Intro to the JSON Data Format

Contents

[What is JSON? 2](#_Toc114231144)

[Some Examples of JSON Data 3](#_Toc114231145)

[EXAMPLE 1 - storing the contents of a company "org chart" in JSON format: 3](#_Toc114231146)

[EXAMPLE 2 - JSON data describing books for sale on a website 5](#_Toc114231147)

[HOW TO EDIT AND VALIDATE JSON FILES 6](#_Toc114231148)

[CREATING AND EDITING JSON FILES 6](#_Toc114231149)

[USING A JSON VALIDATOR TOOL 6](#_Toc114231150)

[EXTRA WHITESPACE IN JSON IS IGNORED (i.e. spaces, tabs and newlines) 7](#_Toc114231151)

[DATA TYPES IN JSON 8](#_Toc114231152)

["PRIMATIVE" (or SIMPLE) JSON DATATYPES (string, number, boolean, null) 9](#_Toc114231153)

[COMPLEX DATA TYPES  -  "OBJECTS" and "ARRAYS" 10](#_Toc114231154)

[OBJECTS 11](#_Toc114231155)

[ARRAYS 13](#_Toc114231156)

[Nested Arrays 13](#_Toc114231157)

[Arrays that contain objects 13](#_Toc114231158)

[Objects that contain arrays 14](#_Toc114231159)

[Complex JSON Files 15](#_Toc114231160)

[BOTTOM LINE 16](#_Toc114231161)

[COMMON MISTAKES 16](#_Toc114231162)

# What is JSON?

JSON is pronounced like the name "Jason" or as "j sahn" - either is Acceptable. JSON is a "text" format for storing and distributing data. The following small example shows information about the USA Postal Code

(“i.e. zip code”) 60527. Many more examples appear throughout this document.

**{**

**"post code": "60527",**

**"country": "United States",**

**"places": [{**

**"place name": "Willowbrook",**

**"latitude": "41.7447",**

**"longitude": "-87.9334",**

**"state": "Illinois"**

**},**

**{**

**"place name": "Hinsdale",**

**"latitude": "41.7447",**

**"longitude": "-87.9334",**

**"state": "Illinois"**

**}**

**]**

**}**

A "text" format means that it is written all in ASCII or Unicode. ASCII and Unicode are standard technologies; every computer system has built in tools for working with ASCII and Unicode data. Therefore, JSON can be read and used on any computer system with a simple text editor without the need to install

special software to read it. CSV formats are similar in this respect. However, while CSV is suited specifically to storing information that can be arranged into "rows and columns", the JSON data format is much more flexible in how the data is arranged.

JSON was originally developed for the JavaScript language. However, today it has become a popular format that is used in many areas of technology. You can process JSON data with R or any other programming language. The name JSON stands for "JavaScript Object Notation:" - but seriously – you do NOT need to know ANY JavaScript to make use of JSON data. Unfortunately, many resources for learning JSON assume knowledge of JavaScript.

That really isn’t necessary if you are using JSON in another language. This document describes JSON for people who DO NOT know JavaScript.

# Some Examples of JSON Data

The full rules of how to create a JSON file are listed later. For now, it may help to see some typical examples of what JSON data looks like. The exact explanations of how to format a JSON file appear below the examples.

### EXAMPLE 1 - storing the contents of a company "org chart" in JSON format:

The following is an "organization chart" for a company. This is NOT in JSON format. One possible JSON format of the data appears below. The chart shows that Sue is the CEO. Frank and Anne report to Sue. Bob, Lisa

and Joe report to Frank. etc.

         Widgets R Us (founded 2022)

                 ---------------Sue (CEO) ------------

                 |                                   |

                 |                                   |

         --Frank (Dir of Operations)---         -----Anne (CFO)-----

         |             |              |         |                  |

         |             |              |         |                  |

         Bob          Lisa          Joe        Pat (CPA)         Pete

                       |

                      Larry

(continued on next page ... )

---------------------------------------------------------------------------

 ONE POSSIBLE JSON ARRANGEMENT OF THIS DATA IN JSON FORMAT

---------------------------------------------------------------------------

*Note that the JSON shown below repeats some data (i.e. the employee names are*

*repeated in both the particular section for that employee and in the list of*

*"direct reports" for that employee's mananger). There are other JSON*

*organizations of this data that would not require repeating any data. However,*

*the goal now is not to debate different alternative JSON representations, but*

*rather to give you a flavor of what JSON looks like in general.*

         {

             "company name": "Widgets R Us",

             "founded": 2022,

             "employees": [

                 {

                     "name": "Sue",

                     "title": "CEO",

                     "directReports": ["Frank", "Anne"]

