JSON Notes

Contents

[1 JSON Introducion 2](#_Toc153289769)

[2 JSON Syntax 3](#_Toc153289770)

[3 JavaScript Objects 4](#_Toc153289771)

[3.1 *for…in* Loop 4](#_Toc153289772)

[3.2 Convert Object into an Array 4](#_Toc153289773)

[3.3 Iterate over Array with *for…of* Loop 4](#_Toc153289774)

[3.4 Iterate over Array with IndexedLoop 5](#_Toc153289775)

[3.5 Quoted Property Names 5](#_Toc153289776)

[4 JS Support for JSON 5](#_Toc153289777)

[5 JSON Example 6](#_Toc153289778)

[5.1 Retrieving the Data 7](#_Toc153289779)

[5.2 Page Structure 8](#_Toc153289780)

[5.3 *populateHeaders* Function 8](#_Toc153289781)

[5.4 *showHeroes* Function 9](#_Toc153289782)

[6 Expectations 9](#_Toc153289783)

[Appendix 1 W3Schools Tutorials 10](#_Toc153289784)

[Appendix 2 Additional Information about JavaScript Objects 11](#_Toc153289785)

[Appendix 3 JavaScript Objects – “Classes” 12](#_Toc153289786)

[Appendix 4 ECMAScript vs. JavaScript 12](#_Toc153289787)

# JSON Introducion

JSON (JavaScript Object Notation) – Similar to XML, is an open format, text-based data exchange format that is easy for humans and machines to read. The examples below show the same data represented in JSON and XML.

|  |
| --- |
| **JSON** |
| 1. {"employees":[ 2. {"name":"Shyam", "email":"shyamjaiswal@gmail.com"}, 3. {"name":"Bob", "email":"bob32@gmail.com"}, 4. {"name":"Jai", "email":"jai87@gmail.com"} 5. ]} |
| 1. **XML** |
| <employees>  <employee>  <name>Shyam</name>  <email>shyamjaiswal@gmail.com</email>  </employee>  <employee>  <name>Bob</name>  <email>bob32@gmail.com</email>  </employee>  <employee>  <name>Jai</name>  <email>jai87@gmail.com</email>  </employee>  </employees> |

JSON vs XML: <https://www.w3schools.com/js/js_json_xml.asp>

JSON is

* A *lightweight* data-interchange format
* Text, so it can be sent to and from a server
* Self-describing and easy to understand
* Used by many web services and API’s for providing data to clients. For example, [AWS](https://aws.amazon.com/documentdb/what-is-json/) provides support for jSON document databases, that are nonrelational databases designed to store and query data.

Many API’s use JSON: [Wikipedia](https://codesnippet.io/wikipedia-api-tutorial/), [Facebook](https://developers.facebook.com/docs/graph-api/overview/), [Google Maps](https://developers.google.com/maps/documentation/javascript/importing_data), [Home Depot](https://serpapi.com/home-depot-product), *etc.*

# JSON Syntax

JSON syntax is derived from JavaScript object notation syntax:

* Data is in key/value pairs
* The *key* must be double quoted
* Data is separated by commas
* Curly braces hold objects
* Square brackets hold arrays

JSON Object Example:

{"name": "Lorenzo", "salary": 76000, "married": **true** }

JSON Array:

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

JSON Array with Objects Example:

[ {"name":"Ram", "email":"Ram@gmail.com"},

{"name":"Bob", "email":"bob32@gmail.com"} ]

JSON Object with Array Property Example:

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

JSON Object Example:

{"employee": {"name": "Lorenzo", "salary": 76000, "married": **true** } }

Syntax: <https://www.w3schools.com/js/js_json_syntax.asp> (stop at JavaScript Objects)

Data Types: <https://www.w3schools.com/js/js_json_datatypes.asp>

Arrays: <https://www.w3schools.com/js/js_json_arrays.asp>

# JavaScript Objects

JavaScript does not have classes, it only has objects. However, you can encapsulate data and methods into objects. A JavaScript object is a list of *name:value* pairs. An example of a JS *literal* is:

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

Note that a JS object is very similar to a JSON string. One difference is the name of the data fields (*e.g. firstName, lastName, age, eyeColor*)do not have to be quoted as they do with JSON. As we see later, they can be quoted. A JS object can also be created as shown below. However, the preceding technique is preferred because it is simpler, more readable, and faster:

var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

JS objects are addressed by reference, just the same as Java. Thus, the statement below defines another reference to the same object:

var x = person;

You can use several syntaxes for accessing the property of an object. For example:

1. person.age
2. person[“age”]
3. objectName[expression] – where the expression evaluates to a property name. For example:

x=“age”;

person[x]

## *for…in* Loop

JS provides a *for…in* loop for iterating over the properties of an object. Note in the code below, that the loop is actually looping over the property names, which are then used to access the property values. For example:

const person = { name: "John", age: 50, city: "New York" };

let msg = "";

for (x in person) {

msg += x + "=" + person[x] + ", ";

