Riverside JS Workshop
JavaScript Skills Ladder, Lesson 2.1

# Introduction to JSON

Welcome to of the Core JavaScript Skills Ladder, Phase 2! In this phase, you'll learn advanced JavaScript, JSON and AJAX and how to use them to suit your professional and creative goals.

---

### *Phase 2 Objectives*

***When you complete phase 2, you will be able to:***

- *use JSON to serialize data for storage in the browser or on the server.*
- *store and retrieve data using Ajax and LocalStorage.*
- *optimize your DOM manipulation code with Document Fragments.*
- *use Strings and Dates more effectively in your code.*
- *catch errors with Exceptions.*
- *add location and maps to your applications with Geolocation and Google Maps.*
- *build a dynamic, interactive, front-end web application.*

---

## Review of Arrays and Objects

In Core JavaScript Skills Ladder, Phase 1, you learned about arrays and objects. You learned that an array is a collection of values, usually related in some way. For instance, you can create an array of colors like this:

***OBSERVE:***

```
var springColors = [ "AF7575", "EFD8A1", "BCD693", "AFD7DB",
"3D9CA8" ];
```

...or an array of temperatures like this:

***OBSERVE:***

```
var temperatures = [ 47.5, 63.2, 56.7, 53, 51.2 ];
```

You also learned about objects. You learned that an object usually describes a thing, for instance a pet cat, and consists of properties of that thing, each of which has a name and a value:

***OBSERVE:***

```
var pickles = {
    type: "cat",
    name: "Pickles",
    weight: 7
};
```

Let's make a simple HTML page with a JavaScript program to create an array and an object, and display their values in the console.  Create a new file in as shown:

***CODE TO TYPE:***

```
<!doctype html>
<html>
<head>
  <title>Introduction to JSON</title>
  <meta charset="utf-8">
  <script src="intro.js"></script>
</head>
<body>

</body>
</html>
```

Save this as *intro.html* in your work folder.  As you can see, this web page has no content, but it does have a link to a JavaScript file, *intro.js*. Let's make that next.  Create a new file and enter the code shown below.

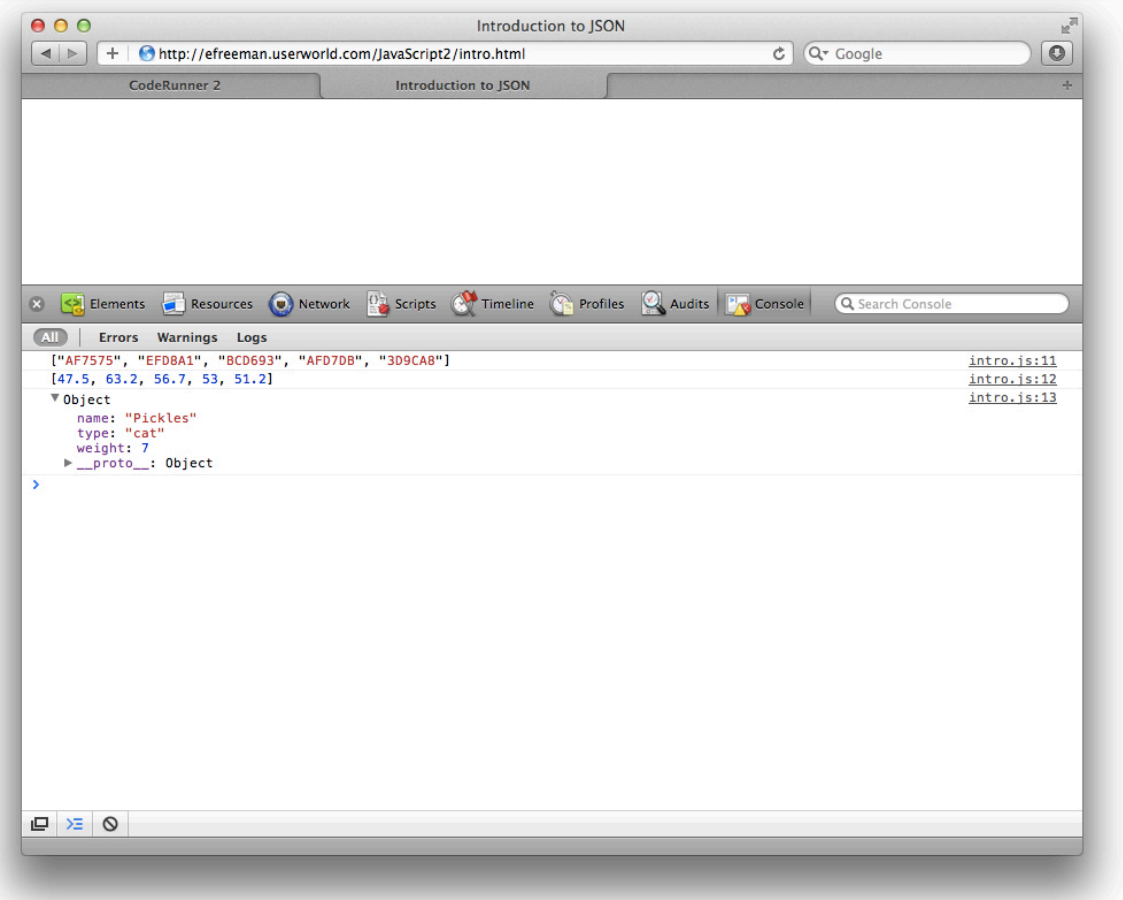
***CODE TO TYPE:***

```
var springColors = [ "AF7575", "EFD8A1", "BCD693", "AFD7DB", "3D9CA8"
];
var temperatures = [ 47.5, 63.2, 56.7, 53, 51.2 ];
var pickles = {
    type: "cat",
    name: "Pickles",
    weight: 7
};
window.onload = init;

function init() {
    console.log(springColors);
    console.log(temperatures);
    console.log(pickles);
}
```

Save this file as *intro.js* in your work folder.  Now, open *intro.html* in a browser, and then open the console in your browser's developer tools.  You should see the results of the JavaScript code displayed in the console window.

The developer tools are a little different in every browser, and they changing frequently as new versions of browsers are released, so you may have to be a little industrious and do some experimenting on your own.

When you open the console, if you don't see any results, just reload the page (intro.html). You'll see the values of the two arrays and the object.




Take a careful look at how these values are displayed; we'll come back to compare these values with other values shortly.

## JavaScript Object Notation (JSON)

In Phase Two of the Core JavaScript Skills Ladder, you'll learn how to communicate with web services from your JavaScript programs.  A web service is a file or program on the web that often provides data you can use in your web application.  But what kind of data, you might be

asking? All kinds! There are many types of web services available on the internet that offer their data for use (Twitter, for example).

When you get data from a web service—even if it's just a local file on your own computer—you need to know what format the data is in. It could be just a plain text file containing words, or a comma-separated values (CSV) file, or even an XML file (eXtensible Markup Language, a common data interchange format).

The format we'll use in this course is called *JavaScript Object Notation* or *JSON*. Why are we using this format? Because the data format is almost identical to the way you write arrays and objects in JavaScript, so it's easy for you (a human) to read, and easy for JavaScript to understand! It's also a common format used to represent data in web applications.

Let's start by looking at an example of an array and an object written in JSON format. First, here's the `springColors` array written in JSON, and also as we wrote it earlier in our JavaScript program:

***OBSERVE:***

```
JSON: ["AF7575","EFD8A1","BCD693","AFD7DB","3D9CA8"]

JavaScript: var springColors = [ "AF7575", "EFD8A1", "BCD693",
"AFD7DB", "3D9CA8" ];
```

Compare the two formats, here and where it printed to the console. Pretty similar, right? What about the temperatures array?

***OBSERVE:***

```
JSON: [47.5,63.2,56.7,53,51.2]

JavaScript: var temperatures = [ 47.5, 63.2, 56.7, 53, 51.2 ];
```

Again, compare these to how the array looks when it's printed in the console. It looks the same, right?

Finally, let's take a look at the `pickles` object:

***OBSERVE:***

```
JSON: {"type":"cat","name":"Pickles","weight":7}
JavaScript: var pickles = {
    type: "cat",
```

```
    name: "Pickles",
    weight: 7
};
```

This time, there's a slight difference: each of the property names ("type," "name," and "weight") is enclosed in double quotation marks. Otherwise, the JSON object is written the same way as the object in JavaScript. Note that the JSON object is different from objects displayed in the console, but that doesn't affect how you write the code to create the objects.