                 },

                 {

                     "name": "Frank",

                     "title": "Director of Operations",

                     "directReports": ["Bob","Lisa","Joe"]

                 },

                 {

                     "name": "Anne",

                     "title": "CFO",

                     "directReports": ["Pat","Pete"]

                 },

                 {

                     "name": "Bob"

                 },

                 {

                     "name": "Lisa",

                     "directReports": ["Larry"]

                 },

                 {

                     "name": "Joe"

                 },

                 {

                     "name": "Larry"

                 },

                 {

                     "name": "Pat",

                     "title": "accountant",

                     "certified": true

                 },

                 {

                     "name": "Pete",

                     "title": "accountant",

                     "certified": false

                 }

             ]

         }

### EXAMPLE 2 - JSON data describing books for sale on a website

The following example shows a JSON file that contains information about some books.

     [

       {

         "hardcover": false,

         "title": "How to program in R",

         "copyright": 2017,

         "publisher": {

           "pubName": "Books R Us",

           "pubWebsite": "https://booksrus.com/"

         },

         "categories": ["R","technology","statistics"],

         "author" :

           {

             "first": "Robert",

             "last": "Rosen"

           }

       },

       {

         "hardcover": false,

         "title": "Python For Perfectionists",

         "copyright": 2018,

         "categories": ["Python", "technology"],

         "author" :

           {

             "title": "Dr.",

             "first": "Sue",

             "last": "Smith"

           }

       },

       {

         "hardcover": true,

         "title": "Cooking For Programmers",

         "copyright": 2018,

         "categories": ["cooking"],

         "author" :

           {

             "first": "Joe",

             "middle": "J",

             "last": "Johnson"

           }

       }

     ]

# HOW TO EDIT AND VALIDATE JSON FILES

## CREATING AND EDITING JSON FILES

You can create JSON files with any text editor (e.g. Notepad, VSCode, the RStudio editor, etc.) Many text editors will help you with editing JSON files. For example, VSCode displays different JSON datatypes (see below) in different colors and highlights any errors that appear in the JSON format (such as missing commas, missing quotes, etc). This helps you correct the errors to make sure that the JSON is "valid".   
  
In addition VSCode makes it easy to navigate large JSON files by allowing you to "collapse and expand" entire JSON objects and arrays. If you point at the line numbers with your mouse, arrows appear next to the line numbers. Click on these arrows to collapse and expand objects and arrays that span multiple lines.  
  
*Note that for VSCode to recognize a file as a JSON file you should name the file with a ".json" extension at the end of the filename.*

## USING A JSON VALIDATOR TOOL

Another way to determine if a JSON file is "valid" is to copy the contents into a JSON validation service such as https://jsonlint.com/. Try doing so with the JSON examples shown above. If you paste one of them into

https://jsonlint.com/  and press the "Validate JSON" button, you should see "valid JSON" in the Results section.

## EXTRA WHITESPACE IN JSON IS IGNORED (i.e. spaces, tabs and newlines)

Formatting JSON files with spaces, tabs and newlines to clearly show the structure of the data is highly recommended. However, strictly speaking, whitespace (i.e. spaces, tabs and newlines) that does not appear between quotes do not affect the "validity" of the JSON. For example, the following would also be a valid  
JSON representation of the data from the last example. Of course, spaces that appear between quotes, such as "Books R Us" are not ignored.

The following IS valid JSON (but don't do this):

[{"hardcover": false,"title": "How to Program in R"  
,"copyright":2017,"publisher": {"pubName": "Books R Us","pubWebsite":

"https://booksrus.com/"},"categories":

["R","technology","statistics"],"author" :{"first": "Robert","last":

"Rosen"}},{"hardcover": true,"title": "Python For Perfectionists","copyright":

2018,"categories": ["Python", "technology"],"author" :{"title": "Dr.","first":

"Sue","last": "Smith"}},{"hardcover": true,"title": "Cooking For Programmers","copyright":

2018,"categories": ["cooking"],"author" :{"first":

"Joe","middle": "J","last": "Johnson"}}]

*Note that any "quoted information" must all appear on one line. For example, the quoted title* "How to Program in R" *appears at the very end of the fist line of the JSON shown above. If the title were split between the 1st and 2nd lines, the JSON would NOT be valid.*

Removing all the unnecessary whitespace (spaces, tabs and newlines) from a JSON file is known as "minifying" the JSON file. Sometimes JSON files are intentionally "minified". Specifically, this often is done to reduce the amount of data that needs to be transferred when JSON data is sent from one computer to another via an API. For right now, don't worry about APIs. We will cover APIs very soon.

