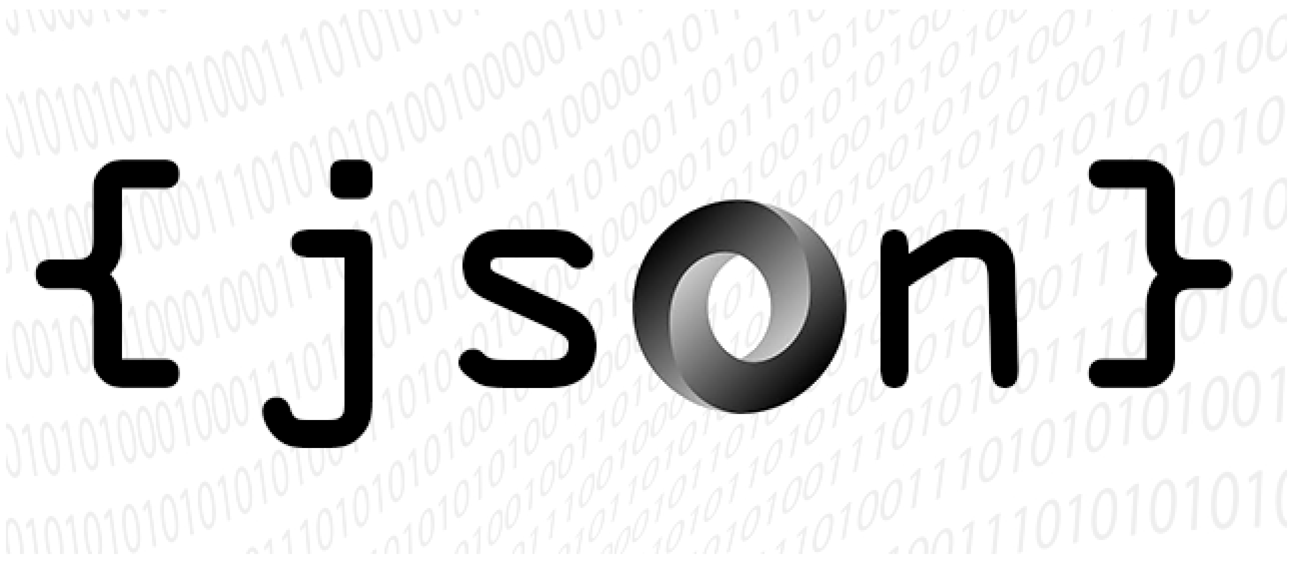
Understanding JSON in JavaScript

Timothy Robards
Jan 10 · 7 min read ★



JSON (**J**ava**S**cript **O**bject **N**otation) is a lightweight format for sharing data. Although it's derived from JavaScript — it may be used with many programming languages. In this article however, we'll be focusing on the use of JSON in JavaScript.

Where will we use JSON?

Some scenarios include:

- Data storage
- Data configuration and verification
- Generating data structures from user input
- Data transfer from client to server, server to client, and between servers

A note on AJAX — JSON is often used with AJAX, or **A**synchronous **J**avaScript **a**nd **X**ML. AJAX is used to transfer data without having to refresh the browser page. Later in this article, we'll look at how to make an AJAX request to a JSON file, and output the content in the browser.

Syntax & Structure

Most of the time you'll encounter JSON saved in a `.json` file. JSON files consist of a series of key/value pairs.

```
{ "key": "value" }
```

Note that both the key and value are wrapped in double-quotes. If this was a JavaScript object literal — we wouldn't need the quotes around the key, just the string value. To parse this as valid JSON, we need to remember to keep our keys in quotes and also follow the specific data types which follow..

Data Types

The following data types can be used with JSON:

- strings (in double quotes)
- numbers
- objects
- arrays
- Booleans (true or false)
- null

Each of the data types that are passed into JSON as values will maintain their own syntax, strings will be in double-quotes, but numbers will not be.

Use cases

Lets take a look at an example JSON file, `contacts.json` :

```
{
  "name":"Timothy",
  "age":35,
  "address":{
        "street":"1 Main St",
        "city": "Montreal"
  },
  "interests":["cooking", "biking"]
}
```

Here we have a `name` set as a string, `age` as a number, the `address` is an embedded object and the `interests` are an array of strings.

