Theory: Json module

19 minutes 0 / 5 problems solved

As you know, JSON is a very common format for storing text-based data. Even though it originally derived from JavaScript, nowadays this format is language-independent and is used in all kinds of situations. Naturally, programming languages have their ways of dealing with JSON, and in this topic, we will see how it is done in Python.

§1. json module

Python has a built-in module for working with the JSON format: `json`. If we want to use it, we just need to import it at the beginning of the program.

What does it allow us to do? Well, the two main procedures are converting Python data into JSON and the other way around. To better understand the logic behind the conversion, let's take a look at a JSON object:

```
{
  "movies": [
    {
      "title": "Inception",
      "director": "Christopher Nolan",
      "year": 2010
    },
    {
      "title": "The Lord of the Rings: The Fellowship of the Ring",

      "director": "Peter Jackson",

      "year": 2001

    },

    {

      "title": "Parasite",

      "director": "Bong Joon Ho",

      "year": 2019

    }

  ]

}
```

You can see that there are a lot of similarities between JSON notation and Python data types: we have strings and numbers, a JSON object looks similar to a Python dictionary, an array — to list. This makes conversions between JSON and Python quite easy and intuitive. Here's a full conversion table for encoding Python data to JSON:

Python	JSON
`dict`	object
`list`, `tuple`	array
`str`	string
`int`, `float`	number
`True`	true
`False`	false

Current topic:
Json module Stage 2

Topic depends on:
JSON Stage 2
Load module Stage 1 5★
Context manager Stage 2

Table of contents:
↑ Json module
§1. json module
§2. Encoding to JSON
§3. Decoding JSON
§4. Summary
Feedback & Comments

Python	JSON
`None`	null

Conversion table

Now let's take a look at specific methods available in the `json` module and see how the conversion happens.

§2. Encoding to JSON

Generally, encoding to JSON format is called **serialization**. The `json` module has two methods for serializing: `json.dump()` and `json.dumps()`. The key difference between these two methods is the type we're serializing to: `json.dump()` creates a file-like object, and `json.dumps()` creates a string.

Suppose, we have a dictionary equivalent to the JSON we've seen earlier.

```
# Python dictionary
movie_dict = {
  "movies": [
    {
      "title": "Inception",
      "director": "Christopher Nolan",
      "year": 2010
    },
    {

      "title": "The Lord of the Rings: The Fellowship of the Ring",

      "director": "Peter Jackson",

      "year": 2001

    },

    {

      "title": "Parasite",

      "director": "Bong Joon Ho",

      "year": 2019

    }

  ]

}
```

Here's how we can save it to the JSON file *movies.json:*

```
import json


with open("movies.json", "w") as json_file:
    json.dump(movie_dict, json_file)
```

As you can see, this method has two required arguments: the data and the file-like object that you can write to. If you run this code, you'll create a JSON file with the data about movies.

Another option is serializing the data into a string using `json.dumps()`. In this case, the only required argument is the data we want to serialize:

```
json_str = json.dumps(movie_dict)

print(json_str)

# {"movies": [{"title": "Inception", "director": "Christopher Nolan", "year": 2010
}, {"title": "The  Lord of the Rings: The Fellowship of the Ring", "director": "Pe
ter Jackson", "year": 2001}, {"title": "Parasite", "director": "Bong Joon Ho", "ye
ar": 2019}]}
```

> Careful with data types! JSON only supports strings as keys. Basic Python types like integers will get converted to strings automatically but for other types of keys, like tuple, you'll get a `TypeError` because the `.dump()` and `.dumps()` functions cannot convert it to a string.

In addition to the required parameters, both methods have several optional ones. You can check them all out in the official documentation, here we'll only look at the `indent` parameter. You can see that the string we got in the example above is quite hard to read, compared to the original dictionary. Well, if we specify `indent` (an integer or a string), we can pretty-print our resulting JSON:

```
json_str = json.dumps(movie_dict, indent=4)
print(json_str)
```

And the resulting string:

```
# json_str
'{
    "movies": [
        {
            "title": "Inception",
            "director": "Christopher Nolan",
            "year": 2010
        },
        {

            "title": "The Lord of the Rings: The Fellowship of the Ring",

            "director": "Peter Jackson",

            "year": 2001

        },

        {

            "title": "Parasite",

            "director": "Bong Joon Ho",

            "year": 2019

        }

    ]

}'
```

§3. Decoding JSON

The opposite procedure is **deserialization**. Similarly to serialization, the `json` module has two methods: `json.load()` and `json.loads()`. Here the difference is in the input JSONs: file-like objects or strings.

Let's convert the JSONs we've just created back to Python data types.

```
# from a file
with open("movies.json", "r") as json_file:
    movie_dict_from_json = json.load(json_file)

print(movie_dict_from_json == movie_dict)  # True
```

You can see that the dictionary that we got as a result of `json.load()` equals our original dictionary. The same with `json.loads()`:

```
# from string
print(movie_dict == json.loads(json_str))  # True
```

> Remember the issue with non-string keys? Well, if we convert a Python dictionary with non-string keys to JSON and then back to Python

object, we will not get the same dictionary.

§4. Summary

In this topic, we've seen how to work with JSON using the built-in Python module `json`. We can

- convert Python objects to JSON using either `json.dump()` or `json.dumps()`;
- convert JSON to Python objects using either `json.load()` or `json.loads()`.

The conversions are done according to the conversion table and not every Python object can be converted to JSON.

Considering that JSON is a very commonly used data format, it's important to be able to work with it!

Report a typo

128 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion