr"];
         }
}
```

In the above code, the song starts with a curly bracket. Therefore, a song is an object. It contains three key-value pairs wherein title, artist and musicians are the keys.'

What is XML?

XML stands for an extensible markup language. It is like HTML, where HTML stands for Hypertext Markup language. HTML is used for creating websites, whereas XML can be used for any kind of structured data.

XML has two ways of handling data, i.e., Tags and Attributes. The tags work as HTML. The start tags start with the <_> and end with the </_>. The start and end tags must match. The names must only be letters, numbers, and underscore, and the tag name must start with a letter only.

For example:

<title> Hello World </title>

Nested Tags

When we put the tag inside of another tag that creates the nested data.

For example:

```
<color>
 <red> 1 </red>
<yellow> 2 </yellow>
<green> 3 </green>
</color>
```

As we can observe in the above code that inside the color tag, we have three more tags, i.e., red, yellow, and green.

Similarities between the json and XML.

- **Self-describing:** Both json and xml are self-describing as both xml data and json data are human-readable text.
- **Hierarchical:** Both json and xml support hierarchical structure. Here hierarchical means that the values within values.
- **Data interchange format:** JSON and XML can be used as data interchange formats by many different programming languages.
- **Parse:** Both the formats can be easily parsed.
- **Retrieve:** Both formats can be retrieved by using HTTP requests. The methods used for retrieving the data are GET, PUT, POST.

Differences between the json and XML.

JSON	XML
JSON stands for javascript object notation.	XML stands for an extensible markup language.
The extension of json file is .json.	The extension of xml file is .xml.
The internet media type is application/json.	The internet media type is application/xml or text/xml.
The type of format in JSON is data interchange.	The type of format in XML is a markup language.
It is extended from javascript.	It is extended from SGML.
It is open source means that we do not have to pay anything to use JSON.	It is also open source.
The object created in JSON has some type.	XML data does not have any type.

The data types supported by JSON are strings, numbers, Booleans, null, array.	XML data is in a string format.
It does not have any capacity to display the data.	XML is a markup language, so it has the capacity to display the content.
JSON has no tags.	XML data is represented in tags, i.e., start tag and end tag.
	XML file is larger. If we want to represent the data in XML then it would create a larger file as compared to JSON.
JSON is quicker to read and write.	XML file takes time to read and write because the learning curve is higher.
JSON can use arrays to represent the data.	XML does not contain the concept of arrays.
It can be parsed by a standard javascript function. It has to be parsed before use.	XML data which is used to interchange the data, must be parsed with respective to their programming language to use that.
It can be easily parsed and little bit code is required to parse the data.	It is difficult to parse.
File size is smaller as compared to XML.	File size is larger.
JSON is data-oriented.	XML is document-oriented.
It is less secure than XML.	It is more secure than JSON.

EXAMPLE:

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 Syntax(**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 Data - A Name and a Value

JSON data is written as name/value pairs (aka key/value pairs).

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

Example

"name":"John"

JSON names require double quotes.

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
- 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

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 Files

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

JSON Data Types

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

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

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

- a function
- a date
- *undefined*

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"}
}
```

Objects as values in JSON must follow the JSON syntax.

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}
```

Data Type	Description	Example
String	A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters.	"student", "name", "1234", "Ver_1"
Number	Number represents the numeric characters.	121, 899
Boolean	It can be either True or False.	true
Null	It is an empty value.	

JSON Object

This is a JSON string:

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

Inside the JSON string there is a JSON object literal:

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

JSON object literals are surrounded by curly braces {}.

JSON object literals contains key/value pairs.

Keys and values are separated by a colon.

Keys must be strings, and values must be a valid JSON data type:

- string
- number
- object
- array
- boolean
- null

Each key/value pair is separated by a comma.

It is a common mistake to call a JSON object literal "a JSON object".

JSON cannot be an object. JSON is a string format.

The data is only JSON when it is in a string format. When it is converted to a JavaScript variable, it becomes a JavaScript object.

Let's see an example of JSON object.