}

alert(msg); // name=John, age=50, city=New York

## Convert Object into an Array

The values in a JS object can be converted to an array using: *Object.values()*. For example:

const person = { name: "John", age: 50, city: "New York" };

const myArray = Object.values(person);

You can display an array: alert(myArray); // John,50,New York

## Iterate over Array with *for…of* Loop

JS provides a *for…of* loop for iterating over the values in an array.

for (val of myArray) {

msg += val + ", ";

}

alert(msg); // John, 50, New York,

## Iterate over Array with IndexedLoop

You can also iterate over an array with an indexed loop:

for (i = 0; i < myArray.length; i++) {

msg += myArray[i] + ", ";

}

alert(msg); // John, 50, New York,

## Quoted Property Names

JS also allows you to use quoted property names when defining an object. For example:

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

Using this notation, the only thing you can’t do (from the examples above) is use the “dot” notation to access properties. They must be quoted. For example: person["first name"].

Note that this style of object is very similar to JSON. The difference is that JSON is a string, not an object.

# JS Support for JSON

JavaScript has a method, *JSON.parse*, to convert a JSON string into a JavaScript object. For example:

const jSONString = '{ "name": "John", "age": 31, "city": "New York" }';

const jsObjectPerson = JSON.parse(jSONString);

myDiv.innerHTML = jsObjectPerson.name;

JavaScript has a method, *JSON.stringify*, to convert a JavaScript object into a JSON string. For example:

const jsObjectPerson = { name: "John", age: 50, city: "New York" };

const jSONString = JSON.stringify(jsObjectPerson);

myDiv.innerHTML = jSONString;

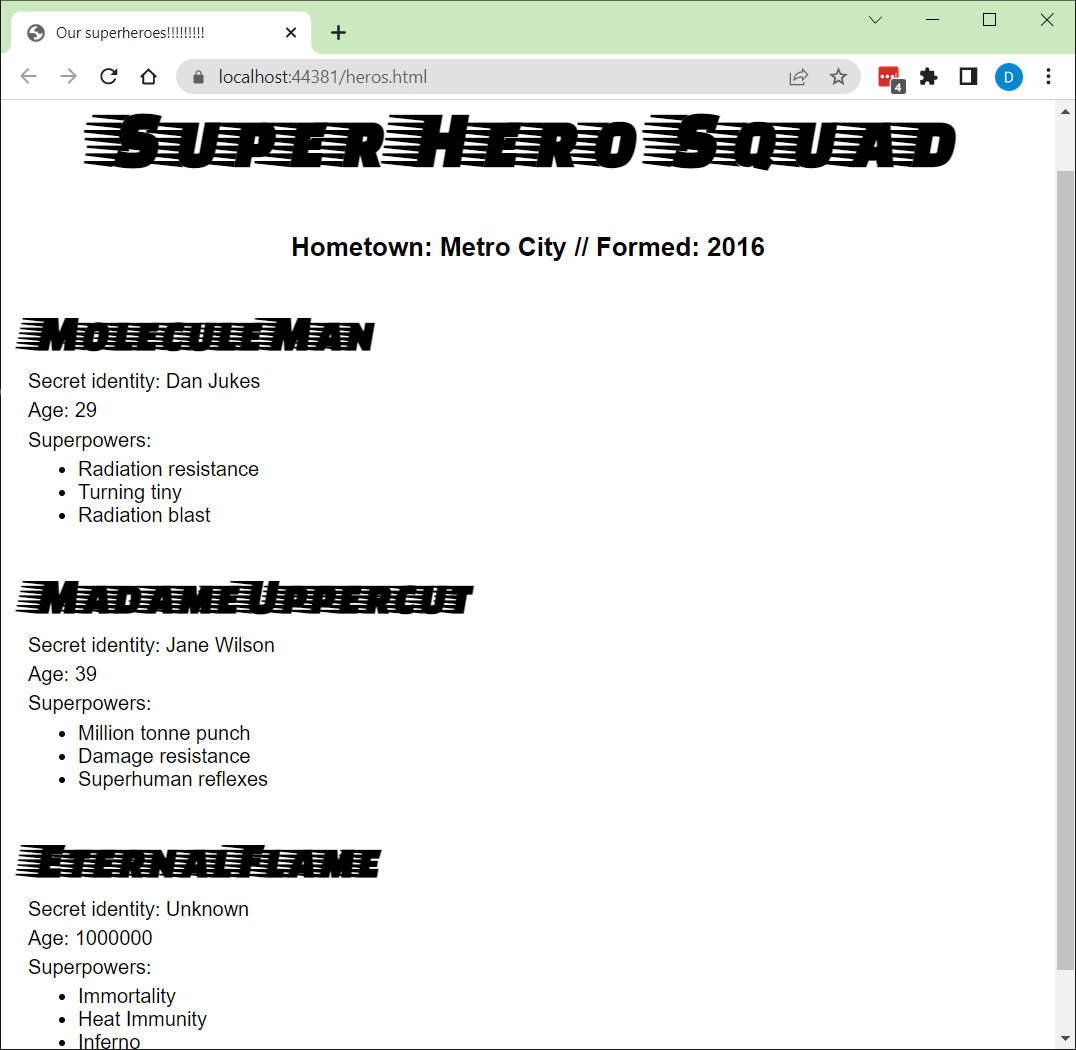
alert(jSONString); // {"name":"John","age":50,"city":"New York"}

