**JSON Best Practices**

***What is JSON?***

JSON is a data interchange format that is easy to parse and generate. JSON is an extension of the syntax used to describe object data in JavaScript. Yet, it’s not restricted to use with JavaScript. It has a text format that uses object and array structures for the portable representation of data. All modern programming languages support these data structures, making JSON completely language independent.

***JSON Structure, Syntax, and Usage***

JSON’s simplicity is part of its appeal. It’s easy to write, easy to read, and easy to translate between the data structures used by most languages. Let’s look at what makes up a JSON object, the data types that JSON supports, and other specifics with the syntax of this popular data format.

Chances are you’ve seen JSON as you’ve looked through data or API documentation. The criteria for valid JSON is rather elementary, though it can be used to describe complex data. The structure of a JSON object is as follows:

Curly braces {} hold objects

The data are in key, value pairs

Square brackets [] hold arrays

Each data element is enclosed with quotes if it‘s a character, or without quotes if it is a numeric value

Commas are used to separate pieces of data

Here’s a basic example:

{ "name":"Katherine Johnson" }

The key is “name” and the value is “Katherine Johnson” in the above example. However, JSON can hold more than one key:value pair. This second example adds an “age” key, which includes a number and a second string value, assigned to the “city” key:

{ "name":"Katherine Johnson", "age":101, "city":"Newport News" }

It’s common to encounter nested JSON structures, like this:

{"inventors":[

{ "name":"Katherine Johnson", "age":101, "city":"Newport News" },

{ "name":"Dorothy Vaughan", "age":98, "city":"Hampton" },

{ "name":"Henry Ford", "age":83, "city":"Detroit" }

]}

In this final example, you see a primary object with a single key (“inventors”) that has an array as its value. Within that array, each item is itself an object, similar to the earlier simple example. Objects and arrays are values that can hold other values, so there’s an unlimited nesting that could happen with JSON data. That allows JSON to describe most data types, from tabular to even more complex.

***JSON Data Types***

Now that you’ve seen the structure of JSON, you’ve been introduced to several of its data types. There are only a couple others to introduce. Here is the complete list of JSON data types:

string – Literal text that’s enclosed in quotes.

number – Positive or negative integers or floating point numbers.

object – A key, value pair enclosed in curly braces

array – A collection of one or more JSON objects.

boolean – A value of either true or false with no quotes.

null – Indicates the absence of data for a key value pair, represented as “null” with no quotes.

Here’s an example of a JSON object that includes all of these data types:

{

"name":"Katherine Johnson",

"age":101,

"orbital\_mechanics": ["trajectories","launch windows","emergency return paths"],

"mathmatician": true,

"last\_location": null

}

***JSON Syntax***

We’ve already discussed the structure of JSON, which provides the basics of the syntax. In this section, we’ll suggest some best practices to avoid common JSON errors:

Always enclose the key, value pair within double quotes. Most JSON parsers don’t like to parse JSON objects with single quotes.

{ "name": "Katherine Johnson" }

Never use hyphens in your key fields. Use underscores ( \_ ), all lower case, or camel case.

{ "first\_name":"Katherine", "last\_name":"Johnson" }

Use a JSON linter to confirm valid JSON. Install a command line linter or use an online tool like JSONLint. If you copy this next example into a JSON linter, you should get a parse error for the pesky single quotes around the value for last\_name.

{ "first\_name":"Katherine", "last\_name":'Johnson' }