**An Overview of Various Ways to Work With Your JSON Data in Python**

**Python provides a host of different methods to work with this common data storage format.**

If you work in tech — especially as a software engineer or data scientist — you’ve probably heard the term JSON thrown around fairly often. In fact, I’d bet that you’ve had to work with it yourself at one point or another.

**What is JSON Data?**

JSON stands for JavaScript Object Notation. Technically, it was derived from a subset of JavaScript focused on arrays and literals by a programmer named Douglas Crockford (who actually still works on the development of JavaScript) [1].

However, the name can be a little misleading. Although its origins lie in JavaScript, JSON is a language-independent entity. It is essentially just a convenient way of specifying and transporting data. It’s also human-readable, especially when compared to alternative formats such as XML.

Now then, enough introduction. If you’re interested in more historical details, feel free to check out the official website: [json.org](http://www.json.org).

In this article, we’ll talk about JSON specifically in the realm of Python. JSON is everywhere — as a result, it’s highly likely you’ll have to deal with it at some point, be it as a software engineer doing server-side development or as a data scientist attempting to read information into a table.

Before we get into different ways to work with JSON in Python, we need to know how it actually looks. At its core, JSON data is just a big array of key-value pairs. If you’re not familiar with that terminology, it effectively just means that data is organized by giving individual values a reference name, or *key*.

This is easier to see via example. For instance, say that we want to store information about student grades on a final exam. Our JSON object might look like the following (poor Kaylee):

{  
 "John": 92,  
 "Kaylee": 44,  
 "Akshay": 78,  
 "Zahra": 100  
}

This structure can be more complex and nested (it is possible to makes the values arrays or even their own objects with further key-value pairs), but this is the basic idea. Once you grasp it, the rest is just a matter of application and some mental dedication.

**The key question here is this: does this look familiar at all?** If you’re familiar with data structures in Python, your brain’s neural pathways should be going crazy right now.

This is basically just a [Python dictionary](https://medium.com/towards-data-science/whats-in-a-dictionary-87f9b139cc03) [2], with **one important caveat: strings in JSON must be contained in double quotes, whereas Python dictionaries allow single or double quotes.**

**Method 1: With Pandas**

The fact that JSON is effectively structured identically to Python dictionaries is a wonderful gift to Pandas programmers such as ourselves. Recall one of the easiest ways to make a DataFrame in Pandas, directly from a dictionary:

my\_dict = {  
 "Name": ["Alisha", "Ariel", "Aditi", "Tatiana", "Juan"],  
 "Degree": ["Nursing", "Biomedical Engineering", "History", "Psychology", "Mathematics"]  
 }  
df = pd.DataFrame(my\_dict)  
df

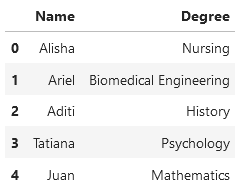


Image By Author

JSON, as we saw above, is structured practically identically to a dictionary. Thus, it’s unsurprising that Pandas comes with a utility method that makes reading JSON files into DataFrames incredibly simple.

If you’re used to working with CSV files, then you may have run something like the following before:

my\_df = pd.DataFrame('path/to/data.csv')

Working with JSON follows the same pattern. If we have the data above saved as a JSON file, we can read it into a DataFrame as follows:

my\_json\_df = pd.DataFrame('path/to/data.json')  
my\_json\_df

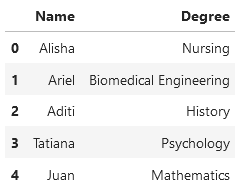


Image By Author

And that’s that. Remember that JSON data is formatted identically to Python dictionaries, as a collection of key-value pairs. As a result, this DataFrame is identical to the example we showed previously using native Python.

**Method 2: Native Python**

If you’re unfamiliar with Pandas, but still need a way to work with JSON, fear not. Python has a wonderful library that will aid in this task: json (clever, I know).

There are two main functions you will need from this library: load and dump.

The load function lets you read in data from a JSON file. However, rather than converting it into a dictionary like Pandas’s read\_json function, it will give the data to you as a Python dictionary. Before calling the load function on the file, we also need to open it using Python’s built-in open function:

>>> import json  
>>> my\_file = open('path/to/example.json', 'r') # 'r' for read mode  
>>> data\_dict = json.load(my\_file)  
>>> my\_file.close() # Don't forget to close the file  
>>> data\_dict  
{  
 "Name": ["Alisha", "Ariel", "Aditi", "Tatiana", "Juan"],  
 "Degree": ["Nursing", "Biomedical Engineering", "History", "Psychology", "Mathematics"]  
}

The dump function does the exact opposite: it lets you define a dictionary in Python, and then write it to a JSON file. Be aware that if you open the file in write mode, signified by a “w,” you will overwrite any existing content. If you want to add to the file, use append mode (signified by “a”).

>>> data\_dict = {  
 "Name": ["Alisha", "Ariel", "Aditi", "Tatiana", "Juan"],  
 "Degree": ["Nursing", "Biomedical Engineering", "History", "Psychology", "Mathematics"]  
 }  
  
>>> my\_file = open('path/to/example.json', 'w') # 'w' for write mode  
>>> json.dump(data\_dict, my\_file)  
>>> my\_file.close() # Don't forget to close the file

There are two related functions worth mentioning here: loads and dumps. Rather confusingly, these are meant to be read as “load-s” and “dump-s.” The “s” at the end stands for “string.”

These functions work similarly to their counterparts, except they aren’t configured to work with files. Many times when programming, we receive data from some server directly in the form of a JSON string, without needing a file to act as the middle man. To read this data into a Python dictionary, we can use the loads function. Alternatively, to convert a dictionary into a JSON string that we need to send to a server, we can use dumps.

**Recap + Final Thoughts**

Here’s a mini JSON cheat-sheet for you:

1. In Pandas, use the read\_json function.
2. If you’re using the JSON module, use load and dump if you’re working with JSON files, and loads and dumps if you’re working directly with JSON strings.

And with that, you should be ready to deal with JSON out in the wild. Until next time, and happy holidays!