# JSON Example

Consider this [jSON data](https://github.com/drgap/json_example/blob/main/superheroes.json) that represents a squad of super heroes. The data is shown below on the left and on the right is a class diagram that summarizes the data.

|  |  |
| --- | --- |
|  |  |

We will write a web page to read and display this data as shown below:



## Retrieving the Data

The code for this example is in the download (*heroes.html*). In what follows, we step through the code. First, the code uses AJAX to make an asynchronous call to retrieve the data from my GitHub account. When the data is received (*request.response*) it is converted into a JS object (*superHeroes*), which is passed to two functions (*populateHeader* & *showHeroes*) to build the page and display the data.

let requestURL = 'https://raw.githubusercontent.com/drgap/json\_example/main/superheroes.json';

// AJAX - make asynchronous call to get data.

let request = new XMLHttpRequest();

request.open('GET', requestURL);

request.responseType = 'json'; // type of data being returned

request.send();

// When server responds, call this function

request.onload = function () {

const superHeroes = request.response; // Converts the jSON to a single object.

populateHeader(superHeroes); // Build page header

showHeroes(superHeroes); // Build main section of page

}

Note in the code above, by setting:

request.responseType = 'json';

when the response is received:

const superHeroes = request.response;

the data has been automatically converted to a JS object. However, if we had set:

request.responseType = 'text';

when the response is received we would have need to parse it to create a JS object:

const superHeroes = JSON.parse(request.response);

## Page Structure

The HTML is shown below. Note the following:

* We use the *semantic* tags: *header* and *section –* these tags are [layout elements](https://www.w3schools.com/html/html_layout.asp). We study this in more detail later when we consider *responsive design.*
* The main script is in the body of the page because we need it to execute and build the page. The helper functions, *populateHeader* & *showHeroes* are in the *head* section (not shown here).
* The *header* and *section* variables are global variables because they are declared outside any functions. Thus, they will be available in the helper functions.
* The *querySelector* method returns the first occurrence of the argument. Alternately, *header* could have been accessed with: const header = document.getElementById('hdr');

<body>

<header id="hdr">

</header>

<section>

</section>

<script>

const header = document.querySelector('header');

const section = document.querySelector('section');

let requestURL = 'https://raw.githubusercontent.com/...superheroes.json';

let request = new XMLHttpRequest();

request.open('GET', requestURL);

...

populateHeader(superHeroes); // Build page header

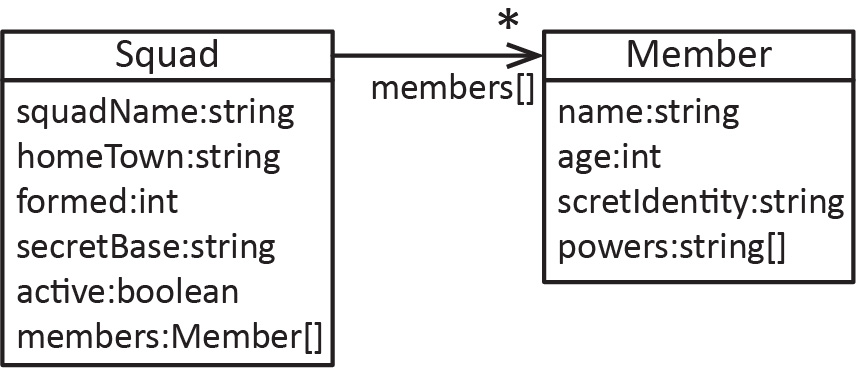
showHeroes(superHeroes); // Build main section of page

}

</script>

</body>

## *populateHeaders* Function

The code is below. Note that *header* is the global variable representing the header section in the page.

function populateHeader(superHeroes) {

const myH1 = document.createElement('h1');

myH1.textContent = superHeroes['squadName'];

header.appendChild(myH1);

const myPara = document.createElement('p');