```
{
  "employee": {
    "name":     "sonoo",
    "salary":    56000,
    "married":   true
  }
}
```

In the above example, employee is an object in which "name", "salary" and "married" are the key. In this example, there are string, number and boolean value for the keys.

JSON Object with Strings

The string value must be enclosed within double quote.

```
{
    "name":     "sonoo",
    "email":    "sonoojaiswal1987@gmail.com"
}
```

JSON Object with Numbers

JSON supports numbers in double precision floating-point format. The number can be digits (0-9), fractions (.33, .532 etc) and exponents (e, e+, e-,E, E+, E-).

```
{
"integer": 34,
"fraction": .2145,
"exponent": 6.61789e+0
}
```

JSON Object with Booleans

JSON also supports boolean values *true* or *false*.

```
{
"first": true,
"second": false
}
```

JSON Nested Object Example

A JSON object can have another object also. Let's see a simple example of JSON object having another object.

```
{
   "firstName": "Sonoo",
   "lastName": "Jaiswal",
   "age": 27,
   "address" : {
     "streetAddress": "Plot-6, Mohan Nagar",
```

```
        "city": "Ghaziabad",
        "state": "UP",
        "postalCode": "201007"
      }
    }
```

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

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

You can access a JavaScript object like this:

Example

```
// returns John
person.name;
```

```
<!DOCTYPE html>
<html>
<body>

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj.name;
</script>

</body>
</html>
```

It can also be accessed like this:

Example

```
// returns John
person["name"];
```

```
<!DOCTYPE html>
<html>
```

```
<body>

<h2>Access a JavaScript object</h2>
<p id="demo"></p>

<script>
const myObj = {name:"John", age:30, city:"New York"};
document.getElementById("demo").innerHTML = myObj["name"];
</script>

</body>
</html>
```

Data can be modified like this:

Example

```
person.name = "Gilbert";
```

It can also be modified like this:

Example

```
person["name"] = "Gilbert";
```

JSON Array

This is a JSON string:

```
'["Ford", "BMW", "Fiat"]'
```

Inside the JSON string there is a JSON array literal:

```
["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.*

JSON Array of Strings

Let's see an example of JSON arrays storing string values

```
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]
```

JSON Array of Numbers

Let's see an example of JSON arrays storing number values.

```
[12, 34, 56, 43, 95]
```

JSON Array of Booleans

Let's see an example of JSON arrays storing boolean values.

```
[true, true, false, false, true]
```

JSON Array of Objects

Let's see a simple JSON array example having 4 objects.

```
{"employees":[
   {"name":"Ram", "email":"ram@gmail.com", "age":23},
   {"name":"Shyam", "email":"shyam23@gmail.com", "age":28},
   {"name":"John", "email":"john@gmail.com", "age":33},
   {"name":"Bob", "email":"bob32@gmail.com", "age":41}
]}
```

JSON Array of Objects is an ordered list of objects. Following is the example of JSON array having three Objects. In the example, the object "Book" is an array containing three objects Each object is a record of a book (with a title and price).

```
{"Book":
 [
 {"title":"Advanced Web Technologies", "price":200},
 {"title":"Fundamental of web", "price":140},
{"title":"Advanced concept", "price":300},
]};
```

The first title in the object array can be accessed like this: Book[0].title (remember: The array index starts with 0)

Book [0].title

OR

Book [0]["title"]

How to Assign / modify value:

To assign value use syntax:
Array_name[index].attribute=value;

Example:
Book[0].title=”Advanced Web Technologies”;
Book[0][“title”]=” Advanced Web Technologies”;

JSON Multidimensional Array

We can store array inside JSON array, it is known as array of arrays or multidimensional array.

```
[
 [ "a", "b", "c" ],
 [ "m", "n", "o" ],
 [ "x", "y", "z" ]
]
```

It is possible to store an array inside another JSON array. This concept is known as an array of arrays or a multi-dimensional array.