JSON can represent all of the core values you typically write in JavaScript: strings, numbers, and Boolean values, as well as arrays and objects that use these kinds of values in them. JSON can even represent objects that contain other objects and arrays. For instance, you can add the "likes" property to the `pickles` object, like this:

***CODE TO TYPE:***

```
var springColors = [ "AF7575", "EFD8A1", "BCD693", "AFD7DB", "3D9CA8"
];
var temperatures = [ 47.5, 63.2, 56.7, 53, 51.2 ];
var pickles = {
    type: "cat",
    name: "Pickles",
    weight: 7,
    likes: ["sleeping", "purring", "eating butter"]
};
window.onload = init;

function init() {
    console.log(springColors);
    console.log(temperatures);
    console.log(pickles);
}
```

Save the file changes, and reload the page in your browser. The `pickles` object in the console now displays the `likes` property. If we represent the `pickles` object in JSON, it looks like this:

***OBSERVE:***

```
{"type":"cat","name":"Pickles","weight":7,"likes":["sleeping",
"purring", "eating butter"]}
```

The only difference in the JSON representation of the `likes` property of the object and the JavaScript is that in JSON, the property name is enclosed in double quotation marks ("likes").

So, *why* would you want to represent an object or array in JSON format, and *how* do you do it? I'm glad you asked...

## Why Use JSON?

Imagine you run a pet adoption center, and you need a web application that does two things: allows you to enter new pets up for adoption, and shows you all the current pets available for adoption.

Instead of editing the whole HTML page each time you make a change to the list of pets for adoption, you want to be able to load the images of individual pets dynamically into your page whenever the page is loaded. If you store the data separate from the HTML and the JavaScript, then you won't have to edit the page each time (and risk making a mistake) and it will be a lot easier to update.

In addition, you have to represent the data about the pets somehow. You need to store the data in a format that your web application can understand. You could invent a format, or you could use an existing format like CSV, XML, or...JSON! Using JSON makes getting the data into your application really convenient. Let's look at how you might represent a list of pets available for adoption using JSON:

***OBSERVE:***

```
["Pickles", "Tilla"]
```

This is an array of all the pets available for adoption. It includes only their names; you could have much more information about each pet if you use objects. Create a new file that looks like this:

***CODE TO TYPE:***

```
[ { "type":"cat",
    "name":"Pickles",
    "weight":7,
    "likes":["sleeping","purring","eating butter"]
  },
  { "type":"dog",
    "name":"Tilla",
    "weight":25,
    "likes":["sleeping","eating","walking"]
  }
]
```

Save the file in your work folder as *pets.json*.

**Note** Make sure to type the code above *just as you see it*! You can copy and paste the JSON into your file if you want, just to be certain.

We're not going to do anything with this file just yet. We'll get to that in a later lesson. For now, we'll learn how to turn an array of objects, just like the one above, into a JSON string in your JavaScript.

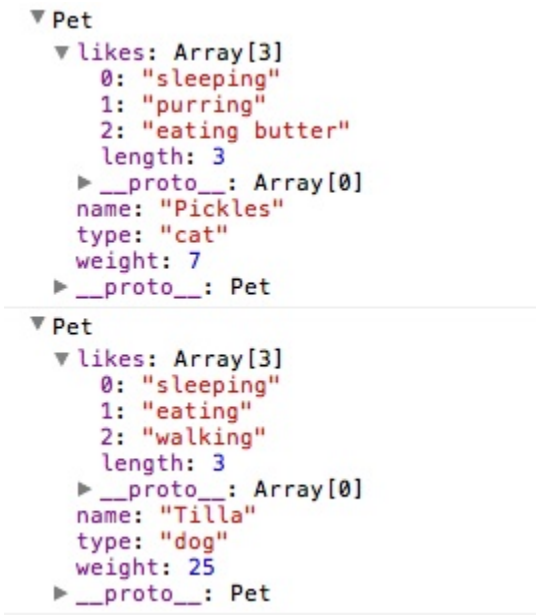
## How to Use JSON in JavaScript (Serializing a JavaScript Object)

Let's say you have a JavaScript program with two pet objects. Update *intro.js* as shown:

