

MDN's new design is in Beta! A sneak peek: <https://blog.mozilla.org/opendesign/mdns-new-design-beta/>

Learn web development

Working with JSON data

[← Previous](#)[↑ Overview: Objects](#)[Next →](#)

JavaScript Object Notation (JSON) is a standard format for representing structured data as JavaScript objects, which is commonly used for representing and transmitting data on web sites (i.e. sending some data from the server to the client, so it can be displayed on a web page). You'll come across it quite often, so in this article we give you all you need to work with JSON using JavaScript, including accessing data items in a JSON object and writing your own JSON.

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, familiarity with JavaScript basics (see First steps and Building blocks) and OJS basics (see Introduction to objects).
Objective:	To understand how to work with data stored in JSON, and create your own JSON objects.

No, really, what is JSON?

JSON is a data format following JavaScript object syntax, which was popularized by [Douglas Crockford](#). Even though it is based on JavaScript syntax, it can be used independently from JavaScript, and many programming environments feature the ability to read (parse) and generate JSON.

JSON can exist as an object, or a string — the former is used when you want to read data out of the JSON, and the latter is used when you want to send the JSON across the network. This is not a big issue — JavaScript provides a global [JSON](#) object that has methods available for converting between the two.

A JSON object can be stored in its own file, which is basically just a text file with an extension of `.json`, and a [MIME type](#) of `application/json`.

JSON structure

We've implied above that a JSON object is basically a JavaScript object, and this is mostly right. You can include the same basic data types inside JSON as you can in a standard JavaScript object — strings, numbers, arrays, booleans, and other object literals. This allows you to construct a data hierarchy, like so:

```
1  {
2    "squadName": "Super hero squad",
3    "homeTown": "Metro City",
4    "formed": 2016,
5    "secretBase": "Super tower",
6    "active": true,
7    "members": [
8      {
9        "name": "Molecule Man",
10       "age": 29,
11       "secretIdentity": "Dan Jukes",
12       "powers": [
13         "Radiation resistance",
14         "Turning tiny",
15         "Radiation blast"
16       ]
17     },
18     {
19       "name": "Madame Uppercut",
20       "age": 39,
21       "secretIdentity": "Jane Wilson",
22       "powers": [
23         "Million tonne punch",
24         "Damage resistance",
25         "Superhuman reflexes"
26       ]
27     },
28     {
29       "name": "Eternal Flame",
30       "age": 1000000,
31       "secretIdentity": "Unknown",
32       "powers": [
33         "Immortality",
34         "Heat Immunity",
35         "Inferno",
36         "Teleportation",
37         "Interdimensional travel"
38       ]
39     }
40   ]
41 }
```


If we loaded this object into a JavaScript program, saved in a variable called `superHeroes` for example, we could then access the data inside it using the same dot/bracket notation we looked at in the [JavaScript object basics](#) article. For example:

```
1 | superHeroes.hometown
2 | superHeroes['active']
```

To access data further down the hierarchy, you simply have to chain the required property names and array indexes together. For example, to access the third superpower of the second hero listed in the members list, you'd do this:

```
1 | superHeroes['members'][1]['powers'][2]
```

1. First we have the variable name — `superHeroes`.
2. Inside that we want to access the `members` property, so we use `["members"]`.
3. `members` contains an array populated by objects. We want to access the second object inside the array, so we use `[1]`.
4. Inside this object, we want to access the `powers` property, so we use `["powers"]`.
5. Inside the `powers` property is an array containing the selected hero's superpowers. We want the third one, so we use `[2]`.

 **Note:** We've made the JSON seen above available inside a variable in our [JSONTest.html](#) example (see the [source code](#)). Try loading this up and then accessing data inside the variable via your browser's JavaScript console.

Arrays as JSON

Above we said "We've implied above that a JSON object is basically a JavaScript object, and this is mostly right" — the reason we said "mostly right" is that an array can also be a valid JSON object, for example:

```
1 | [
2 |   {
3 |     "name": "Molecule Man",
4 |     "age": 29,
5 |     "secretIdentity": "Dan Jukes",
6 |     "powers": [
7 |       "Radiation resistance",
8 |       "Turning tiny",
9 |       "Radiation blast"
10 |    ]
11 |  },
12 |  {
13 |    "name": "Madame UpperCut",
```

```
14     "age": 39,  
15     "secretIdentity": "Jane Wilson",  
16     "powers": [  
17         "Million tonne punch",  
18         "Damage resistance",  
19         "Superhuman reflexes"  
20     ]  
21 }  
22 ]
```

The above is perfectly valid JSON. You'd just have to access array items by starting with an array index, for example `[0]["powers"][0]`.

Other notes

- JSON is purely a data format — it contains only properties, no methods.
- JSON requires double quotes to be used to be valid. It is safest to write it with double quotes, not single quotes.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You should be careful to validate any data you are attempting to use (although computer-generated JSON is less likely to include errors, as long as the generator program is working correctly). You can validate JSON using an application like [JSONLint](#).
- JSON can actually take the form of any data type that is valid for inclusion inside a standard JSON object, not just arrays or objects. So for example, a single string or number would be a valid JSON object. Not that this would be particularly useful...
- Unlike in JavaScript code in which identifiers may be used as properties, in JSON, only strings may be used as properties.