Following example Create multidimensional array object

```
var books = { ".net" : [

{"title":".net complete", "price": 200},
{"title":".net black book", "price" : 300 }
],
"web" : [

{"title":"concepts of web technology ", "price": 150),
 {"title":"Advanced Web Technologies", "price": 130}]}
```

JSON Comments

JSON doesn't support comments. It is not a standard.

But you can do some tricks such as adding extra attribute for comment in JSON object, for example:

```
{
   "employee": {
      "name":     "Bob",
      "salary":     56000,
      "comments":   "He is a nice man"
   }
}
```

Here, "comments" attribute can be treated as comment.

JSON doesn't support comments by default. However, there are some tricks we can use, such as adding an extra attribute or key that work as comments in a JSON object. Consider following example. Here, key-value pair ("comments": "This is comment" serve as our comment.

```
{"book": {
"title": "Advanced Web Technologies",
"price": 200,
"comments": "This is comment"
}}
```

Consider following example. Here, attribute "comment_1" serve as comment.

```
{"book":{

"title": "Advanced Web Technologies",

"comment_1": "This is good book"
}}
```

JSON with javascript Example

Following example demonstrate creation of JSON object "book" using javascript.

```
<html>

<head>
```

```
<h3>Demo Example: JSON with JavaScript</h3>

<script>

var book = {"title":"Advanced Web Technologies", "author":"imp"};

document.write("<br>");

document.write("Title = " + book.title);

document.write("<br>");

document.write("Author = " + book.author);

</script>

</head>

<body>

</body>

</html>
```

Output:

Demo Example: JSON with JavaScript

Title Advanced Web Technologies
Author=jmp

JSON array with javascript Example:

Following example demonstrate creation of JSON object "book" and array "author

using javascript.

```
<html>

<head>

<h3>Demo Example: JSON array with JavaScript</h3>
```

```
<script>

var book = {"title":"Advanced Web Technologies",

"authors" : ["imp","mv"] };

document.write("<br>");

document.write("Title = " + book.title);

document.write("<br>");

document.write("Author name -1: " + book.authors[0]);

document.write("<br>");

document.write("Author name -2: " + book.authors[1]);

</script>

</head>

</body>

</html>
```

Output

Demo Example: JSON array with JavaScript

Title= Advanced Web Technologies

Author name -1: jmp
Author name -2: mv

EXAMPLES

JSON OBJECT

JavaScript Objects

You can create a JavaScript object from a JSON object literal:

Example

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

Normally, you create a JavaScript object by parsing a JSON string:

Example

```
myJSON = '{"name":"John", "age":30, "car":null}';
myObj = JSON.parse(myJSON);
```

Accessing Object Values

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

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj.name;
```

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

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj["name"];
```

Looping an Object

You can loop through object properties with a for-in loop:

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += x + ", ";
}
```

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

Example

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += myObj[x] + ", ";
}
```

JSON ARRAY

JavaScript Arrays

You can create a JavaScript array from a literal:

Example

```
myArray = ["Ford", "BMW", "Fiat"];
```

You can create a JavaScript array by parsing a JSON string:

Example

```
myJSON = '["Ford", "BMW", "Fiat"]';
myArray = JSON.Parse(myJSON);
```

Accessing Array Values

You access array values by index:

Example

```
myArray[0];
```

Arrays in Objects

Objects can contain arrays:

Example

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

You access array values by index:

Example

```
myObj.cars[0];
```

Looping Through an Array

You can access array values by using a for in loop:

Example

```
for (let i in myObj.cars) {
  x += myObj.cars[i];
}
```

Or you can use a for loop:

Example

```
for (let i = 0; i < myObj.cars.length; i++) {
  x += myObj.cars[i];
}
```

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.