Most JSON editors have features for automatically "prettifying" the JSON file so that it is indented properly. For example, in VSCode you can press Shift + Alt + F (Windows) or Shift + Option + F (Mac) to format a JSON document. See this page (or search online for others) for more tips on using VSCode to edit

JSON documents: https://linuxpip.org/vscode-format-json/

# DATA TYPES IN JSON

Individual pieces of data in a JSON file must be one of the following simple

data types: string, number, boolean and null (these are described in the

following section). In addition, multiple data values can be arranged into

more complex structures, i.e. objects and arrays (see below).

## 

*(continued on next page)*

## "PRIMATIVE" (or SIMPLE) JSON DATATYPES (string, number, boolean, null)

Each individual piece of data in a JSON file must be one the of the following data types: string, number, boolean, null. Technically a file is a valid JSON file if it contains just a single number, a single string, a single true or false or a single null. Practically speaking though, most JSON files arrange mulitple pieces of data into objects and arrays as described in the next section.

The simple (or primative) datatypes are:

   - "string": string is just another way of saying "character" data.

               Strings in JSON must be enclosed in "double quotes" (i.e. quotation marks).

               You may NOT use 'single quotes' (i.e. apostrophes) to enclose strings.

               Strings may contain the following "backslash escape sequences", i.e.

characters preceded by a backslash:

               \"  -  Double quote                \\  -  Backslash

               \/  -  Forward slash               \b  -  Backspace

               \f  -  Form feed                   \n  -  Newline

               \r  -  Carriage return             \t  -  Tab

               \u  - followed by four hexadecimal digits indicates the

                     Unicode character with the specified "code point". For

                     example, \u05D0 represents a Hebrew letter Aleph. Here are

                     all codepoints for the Hebrew alphabet:

                     https://unicodeplus.com/block/0590 (for more info see:

                     https://developer.mozilla.org/en-US/docs/Glossary/Code\_point)

   - number: 123 and 123.45 are both valid numbers. Do not put quotes around

             numbers. A number with quotes around it is considered a "string". Software

             that processes the JSON will not associate a numeric value with numbers that

             have "quotes" around them.

   - boolean: boolean is just another way of saying "logical" data. A boolean

              value can be either true or false. The words true/false must be lowercase.

              Do not enclose the true and false in quotes. A true or false with quotes around

              it is considered a "string".

   - null: A value that is null is considered non-existant. For example, the

           following JSON is used to represent some data about the ACME Shipping

           Company. Notice that some employees have a middle name and some don't. The

           ones who don't have a middle name have a value of null for the middle name.

               {

                 "company": "ACME Shipping Company",

                 "employees": [ {"first": "Joe",     "middle": null,     "last": "Smith"},

                                {"first": "Sheila",  "middle": null,     "last": "Cohen"},

                                {"first": "Jack",    "middle": "Edgar",  "last": "Jones"} ]

               }

           Note that another way to indicate that an employee doesn't have a

           middle name is just to leave out the middle name for that employee.

           For example, the following is also valid JSON:

               {

                 "company": "ACME Shipping Company",

                 "employees": [ {"first": "Joe",                          "last": "Smith"},

                                {"first": "Sheila",                       "last": "Cohen"},

                                {"first": "Jack",     "middle": "Edgar",  "last": "Jones"} ]

               }

           The choice of whether to use null or to simply leave out the data is

           a choice made by the people who design the data format. Either are

           valid choices. As mentioned above, the exact "spacing" of the data does

           not affect the validity of the JSON file. The spacing is very useful

           though to make the file easier to read.

## COMPLEX DATA TYPES  -  "OBJECTS" and "ARRAYS"

 "Objects" and "arrays" are two different "complex data types" that allow you

 to organize multiple data values in different ways.

*(continued on next page)*

### OBJECTS

 A JSON "object" is surrounded by a single set of {curly braces}. Inside the curly braces are zero or more "name":value pairs (also known as "key":value pairs).

 For example, the following is a valid JSON object:

                 {

                     "recipeName": "Midnight Chocolate Cake",

                  "submittedBy": "Jack Sep 1, 2022 9:00am",

                     "caloriesPerServing": 450,

                     "dietMenu": false,

                     "prepTime": "2 hours 45 minutes"

                 }

 Notice a few things about this example:

 1. The data for the "object" is surrounded by { curly braces }.

 2. Each line inside the {curly braces} contains a "name":value pair  (AKA "key":value pair).

    Note that it is NOT required that each "name":value pair be on a separate line

    but it is considered good practice to make the files more readable.