***CODE TO TYPE:***

```
var springColors = [ "AF7575", "EFD8A1", "BCD693", "AFD7DB", "3D9CA8"
];
var temperatures = [ 47.5, 63.2, 56.7, 53, 51.2 ];
var pickles = {
    type: "cat",
    name: "Pickles",
    weight: 7,
    likes: ["sleeping", "purring", "eating butter"]
};

function Pet(type, name, weight, likes) {
    this.type = type;
    this.name = name;
    this.weight = weight;
    this.likes = likes;
}

window.onload = init;

function init() {
    console.log(springColors);
    console.log(temperatures);
    console.log(pickles);

    var pickles = new Pet("cat", "Pickles", 7, ["sleeping", "purring",
"eating butter"]);
    console.log(pickles);

    var tilla = new Pet("dog", "Tilla", 25,
["sleeping","eating","walking"]);
    console.log(tilla);

}
```

Save the changes to your file, and open or refresh *intro.html* in your browser.  In the console, you see the two Pet objects you created:




Now, let's turn these objects into JSON. Update *intro.js* as shown:

***CODE TO TYPE:***

```
function Pet(type, name, weight, likes) {
    this.type = type;
    this.name = name;
    this.weight = weight;
    this.likes = likes;
}

window.onload = init;

function init() {
    var pickles = new Pet("cat", "Pickles", 7, ["sleeping", "purring",
"eating butter"]);
    console.log(pickles);
    var picklesJSON = JSON.stringify(pickles);
    console.log(picklesJSON);

    var tilla = new Pet("dog", "Tilla", 25,
["sleeping","eating","walking"]);
    console.log(tilla);
    var tillaJSON = JSON.stringify(tilla);
    console.log(tillaJSON);
}
```

Save the changes to your file, and open or refresh *intro.html* in your browser. In the console, you see the two Pet objects you created as before, and under each one, their JSON representations:


Compare the JavaScript objects with the JSON versions. They are very similar.

## JSON.stringify()

To convert a JavaScript object to JSON, use the built-in `JSON` object and its method, `stringify()`:

***OBSERVE:***

```
var pickles = new Pet("cat", "Pickles", 7, ["sleeping",
"purring", "eating butter"]);

var picklesJSON = JSON.stringify(pickles);
```

First, we create a `Pet` object, `pickles`. You've seen this kind of JavaScript before. Then, we use the `stringify()` method of the `JSON` object, and pass in the pickles object as the argument. We get back a JSON version of the object, which we store in the variable `picklesJSON`. We did the same thing with *tilla* to turn the `tilla` object into JSON.

**Note** The **JSON** object is built-in to JavaScript in all *modern* browsers, just like the **document** object, but you need to make sure you're using a fairly recent version of your favorite browser. That means IE9+, Safari 5+, Chrome 18+, Firefox 11+, or Opera 11+. Some older versions of some of these browsers have the JSON object, but for this course, use one of these versions (or later).

So now let's create an array of the two objects, and convert the array to JSON. Update *intro.js* as shown:

***CODE TO TYPE:***

```
function Pet(type, name, weight, likes) {
    this.type = type;
    this.name = name;
    this.weight = weight;
    this.likes = likes;
}

window.onload = init;

function init() {
    var pickles = new Pet("cat", "Pickles", 7, ["sleeping", "purring",
"eating butter"]);
    console.log(pickles);
    var picklesJSON = JSON.stringify(pickles)
    console.log(picklesJSON);

    var tilla = new Pet("dog", "Tilla", 25,
["sleeping","eating","walking"]);
    console.log(tilla);
    var tillaJSON = JSON.stringify(tilla);
    console.log(tillaJSON);

    var petsArray = [ pickles, tilla ];
    var petsArrayJSON = JSON.stringify(petsArray);
    console.log(petsArrayJSON);
}
```

We are passing an array to `JSON.stringify()`. This is not a problem. As with objects, you can turn an array into JSON.

Save the changes to your JavaScript file, an open or refresh *intro.html* in your browser. In the console, you see the two Pet objects as before. Underneath the two objects, you see the JSON string representation of the array containing the two objects:


Compare the JavaScript objects' appearance with the appearance of the JSON-formatted array of objects. JSON is called "JavaScript Object Notation" because it looks just like JavaScript objects. The advantage is that a JSON-formatted object is a *string*. That means you can store it in a file, just like you did before when you created the *pets.json* file.

If you compare the output in the console that shows the JSON formatted array to what you typed into *pets.json*, you'll see they are *exactly the same* (ignore the extra white space you added in the file).