Active learning: Working through a JSON example

So, let's work through an example to show how we could make use of some JSON data on a website.

Getting started

To begin with, make local copies of our [heroes.html](#) and [style.css](#) files. The latter contains some simple CSS to style our page, while the former contains some very simple body HTML:

```
1 <header>  
2 </header>  
3  
4 <section>  
5 </section>
```

Plus a `<script>` element to contain the JavaScript code we will be writing in this exercise. At the moment it only contains two lines, which grab references to the `<header>` and `<section>` elements and store them in variables:

```
1 | var header = document.querySelector('header');  
2 | var section = document.querySelector('section');
```

We have made our JSON data available on our GitHub, at <https://mdn.github.io/learning-area/javascript/ojs/json/superheroes.json>.

We are going to load it into our page, and use some nifty DOM manipulation to display it, like this:

SUPERHERO SQUAD

Hometown: Metro City // Formed: 2016

MOLECULE MAN

Secret identity: Dan Jukes

Age: 29

Superpowers:

- Radiation resistance
- Turning tiny
- Radiation blast

MADAME UPPERCUT

Secret identity: Jane Wilson

Age: 39

Superpowers:

- Million tonne punch
- Damage resistance
- Superhuman reflexes

ETERNAL FLAME

Secret identity: Unknown

Age: 1000000

Superpowers:

- Immortality
- Heat Immunity
- Inferno
- Teleportation
- Interdimensional travel

Loading our JSON

To load our JSON into our page, we are going to use an API called `XMLHttpRequest` (often called XHR). This is a very useful JavaScript object that allows us to make network requests to retrieve resources from a server via JavaScript (e.g. images, text, JSON, even HTML snippets), meaning that we can update small sections of content without having to reload the entire page. This has led to more responsive web pages, and sounds exciting, but it is unfortunately beyond the scope of this article to teach it in much more detail.

1. To start with, we are going to store the URL of the JSON file we want to retrieve in a variable. Add the following at the bottom of your JavaScript code:

```
1 | var requestURL = 'https://mdn.github.io/learning-area/javascript/ojs/';
```

2. To create a request, we need to create a new request object instance from the `XMLHttpRequest` constructor, using the `new` keyword. Add the following below your last line:

```
1 | var request = new XMLHttpRequest();
```

3. Now we need to open a new request using the `open()` method. Add the following line:

```
1 | request.open('GET', requestURL);
```

This takes at least two parameters — there are other optional parameters available. We only need the two mandatory ones for this simple example:

- The HTTP method to use when making the network request. In this case `GET` is fine, as we are just retrieving some simple data.
- The URL to make the request to — this is the URL of the JSON file that we stored earlier.

4. Next, add the following two lines — here we are setting the `responseType` to JSON, so the server will know we want a JSON object returned, and sending the request with the `send()` method:

```
1 | request.responseType = 'json';  
2 | request.send();
```

5. The last bit of this section involves waiting for the response to return from the server, then dealing with it. Add the following code below your previous code:

```
1 | request.onload = function() {  
2 |     var superHeroes = request.response;  
3 |     populateHeader(superHeroes);  
4 |     showHeroes(superHeroes);  
5 | }
```

Here we are storing the response to our request (available in the `response` property) in a variable called `superHeroes`; this variable will now contain our JSON! We are then passing that JSON to two function calls — the first one will fill the `<header>` with the correct data, while the second one will create an information card for each hero on the team, and insert it into the `<section>`.

We have wrapped the code in an event handler that runs when the load event fires on the request object (see `onload`) — this is because the load event fires when the response has successfully returned; doing it this way guarantees that `request.response` will definitely be available when we come to try to do something with it.

Populating the header

Now we've retrieved our JSON data, let's make use of it by writing the two functions we referenced above. First of all, add the following function definition below the previous code:

```
1 | function populateHeader(jsonObj) {  
2 |     var myH1 = document.createElement('h1');  
3 |     myH1.textContent = jsonObj['squadName'];  
4 |     header.appendChild(myH1);  
5 | }
```

```
6   var myPara = document.createElement('p');
7   myPara.textContent = 'Hometown: ' + jsonObj['homeTown'] + ' // Formed:
8   header.appendChild(myPara);
9 }
```

We have called the parameter `jsonObj`, so that's what we need to call the JSON object inside it. Here we first create an `<h1>` element with `createElement()`, set its `textContent` to equal the `squadName` property of the JSON, then append it to the header using `appendChild()`. We then do a very similar operation with a paragraph: create it, set its text content and append it to the header. The only difference is that its text is set to a concatenated string containing both the `homeTown` and `formed` properties of the JSON.