 3. The "name" in each "name":value pair (AKA the "key" in each "key":value pair) is enclosed in quotes.

    It's not uncommon to see JSON files that incorrectly leave out the quotes around the names.  This is  
 wrong. While these files may sometimes still be able to be processed correctly by software that accounts  
 for this mistake, you should always make sure to include the "quotes" around the "names" in JSON files  
 that you create yourself.

    Note that technically the "name" (AKA "key") in a name value pair may contain spaces. However, including  
 spaces in names can sometimes cause problems when using software to process a JSON file. Therefore it is

recommended not to use spaces in names. The example above uses "camelCase" for the names to clearly

identify each word in a multiWord name.

 4. The value in each "name":value pair may be any valid JSON datatype. The different "name":value pairs in

the example above use different data types. For example "Midnight Chocolate Cake" is a string, 450 is a

number and false is a boolean value. Note that "45 minutes" is a string and not a number.

 5. The "name":value pairs are separated from each other with commas. Note that there is NO comma after the

last "name":value pair in the object (i.e. no comma after "dietMenu": false). Extra commas and missing

commas are a common source of errors that can be hard to find.

 6. The same "name" may not appear twice in a single object. For example, the following is NOT valid!

       { "thing": "ball", "thing": "cookie" }        # THIS IS NOT VALID - DUPLICATE NAMES

NESTED OBJECTS

 The values in "name":value pairs (AKA "key":value pairs) may be objects.  
 Below, is a modified version of the recipe example shown above. Notice  
 that in the following version of the JSON, the value for "prepTime" is now  
 itself an object. An object inside an object is known as a "nested object".

                 {

                     "recipeName": "Midnight Chocolate Cake",

                     "submittedBy": "Jack Sep 1, 2022 9:00am",

                     "caloriesPerServing": 450,

                     "dietMenu": false,

                     "prepTime": {

                         "hours": 2,

                         "minutes": 45

                     }

                 }

 Continuing with the same example ... we added another "name":value pair whose

 name is "submittedBy". The value of "submittedBy" is itself a nested object.

                 {

                     "recipeName": "Midnight Chocolate Cake",

                     "submittedBy": {

                         "userName": "Jack",

                         "when": {

                             "date": "Sep 1, 2022",

                             "time": "9:00 am"

                         }

                     },

                     "caloriesPerServing": 450,

                     "dietMenu": false,

                     "prepTime": {

                         "hours": 2,

                         "minutes": 45

                     }

                 }

 Continuing once again with the same example ... we can also make the values for   
 "date" and "time" into nested objects:

                 {

                     "recipeName": "Midnight Chocolate Cake",

                     "submittedBy": {

                         "userName": "Jack",

                         "when": {

                             "date": {

"month": "Sep",

"day": 1,

"year": 2022

},

                             "time": {

"hour": 9,

"minute": 0,

"am": true

}

                         }

                     },

                     "caloriesPerServing": 450,

                     "dietMenu": false,

                     "prepTime": {

                         "hours": 2,

                         "minutes": 45

                     }

                 }

 Objects may be nested for many levels. Without proper indentation, nested

 objects (and arrays - see below) can become very difficult to read. Make sure

 that your JSON objects are indented properly. (Note that the value in a

 "name":value pair may also be an array as explained in the next section).

### ARRAYS

 JSON arrays combine a sequence of values between [square brackets]. The values

 in an array are separated from each other using commas. There is no comma

 after the last value. For example, the following is a valid JSON array that

 contains several strings:

                 ["apple", "pear", "plum"]

 Note that the values in an array are not "named" as they are in objects (see

 above). The following JSON array contains numbers:

                 [3.14159, 2.7182, 1.6180, 186282]

 There is technically no requirement that all values in a JSON array must be

 the same datatype (i.e. a "homogeneous array"). JSON does allow

 for "heterogeneous arrays", or arrays that contain values of different data

 types. These are not as common as homogenous arrays, however they are still

 considered valid JSON. For example, the following is a JSON array that

 contains values of different data types.

                 [ "apple" , 3.14159 , true , null , false , true, 92 , "abc"]

### Nested Arrays

 In addition to the simple data types (string, number, boolean, null), JSON

 arrays may also contain nested arrays as well as objects. For example, the

 following is an array that contains nested arrays.