When you're working with JSON, you might also encounter JSON as an object or a string within the context of a program. If your JSON object is in a `.js` or `.html` file — it'll likely be set to a variable. Then our example would look like:

```
let contacts = {
  name:"Timothy",
  age:35,
  address:{
        street:"1 Main St",
        city: "Montreal"
  },
  interests:["cooking", "biking"]
}
```

Or if stored as a string, it'll be a one-liner:

```
let contacts = { "name":"Timothy", "age":35, "address":{ "street":"1
Main St", "city": "Montreal"}, "interests":["cooking", "biking"]}
```

Of course writing in the JSON format on multiple lines makes it much more readable, especially when dealing with a lot of data. JSON ignores whitespace between its elements, so you're free to space out your key-value pairs to make the data easier on the eyes!

Working with JSON in JavaScript

Let's run through an our own example! Open up your text editor & create an HTML file as follows:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JSON Demo</title>
</head>
<body>
  <script>
```

```
    let contacts = {
      name: "Timothy",
      age: 35,
      address: {
        street: "1 Main St",
        city: "Montreal"
      },
      interests: ["cooking", "biking"]
    }
  </script>
</body>
</html>
```

We can access our JSON data through the console using dot notation, like so:

```
contacts.name             // Timothy
contacts.age              // 35
contacts.address.street   // 1 Main St
contacts.address.city     // Montreal
contacts.interests[0]     // cooking
contacts.interests[1]     // biking
```

When we want to transfer or store our data, its very useful to be able to convert our JSON object to a string using `JSON.stringify()` . And likewise convert from a string back into an object using `JSON.parse()` .

JSON.stringify()

The `JSON.stringify()` function converts a JavaScript object into a JSON string. Strings are lightweight and therefore very useful when transporting data from a client to a server.

Let's return our previous example, and assign the `JSON.stringify()` method to variable `s` . We pass our `contacts` object into the function.

```
let contacts = {
  name: "Timothy",
  age: 35,
  address: {
    street: "1 Main St",
    city: "Montreal"
  },
  interests: ["cooking", "biking"]
}

let s = JSON.stringify(contacts);
```

If we now take a look at `s` in the console, we'll see the JSON available to us as a string rather than an object.

```
'{"name":"Timothy","age":35,"address":{"street":"1 Main
St","city":"Montreal"},"interests":["cooking","biking"]}'
```

The `JSON.stringify()` function lets us convert objects to strings. To do the opposite, we use the `JSON.parse()` function.

JSON.parse()

In order to convert a string back into a function we use the built in `JSON.parse()` function, to decode the string.

```
let contacts = JSON.parse(s);
```

We can now access the data like a regular JavaScript object.

```
contacts.name  // Timothy
```

Looping through arrays

Its often necessary to loop through arrays of objects when working with JSON. Lets create a quick array:

```
let users = [
  {
    name: "Timothy",
    age: 35
  },
  {
    name: "Georgette",
    age: 29
  },
  {
    name: "Craig",
    age: 54
    }
  ];
```

And lets create a `for` loop, to iterate through our data:

```
for (var i = 0; i < users.length; i++) {
        console.log(users[i].name);
}

------------
output:

Timothy
Georgette
Craig
```

Lets now revise our HTML file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JSON Demo</title>
</head>
<body>
  <ul id="users"></ul>
  <script>
    let contacts = {
      name: "Timothy",
      age: 35,
      address: {
        street: "1 Main St",
        city: "Montreal"
      },
      interests: ["cooking", "biking"]
    }

    // let s = JSON.stringify(contacts);
    // let s = JSON.parse(contacts);

    let users = [
      {
        name: "Timothy",
        age: 35
      },
      {
        name: "Georgette",
        age: 29
      },
      {
        name: "Craig",
        age: 54
      }
    ];

    let output = '';

    for (var i = 0; i < users.length; i++) {
      // console.log(users[i].name);
      output += '<li>'+users[i].name+'</li>';
    }
    document.getElementById('users').innerHTML = output;
  </script>
</body>
</html>
```

Note the changes in **bold.** I've added a `ul` element to hold our output. And an `output` variable in our script, which stores our data from the loop via `getElementById`.

