Intro to JSON

Javascript Object Notation

Table of Contents

[Structured Data](#_sl3elrmmsg13)

[Synchronous VS Asynchronous](#_jvttilpd5y2r)

[JSON Syntax](#_gjy72rqxrpyl)

[Loading JSON](#_1quvmzikkjif)

[Weather Example](#_j6x14khjh090)

## Structured Data

In today’s modern apps, most data comes from an outside source. Using databases to control and parse that data would be cumbersome and tedious. Instead, flat files are used to handle data exchange.

Until a few years ago, the most common vehicle for this data was XML (extensible markup language). It was so widely used that XML was included in modern web methodology acronyms like [AJAX](https://en.wikipedia.org/wiki/Ajax_(programming)). XML was used to structure data much like what an HTML would look like. Data was contained within tags. However, for simple data transfers, XML was too verbose. For simple data transfers, JSON, with a much simpler interface, should be used.

## Synchronous VS Asynchronous



*Synchronous requests* mean that before the state of a webpage can change, a call to a web server must be made. This results in a page-refreshing.

*Asynchronous requests* remove the need for page refreshes. The JSON can act as an intermediary between the client and the web server. Calls to change the state of a page now only have to be done with the JSON file.

## JSON Syntax

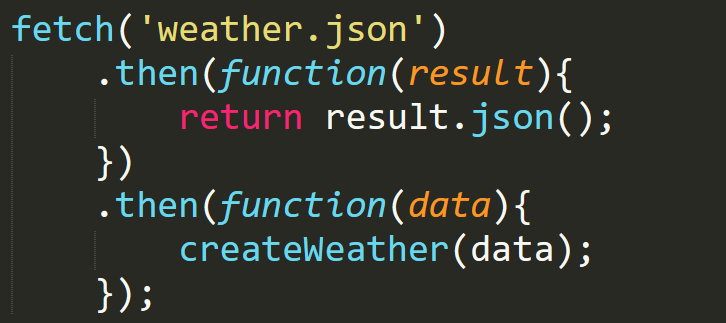
JSON syntax is based on Javascript objects. It’s made up of *key and value pairs* that describe and contain the data.



Here, there are 3 records of weather data. Each entry is for another city. Notice the keys (*name, temperature, ...*) and the values (*Winnipeg, 27, ...*).

## Loading JSON

We need to load the JSON file into our Javascript so we can parse and then use the data. **NOTE:** Keep in mind that in this course, our priority is to parse and use the JSON, not how it’s loaded. That will be covered more indepthly in the next course.



Here we’re retrieving JSON using the *fetch API*. This will retrieve JSON via a URL that’s passed to it. The *then* methods will first, handle the retrieval of JSON, and second, handle the next steps of execution. In this case, passing the data to a function called createWeather.

What is of note here is the passing of functions as parameters in the *then* methods. These are functions that will execute once the data has been retrieved. These are called *callbacks.*

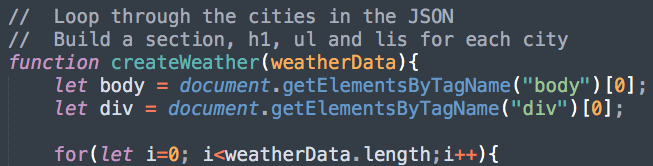
A *callback* is any function (anonymous or otherwise) that will eventually get called back to the calling scope.

This is how asynchronous works. This code is running based on the promise that the JSON will be loaded properly.

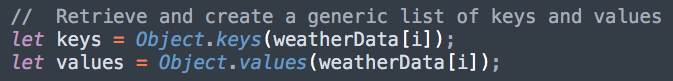
## Weather Example

Using the JSON above, let’s create some HTML elements with values derived from the JSON file.

**Important Note:** Unless you are using a web server, only Firefox will work when importing JSON files. Chrome blocks all external data sources unless you are running your pages on a web server.



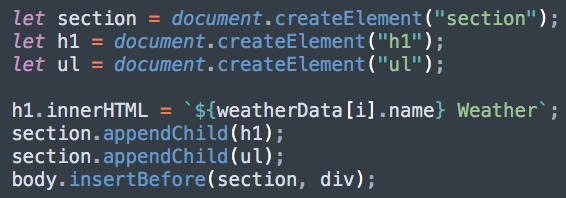
First we’re grabbing the body and div elements from the markup. These will be used to add our newly created elements to. Next we’re going to loop through the *weatherData* variable, which contains the JSON that was parsed earlier.



By using the *Object.keys* method, we can retrieve all the keys in our iteration through weatherData (if this was the first time through, we would be accessing the *Winnipeg* record). These keys will be used to create data for labels that will be created in a bit.

The *Object.values* method is similar, but it retrieves all the values from this iteration in weatherData.

Using these methods is never required, but for a simple example, it makes our new elements easier.



After creating a section, h1, and ul, *dot notation* is being used to retrieve the value associated with the *name* key. This value is used in a template literal to add content to the h1 element.

After all elements have content, they are then inserted into the markup using the [insertBefore method](https://developer.mozilla.org/en-US/docs/Web/API/Node/insertBefore). Previously, appendChild was the only way we’d seen to insert a new element into the markup. insertBefore allows more control, as we can state exactly where in the markup the new element is to be inserted, rather than just added to the end.

To finish our example, an li, label, and span element will all be created and have data from the JSON added to it. A loop is going to be used to make the code as generic as possible.



When viewed in Firefox, this is what the resulting page should look like:

## 

JSON is a simple, but powerful tool in our web development skill set. In the next course there will be more complex uses of JSON, but an introduction to how it works is an important asset.