Reading and Writing JSON to a File in Python

By Scott Robinson (https://twitter.com/ScottWRobinson) •
10 Comments (/reading-and-writing-json-to-a-file-in-python/#disqus_thread)

Introduction

In this article, we'll be parsing, reading and writing JSON data to a file in Python.

Over the last 5-10 years, the JSON (https://en.wikipedia.org/wiki/JSON) format has been one of, if not the most, popular ways to serialize data. Especially in the web development world, you'll likely encounter JSON through one of the many REST APIs (https://en.wikipedia.org/wiki/Representational_state_transfer), application configuration, or even simple data storage.

Given its prevalence and impact on programming, at some point in your development you'll likely want to learn how to read JSON from a file or write JSON to a file. Both of these tasks are pretty easy to accomplish with Python, as you'll see in the next few sections.

Writing JSON to a File

The easiest way to write your data in the JSON format to a file using Python is to use store your data in a `dict` object, which can contain other nested `dict`s, arrays, booleans, or other primitive types like integers and strings. You can find a more detailed list of data types supported here (https://en.wikipedia.org/wiki/JSON#Data_types.2C_syntax_and_example).

The built-in json (https://docs.python.org/3.4/library/json.html) package has the magic code that transforms your Python `dict` object in to the serialized JSON string.

```
import json

data = {}
data['people'] = []
data['people'].append({
    'name': 'Scott',
    'website': 'stackabuse.com',
    'from': 'Nebraska'
})
data['people'].append({
    'name': 'Larry',
    'website': 'google.com',
    'from': 'Michigan'
})
data['people'].append({
    'name': 'Tim',
    'website': 'apple.com',
    'from': 'Alabama'
})

with open('data.txt', 'w') as outfile:
    json.dump(data, outfile)
```

After importing the `json` library, we construct some simple data to write to our file. The important part comes at the end when we use the `with` statement to open our destination file, then use json.dump (https://docs.python.org/3.4/library/json.html#json.dump) to write the `data` object to the `outfile` file.

Any file-like object can be passed to the second argument, even if it isn't an actual file. A good example of this would be a socket, which can be opened, closed, and written to much like a file. With JSON being popular throughout the web, this is another use-case you may encounter.

A slight variation on the `json.dump` method that's worth mentioning is json.dumps (https://docs.python.org/3.4/library/json.html#json.dumps), which returns the actual JSON string instead of sending it directly to a writable object. This can give you some more control if you need to make some changes to the JSON string (like encrypting it, for example).

Reading JSON from a File

On the other end, reading JSON data from a file is just as easy as writing it to a file. Using the same `json` package again, we can extract and parse the JSON string directly from a file object. In the following example, we do just that and then print out the data we got:

```
import json

with open('data.txt') as json_file:
    data = json.load(json_file)
    for p in data['people']:
        print('Name: ' + p['name'])
        print('Website: ' + p['website'])
        print('From: ' + p['from'])
        print('')
```

json.load (https://docs.python.org/3.4/library/json.html#json.load) is the important method to note here. It reads the string from the file, parses the JSON data, populates a Python `dict` with the data and returns it back to you.

Just like `json.dump`, `json.load` has an alternative method that lets you deal with strings directly since many times you probably won't have a file-like object that contains your JSON. As you probably guessed, this method is json.loads (https://docs.python.org/3.4/library/json.html#json.loads). Consider the case where you're calling a REST GET endpoint that returns JSON. This data comes to you as a string, which you can then pass to `json.loads` directly instead.

Options

When serializing your data to JSON with Python, the result will be in the standard format and not very readable since whitespace is eliminated. While this is the ideal behavior for most cases, sometimes you may need to make small changes, like adding whitespace to make it human readable. Both `json.dump` and `json.load` provide quite a few options for more flexibility, a few of which will be described here.

Pretty-Printing

Making JSON human readable (aka "pretty printing") is as easy as passing an integer value for the `indent` parameter:

```
>>> import json
>>> data = {'people':[{'name': 'Scott', 'website': 'stackabuse.com', 'from': 'Nebrask
a'}]}
>>> json.dumps(data, indent=4)
{
    "people": [
        {
            "website": "stackabuse.com",
            "from": "Nebraska",
            "name": "Scott"
        }
    ]
}
```

This is actually quite useful since you'll often have to read JSON data during development. Another option is to use the command line tool, `json.tool`. So if you just want to pretty-print JSON to the command line you can do something like this:

```
$ echo '{"people":[{"name":"Scott", "website":"stackabuse.com", "from":"Nebraska"}]}'
| python -m json.tool
{
    "people": [
        {
            "name": "Scott",
            "website": "stackabuse.com"
            "from": "Nebraska",
        }
    ]
}
```