Turning an object into a string like this is known as *serializing an object*. It is incredibly useful because it allows you to store objects in plain text files, or transfer them between web applications, even if those applications are written in different languages. Let's say you have a Pet object in JavaScript, how would you give that object to, say, a PHP program? The only way you can do that is to serialize the object. Once the object is serialized, you can give it to the PHP program, and if the PHP program knows that the format is JSON, the program can *deserialize* the object (turn the string back into an object again), and voila! You've just transferred an object between two programs written in entirely different languages. That's cool.

## Deserializing an Object

What if you've got an object represented in JSON and you want to get the object back? Update **intro.js** as shown:

***CODE TO TYPE:***

```
function Pet(type, name, weight, likes) {
    this.type = type;
    this.name = name;
    this.weight = weight;
    this.likes = likes;
}

window.onload = init;

function init() {
```

```
    var pickles = new Pet("cat", "Pickles", 7, ["sleeping", "purring",
"eating butter"]);
    console.log(pickles);
    var picklesJSON = JSON.stringify(pickles);
    console.log(picklesJSON);

    var anotherPickles = JSON.parse(picklesJSON);
    console.log(anotherPickles);

    var tilla = new Pet("dog", "Tilla", 25,
["sleeping","eating","walking"]);
    console.log(tilla);

    var petsArray = [ pickles, tilla ];
    var petsArrayJSON = JSON.stringify(petsArray);
    console.log(petsArrayJSON);
}
```

Save the changes to your JavaScript file, and open or refresh *intro.html* in your browser. In the console, you see the JSON representation of the `pickles` object, and below that you see a new `Pet` object that was converted back into an object from JSON:


So what did we just do?

***OBSERVE:***

```
var pickles = new Pet("cat", "Pickles", 7, ["sleeping", "purring",
"eating butter"]);
var picklesJSON = JSON.stringify(pickles);
console.log(picklesJSON);
```

First, we created a new pickles object. Then we converted that object to JSON using `JSON.stringify()`, and stored the result in the variable `picklesJSON`. Next, we printed `picklesJSON` to the console.

***OBSERVE:***

```
var anotherPickles = JSON.parse(picklesJSON);
console.log(anotherPickles);
```

We took `picklesJSON` and we passed it to the `JSON.parse()` method. `JSON.parse()` does the opposite of what `JSON.stringify()` does: it takes a piece of data formatted as JSON, and converts it back into a JavaScript object (or array). This returns an object, `anotherPickles`, which we print to the console.

It's important to recognize that you get back another object, *not* the same object as our original pickles object. It's like `pickles`, in the sense that it has the same properties and property values, but now we have two *different* objects, `pickles` and `anotherPickles`.

So, `JSON.stringify()` takes an object and *serializes* it into JSON format; and `JSON.parse()` takes a piece of data in JSON format and *deserializes* it into a JavaScript object.

This allows you to transfer objects (or arrays) between JavaScript programs, between web services and web applications, and beyond. In the next couple of lessons, you'll see how you can use JSON with Ajax (a way of communicating with external resources from your JavaScript program) to load and save data from your web application.

## JSON or XML?

You may have heard of XML or even used it in web applications before, as a data exchange format. You might be wondering whether we could also use XML to represent objects and arrays, and exchange data with web services. Yes, we could. In general, you may choose to use XML or JSON to represent your data; it depends on the context, the type of data, the web application, or the other programs or web services with which you might be exchanging data.

The good news is that anything you can represent in XML, you can also represent in JSON. While XML has additional features that execute tasks like validating your data to make sure it's correct, the kinds of data that can be represented are the same. So why choose JSON?

JSON is simpler to parse (that is, it's simpler to convert data in JSON format to a JavaScript object or array). A piece of data in JSON format is typically smaller in size than the same data in XML, and it's somewhat easier for us humans to read. JSON is well supported in JavaScript and web application tools, but it is not so well supported in other areas (for example, library systems, content management systems, and so on), where XML has broad support. For instance, you can get text editors that are specifically designed to help you write XML.

If you want to use your data in a variety of different contexts, you might choose XML.  However, for web applications specifically, JSON is easier to work with; that's what we'll be using in this course.

Take some time to practice JSON and do the project before moving on to the next lesson.  We'll use JSON to store and retrieve values throughout the rest of the course.

## *More about the material in this lesson*

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.