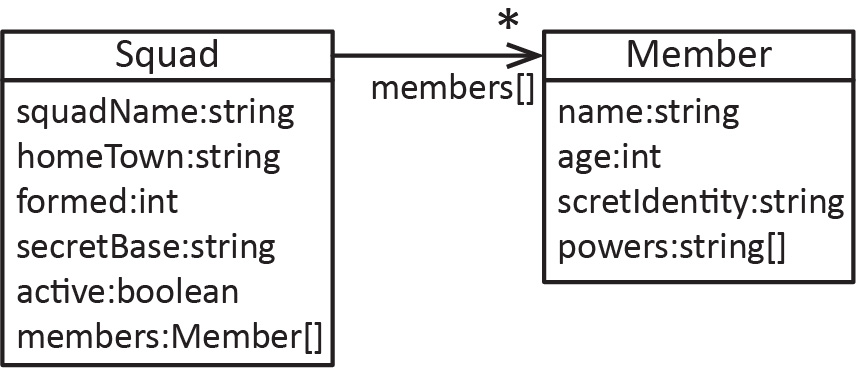
myPara.textContent = 'Hometown: ' + superHeroes['homeTown'] +

' // Formed: ' + superHeroes['formed'];

header.appendChild(myPara);

}

## *showHeroes* Function

 The code is below. Note that *section* is the global variable representing the section tag in the page.

function showHeroes(superHeroes) {

// members is an array of objects

const heroes = superHeroes['members'];

for (let i = 0; i < heroes.length; i++) {

const myArticle = document.createElement('article');

const myH2 = document.createElement('h2');

const myPara1 = document.createElement('p');

const myPara2 = document.createElement('p');

const myPara3 = document.createElement('p');

const myList = document.createElement('ul');

myH2.textContent = heroes[i].name;

myPara1.textContent = 'Secret identity: ' + heroes[i].secretIdentity;

myPara2.textContent = 'Age: ' + heroes[i].age;

myPara3.textContent = 'Superpowers:';

const superPowers = heroes[i].powers; // heros[i].powers is an array of objects

for (let j = 0; j < superPowers.length; j++) {

const listItem = document.createElement('li');

listItem.textContent = superPowers[j];

myList.appendChild(listItem);

}

myArticle.appendChild(myH2);

myArticle.appendChild(myPara1);

myArticle.appendChild(myPara2);

myArticle.appendChild(myPara3);

myArticle.appendChild(myList);

section.appendChild(myArticle);

}

}

# Expectations

1. Given the description of a situation, write a JSON string.

Appendix

1. W3Schools Tutorials

This material is heavily lifted from:

* <https://www.w3schools.com/js/js_json_intro.asp>
* <https://www.w3schools.com/js/js_object_definition.asp>

I have listed the chapters and highlighted in blue the most useful pages.

|  |
| --- |
| [JS Objects](https://www.w3schools.com/js/js_object_definition.asp)   1. Object Definitions 2. Object Properties 3. Object Methods 4. Object Accessors 5. Object Constructors 6. Object Prototypes 7. Object ECMAScript5   [JS JSON](https://www.w3schools.com/js/js_json_intro.asp)   1. JSON Intro 2. JSON Syntax 3. JSON vs XML 4. JSON Data Types 5. JSON Parse 6. JSON Stringify 7. JSON Objects 8. JSON Arrays 9. JSON PHP 10. JSON HTML 11. JSON JSONP |

1. Additional Information about JavaScript Objects
2. JS properties can be added and deleted. For example:

var person = {fname:"John", lname:"Doe", age:25};

person.nationality = "English"; // Add property

document.getElementById("demo").innerHTML =

person.fname + " is " + person.nationality + ".";

delete person.age;

1. Example of a JS method:

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

// Display data from the object:

document.getElementById("demo").innerHTML = person.fullName();

1. JS methods are functions stored as object properties. Thus, if you loop over the object properties, the *fullName* property in the example below will display the method itself

var person = {

firstName: "John",

lastName : "Doe",

id : 5566,

fullName : function() {

return this.firstName + " " + this.lastName;

}

};

for (x in person) {

txt += person[x] + " ";

};

Output:

John Doe 5566 function() { return this.firstName + " " + this.lastName; }

1. JavaScript Objects – “Classes”
2. A JS “class”:

// Constructor

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

// Add a method:

Person.prototype.name = function() {

return this.firstName + " " + this.lastName

};

// Create an object

var myFather = new Person("John", "Doe", 50, "blue");

var myMother = new Person("Mary", "Doe", 51, "brown");

document.getElementById("demo").innerHTML = "My father is " + myFather.name();

1. ECMAScript vs. JavaScript
2. ECMAScript vs. JavaScript

The ECMAScript specification is a standardized specification of a scripting language. ECMAScript is a programming language. JavaScript, on the other hand, can be viewed as a "dialect" of ECMAScript. It conforms to the ECMAScript specification while also providing additional features not described in the specification. However, JS does not provide all the features of ECMAScript.

<https://en.wikipedia.org/wiki/ECMAScript>