Accessing JSON from a URL

Most of the time we'll need to access JSON from a URL. To do this there is an extra step involved — we will use AJAX to make a a **GET** request for the file. For our demo, let's take the above JSON string and put it into a new file named `users.json`. It should look like:

```
[
  {
    "name": "Timothy",
    "age": 35
  },
  {
    "name": "Georgette",
    "age": 29
  },
  {
    "name": "Craig",
    "age": 54
  }
]
```

Note: As this is a JSON file (not an object), you must double-quote your keys!

Now we'll make an `XMLHttpRequest()`.

```
let request = new XMLHttpRequest();
```

We’ll open the file `users.json` via GET (URL) request.

```
request.open('GET', 'users.json', true);
```

Of course if your JSON file is stored externally you would replace `‘users.json’` with the complete URL address e.g. `‘https://api.myjson.com/demo/sample.json’`

Now we parse our JSON data within the `onload` function. And then loop through the data, displaying in our `ul` element.

```
request.onload = function () {

  // Convert JSON data to an object
  let users = JSON.parse(this.response);

  let output = '';
  for (var i = 0; i < users.length; i++) {
    output += '<li>' + users[i].name + ' is ' + users[i].age + '
years old.'; '</li>'
  }
  document.getElementById('users').innerHTML = output;
}
```

Then finally, we submit the request!

```
request.send();
```

Here’s the final code.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>JSON Demo</title>
</head>
<body>
  <ul id="users"></ul>
  <script>
```

```
      let request = new XMLHttpRequest();
      request.open('GET', 'users.json', true);
        request.onload = function () {
        // Convert JSON data to an object
        let users = JSON.parse(this.response);

        let output = '';
        for (var i = 0; i < users.length; i++) {
          output += '<li>' + users[i].name + ' is ' + users[i].age + '
years old.'; '</li>'
        }
        document.getElementById('users').innerHTML = output;
      }

      request.send();
    </script>
  </body>
</html>
```

We're almost done!

But if you open up the HTML file now you'll get a console error. As we're calling `users.json` from the file-system instead of an actual domain (its a browser security protocol). So to test this out locally, we'll need to run it on a local server.

If you have xampp installed, go ahead and upload it to your webroot. However, a simpler solution would be to install liveserver via npm.

Make sure you have NodeJS installed and run `npm install -g live-server` from the directory where your HTML file is stored.

After the module installation completes you simply run `live-server` from the same directory and it'll open up the HTML file on a local server.

And our data will print as specified!

```
Timothy is 35 years old.
Georgette is 29 years old.
Craig is 54 years old.
```

Using Fetch

The Fetch API is a newer built-in feature of JavaScript that makes working with requests and responses much easier. We can use Fetch with our data like so:

```
fetch('./users.json').then(response => {
  return response.json();
}).then(data => {
  // Work with your JSON data here..
  console.log(data);
}).catch(err => {
  // What do when the request fails
  console.log('The request failed!');
});
```

Using jQuery

If you're using jQuery on your project, you can retrieve data using the `getJSON()` function.

```
$(document).ready(function () {
  $.getJSON('users.json', function (data) {
    // Work with your JSON data here..
    console.log(data[0].name);
  });
});
```

Don't forget to add jQuery above this function, if you're testing it out!

Conclusion

Thats it! I hope this article helped you increase your understanding of JSON as well as see it in action with AJAX. You're unlikely to be creating your own `.json` files but rather working with them from other sources. So hopefully the demos here have shed some light on this process, we've used native JavaScript, the Fetch API and jQuery to access data. JSON is widely adopted in within the industry and is a fundamental skill for all serious developers.

I hope you found this article useful! You can follow me on Medium. I'm also on Twitter. Feel free to leave any questions in the comments below. I'll be glad to help out!

JavaScript Json Tech Front End Development Ajax