   [ [92, 80, 70, 83, 98, 77, 86, 99, 25] , [ 96, 72 , 26, 100, 92] , [ 93, 82, 92, 85] ]

 The above example contains three different nested arrays. Often such arrays

 are arranged with each "inner" array (that is contained in the "outer"

 array) on a different line. For example, the above JSON is equivalent to the

 following:

         [

             [92, 80, 70, 83, 98, 77, 86, 99, 25],

             [96, 72, 26, 100, 92],

             [93, 82, 92, 85]

         ]

### Arrays that contain objects

 Arrays may also contain objects. For example, the following shows an array that

 contains 3 objects. Notice how the indentation is arranged to clearly show the

 structure of the array and the objects.

             [

                 {

                     "first": "Sue",

                     "last": "Smith"

                 },

                 {

                     "first": "Frank",

                     "last": "Jones"

                 },

                 {

                     "name": "Anne",

                     "title": "Applebaum"

                 }

             ]

### Objects that contain arrays

 Just as arrays can contain objects, objects can contain arrays. The value of a "name":value pair in an object may be an array (as well as an object). For example, the following is a JSON object that describes a company. Below, the array of objects from the last example is used as the value for the "employees" key in the outermost object.

 {

     "companyName": "Widgets R Us",

     "founded": 2022,

     "employees":

             [

                 {

                     "first": "Sue",

                     "last": "Smith"

                 },

                 {

                     "first": "Frank",

                     "last": "Jones"

                 },

                 {

                     "name": "Anne",

                     "title": "Applebaum"

                 }

             ]

 }

# Complex JSON Files

Very complex JSON files can be constructed from a deep nesting of arrays that contain objects and objects that contain arrays, etc. etc. etc. The outermost structure in the example above is an object. It contains information about a single compmany. If we want to capture information about many companies, we could create an array of different company objects as shown below.

 [

     {

         "companyName": "Widgets R Us",

         "founded": 2022,

         "employees":

                 [

                     {

                         "first": "Sue",

                         "last": "Smith"

                     },

                     {

                         "first": "Frank",

                         "last": "Jones"

                     },

                     {

                         "first": "Anne",

                         "last": "Applebaum"

                     }

                 ]

     },

     {

         "companyName": "Pete's Pizza",

         "founded": 2020,

         "employees":

                 [

                     {

                         "first": "Pete",

                         "last": "Jones"

                     },

                     {

                         "first": "Paula",

                         "last": "Jones"

                     }

     }

 ]

## BOTTOM LINE

 1. Every JSON file must be either:

    - A single value of a simple type, i.e.

        \* a single number

        \* a single "string"

        \* a single true or false

        \* a single null

    - a single array that may contain many other nested arrays, objects or simple types

    - a single object that may contain many other nested arrays, objects or simple types

 2. Collections of data values that have associated "names" (or "keys") for the

    individual items are arranged using "objects".

 3. Collections of data values that do not have "names" (or "keys") are

    arranged using "arrays".

 JSON files may seem a little intimidating at first. However, it is a very

 understandable concept once you take a little time to understand the rules of

 how to construct objects and arrays.

# COMMON MISTAKES

1. Missing commas where they are necessary.  
  
2. An extra comma after the last element in an object or an array.

3. Valid JSON requires "quotes" around the "names" (AKA "keys") in {"name":value} pairs.  
 A common source of errors is forgetting to use these quotes - e.g. {name:value} is wrong, but

{"name":value} is correct.  
  
4. Incorrectly nested {curly braces} or [square brackets]. Indenting properly can help a lot with this.

5. Splitting "quoted values" between two different lines.  
  
6. Technically, the data in JSON may include "non-ASCII" characters such as the  
 British pound sign, "smart quotes", "curly apostrophes", "long dashes" or

letters from languages other than English.  
  
 However, using non-ASCII characters in JSON may introduce complications for  
 some software that processes JSON. Therefore, do not create JSON files in word processors  
 such as Microsoft Word or Mac's "Pages" program. "Word processor" programs are intended for

writing prose (not for programming) and often use non-ASCII characters such as "smart quotes",

"curly apostrophes" and long dashes instead of the ASCII standard versions of these characters.

You should only edit JSON files with a "text editor" (e.g. Notepad, VSCode, etc.)

Note that you can use the Unicode escape sequences in JSON files.

For example, you can type \u00a3 instead of the British pound sign. The raw JSON file

would still only contain ASCII characters and Unicode-aware software would be able to

display \u00a3 as the British pound sign.