Sorting

In JSON, an object is defined as:

> *An object is an unordered set of name/value pairs.*

So the standard is saying that key order isn't guaranteed, but it's possible that you may need it for your own purposes internally. To achieve ordering, you can pass `True` to the `sort_keys` option when using `json.dump` or `json.dumps`.

```
>>> import json
>>> data = {'people':[{'name': 'Scott', 'website': 'stackabuse.com', 'from': 'Nebrask
a'}]}
>>> json.dumps(data, sort_keys=True, indent=4)
{
    "people": [
        {
            "from": "Nebraska",
            "name": "Scott",
            "website": "stackabuse.com"
        }
    ]
}
```

ASCII Text

By default, `json.dump` will ensure that all of your text in the given Python dictionary are ASCII-encoded. If non-ASCII characters are present, then they're automatically escaped, as shown in the following example:

```
>>> import json
>>> data = {'item': 'Beer', 'cost':'£4.00'}
>>> jstr = json.dumps(data, indent=4)
>>> print(jstr)
{
    "item": "Beer",
    "cost": "\u00a34.00"
}
```

This isn't always acceptable, and in many cases you may want to keep your Unicode characters un-touched. To do this, set the `ensure_ascii` option to `False`.

```
>>> jstr = json.dumps(data, ensure_ascii=False, indent=4)
>>> print(jstr)
{
    "item": "Beer",
    "cost": "£4.00"
}
```

Conclusion

In this article we introduced you to the `json.dump`, `json.dumps`, `json.load`, and `json.loads` methods, which help in serializing and deserializing JSON strings.

With JSON having become one of the most popular ways to serialize structured data, you'll likely have to interact with it pretty frequently, especially when working on web applications. Python's `json` module is a great way to get started, although you'll probably find that simplejson (https://github.com/simplejson/simplejson) is another great alternative that is much less strict on JSON syntax (which we'll save for another article).

What are some of your common use-cases for storing JSON data? Data persistence, configuration, or something else? Let us know in the comments!

---

json (/tag/json/), python (/tag/python/), how to (/tag/how-to/)

6%20JSON%20to%20a%20File%20in%20Python&url=https://stackabuse.com/reading-
/)
narer/sharer.php?u=https://stackabuse.com/reading-and-writing-json-to-a-file-in-

ireArticle?mini=true%26url=https://stackabuse.com/reading-and-writing-json-to-a-file-
:abuse.com)

(/author/scott/)

About Scott Robinson (/author/scott/)

Twitter (https://twitter.com/ScottWRobinson)

Subscribe

‹ Previous Post (/copying-a-directory-with-scp/)

Next Post › (/understanding-json-web-tokens-jwt/)

Ad

Follow Us

(https://twitter.com/StackAbuse) (https://www.facebook.com/stackabuse) (https://stackabuse.com/rss/)

Newsletter

Subscribe to our newsletter! Get occassional tutorials, guides, and reviews in your inbox.

Enter your email...

Subscribe

No spam ever. Unsubscribe at any time.

Ad

Want a remote job?

More jobs (https://hireremote.io)

Jobs via HireRemote.io (https://hireremote.io)

Prepping for an interview?

(https://stackabu.se/daily-coding-problem)

- Improve your skills by solving one coding problem every day
- Get the solutions the next morning via email
- Practice on **actual problems** asked by top companies, like:

Google facebook amazon.com Microsoft

Daily Coding Problem (https://stackabu.se/daily-coding-problem)

Ad

Recent Posts

A Guide to JPA with Hibernate - Relationship Mapping (/a-guide-to-jpa-with-hibernate-relationship-mapping/)

Insertion Sort in JavaScript (/insertion-sort-in-javascript/)

The Factory Method Design Pattern in Python (/the-factory-method-design-pattern-in-python/)

Tags

algorithms (/tag/algorithms/) amqp (/tag/amqp/) angular (/tag/angular/)
announcements (/tag/announcements/) apache (/tag/apache/) api (/tag/api/)
arduino (/tag/arduino/) artificial intelligence (/tag/artificial-intelligence/)
asynchronous (/tag/asynchronous/) aws (/tag/aws/)

Follow Us

(https://twitter.com/StackAbuse) (https://www.facebook.com/stackabuse) (https://stackabuse.com/rss/)