Creating the hero information cards

Next, add the following function at the bottom of the code, which creates and displays the superhero cards:


```
1  function showHeroes(jsonObj) {
2    var heroes = jsonObj['members'];
3
4    for (var i = 0; i < heroes.length; i++) {
5      var myArticle = document.createElement('article');
6      var myH2 = document.createElement('h2');
7      var myPara1 = document.createElement('p');
8      var myPara2 = document.createElement('p');
9      var myPara3 = document.createElement('p');
10     var myList = document.createElement('ul');
11
12     myH2.textContent = heroes[i].name;
13     myPara1.textContent = 'Secret identity: ' + heroes[i].secretIdentity;
14     myPara2.textContent = 'Age: ' + heroes[i].age;
15     myPara3.textContent = 'Superpowers: ';
16
17     var superPowers = heroes[i].powers;
18     for (var j = 0; j < superPowers.length; j++) {
19       var listItem = document.createElement('li');
20       listItem.textContent = superPowers[j];
21       myList.appendChild(listItem);
22     }
23
24     myArticle.appendChild(myH2);
25     myArticle.appendChild(myPara1);
26     myArticle.appendChild(myPara2);
27     myArticle.appendChild(myPara3);
28     myArticle.appendChild(myList);
29
30     section.appendChild(myArticle);
31 }
```


```
32 |   }  
    | }  
    |
```

To start with, we store the `members` property of the JSON in a new variable. This array contains multiple objects that contain the information for each hero.

Next, we use a [for loop](#) to loop through each object in the array. For each one, we:

1. Create several new elements: an `<article>`, an `<h2>`, three `<p>`s, and a ``.
2. Set the `<h2>` to contain the current hero's `name`.
3. Fill the three paragraphs with their `secretIdentity`, `age`, and a line saying "Superpowers:" to introduce the information in the list.
4. Store the `powers` property in another new variable called `superPowers` — this contains an array that lists the current hero's superpowers.
5. Use another `for` loop to loop through the current hero's superpowers — for each one we create a `` element, put the superpower inside it, then put the `listItem` inside the `` element (`myList`) using `appendChild()`.
6. The very last thing we do is to append the `<h2>`, `<p>`s, and `` inside the `<article>` (`myArticle`), then append the `<article>` inside the `<section>`. The order in which things are appended is important, as this is the order they will be displayed inside the HTML.

 **Note:** If you are having trouble getting the example to work, try referring to our [heroes-finished.html](#) source code (see it [running live](#) also.)

 **Note:** If you are having trouble following the dot/bracket notation we are using to access the JSON, it can help to have the [superheroes.json](#) file open in another tab or your text editor, and refer to it as you look at our JavaScript. You should also refer back to our [JavaScript object basics](#) article for more information on dot and bracket notation.

Converting between objects and text

The above example was simple in terms of accessing the JSON, because we set the XHR to return the response already in JSON format, using:

```
1 | request.responseType = 'json';
```

But sometimes we aren't so lucky — sometimes we'll receive some JSON data formatted as a text string, and we'll want to convert it to an object. And when we want to send JSON data as some kind of message, we'll often need to convert it to a string for it to work correctly. Luckily, these two problems are so common in web development that a built-in [JSON](#) object was added to browsers quite a while ago, containing the following two methods:

- [parse\(\)](#): Accepts a JSON object in text string form as a parameter, and returns the corresponding object.

- `stringify()`: Accepts a JSON object as a parameter, and returns the equivalent text string form.

You can see the first one in action in our [heroes-finished-json-parse.html](#) example (see the [source code](#)) — this does exactly the same thing as the example we built up earlier, except that we set the XHR to return the JSON as text, then used `parse()` to convert it to an actual JSON object. The key snippet of code is here:

```
1 request.open('GET', requestURL);
2 request.responseType = 'text'; // now we're getting a string!
3 request.send();
4
5 request.onload = function() {
6     var superHeroesText = request.response; // get the string from the resp
7     var superHeroes = JSON.parse(superHeroesText); // convert it to an obje
8     populateHeader(superHeroes);
9     showHeroes(superHeroes);
10 }
```

As you might guess, `stringify()` works the opposite way. Try entering the following lines into your browser's JavaScript console one by one to see it in action:

```
1 var myJSON = { "name": "Chris", "age": "38" };
2 myJSON
3 var myString = JSON.stringify(myJSON);
4 myString
```

Here we're creating a JSON object, then checking what it contains, then converting it to a string using `stringify()` — saving the return value in a new variable — then checking it again.

Summary

In this article, we've given you a simple guide to using JSON in your programs, including how to create and parse JSON, and how to access data locked inside it. In the next article, we'll begin looking at object-oriented JavaScript.

See also

- [JSON object reference page](#)
- [XMLHttpRequest object reference page](#)
- [Using XMLHttpRequest](#)
- [HTTP request methods](#)
- [Official JSON web site with link to ECMA standard](#)

[← Previous](#)[Next →](#)[↑ Overview: Objects](#)

Was this article helpful?



Learn the best of web development



Get the latest and greatest from MDN delivered straight to your inbox.

[SIGN UP NOW](#)

