Web Applications and Services

- **Reading:** Module 2 Introduction
  10 min
- **Reading:** Web Applications and Services
  10 min
- **Reading:** Data Serialization
  10 min
- **Reading:** Data Serialization Formats
  10 min
- **Reading:** More About JSON
  10 min

Python Requests

Module 2 Project

More About JSON

Alright, we've seen a couple of different serialization formats. Let's now dive into more details about ***JSON (JavaScript Object Notation)***, which you'll be using in the lab at the end of this module.

As we mentioned before, JSON is ***human-readable***, which means it's encoded using printable characters, and formatted in a way that a human can understand. This doesn't necessarily mean that you *will* understand it when you look at it, but you *can*.

Lots of web services send messages back and forth using JSON. In this module, and in future ones, you'll serialize JSON messages to send to a web service.

JSON supports a few ***elements*** of different data types. These are very basic data types; they represent the most common basic data types supported by any programming language that you might use.

JSON has ***strings,*** which are enclosed in quotes.

```
"Sabrina Green"
```

It also has ***numbers,*** which are not.

```
1002
```

JSON has ***objects***, which are key-value pair structures like Python dictionaries.

```
{
  "name": "Sabrina Green",
  "username": "sgreen",
  "uid": 1002
}

```

And a key-value pair can contain another object as a value.

```
{
  "name": "Sabrina Green",
  "username": "sgreen",
  "uid": 1002,
  "phone": {
    "office": "802-867-5309",
    "cell": "802-867-5310"
  }
}

```

JSON has ***arrays***, which are equivalent to Python lists. Arrays can contain strings, numbers, objects, or other arrays.

```
[
  "apple",
  "banana",
  12345,
  67890,
  {
    "name": "Sabrina Green",
    "username": "sgreen",
    "phone": {
      "office": "802-867-5309",
      "cell": "802-867-5310"
    },
    "department": "IT Infrastructure",
    "role": "Systems Administrator"
  }
]

```

And as you've probably noticed, JSON elements are always ***comma-delimited***. With these basics under your belt, you could create valid JSON by hand, and edit examples of JSON that you encounter. Except we don't really want to do that, since it's clunky and we're bound to make a ton of errors! Instead, let's use the **json** library that does all the heavy lifting for us.

```
import json
```

The **json** library will help us turn Python objects into JSON, and turn JSON strings into Python objects! The **dump()** method serializes basic Python objects, writing them to a file. Like in this example:

```
import json

people = [
  {
    "name": "Sabrina Green",
    "username": "sgreen",
    "phone": {
      "office": "802-867-5309",
      "cell": "802-867-5310"
    },
    "department": "IT Infrastructure",
    "role": "Systems Administrator"
  },
  {
    "name": "Eli Jones",
    "username": "ejones",
    "phone": {
      "office": "684-348-1127"
    },
    "department": "IT Infrastructure",
    "role": "IT Specialist"
  }
]

with open('people.json', 'w') as people_json:
    json.dump(people, people_json)
```

That gives us a file with a single line that looks like this:

```
[{"name": "Sabrina Green", "username": "sgreen", "phone": {"office": "802-867-5309", "cell": "802-867-5310"}, "department": "IT Infrastructure", "role": "Systems Administrator"}, {"name": "Eli Jones", "username": "ejones", "phone": {"office": "684-348-1127"}, "d
```

JSON doesn't *need* to contain multiple lines, but it sure can be hard to read the result if it's formatted this way! Let's use the **indent** parameter for **json.dump()** to make it a bit easier to read.

```
with open('people.json', 'w') as people_json:
    json.dump(people, people_json, indent=2)
```

The resulting file should look like this:

```
[
  {
    "name": "Sabrina Green",
    "username": "sgreen",
    "phone": {
      "office": "802-867-5309",
      "cell": "802-867-5310"
    },
    "department": "IT Infrastructure",
    "role": "Systems Administrator"
  },
  {
    "name": "Eli Jones",
    "username": "ejones",
    "phone": {
      "office": "684-348-1127"
    },
    "department": "IT Infrastructure",
    "role": "IT Specialist"
  }
]

```

Now it's much easier to follow! In fact, it looks very similar to how you'd write out the object in Python. Cool!

Another option is to use the **dumps()** method, which also serializes Python objects, but returns a string instead of writing directly to a file.

```
>>> import json
>>>
>>> people = [
...   {
...     "name": "Sabrina Green",
...     "username": "sgreen",
...     "phone": {
...       "office": "802-867-5309",
...       "cell": "802-867-5310"
...     },
...     "department": "IT Infrastructure",
...     "role": "Systems Administrator"
...   },
...   {
...     "name": "Eli Jones",
...     "username": "ejones",
...     "phone": {
...       "office": "684-348-1127"
...     },
...     "department": "IT Infrastructure",
...     "role": "IT Specialist"
...   }
... ]
>>> people_json = json.dumps(people)
>>> print(people_json)
[{"name": "Sabrina Green", "username": "sgreen", "phone": {"office": "802-867-5309", "cell": "802-867-5310"}, "department": "IT Infrastructure", "role": "Systems Administrator"}, {"name": "Eli Jones", "username": "ejones", "phone": {"office": "684-348-1127"}, "d
```

The **load()** method does the inverse of the **dump()** method. It deserializes JSON from a file into basic Python objects. The **loads()** method also deserializes JSON into basic Python objects, but parses a string instead of a file.

```
>>> import json
>>> with open('people.json', 'r') as people_json:
...     people = json.load(people_json)
...
>>> print(people)
[{'name': 'Sabrina Green', 'username': 'sgreen', 'phone': {'office': '802-867-5309', 'cell': '802-867-5310'}, 'department': 'IT Infrastructure', 'role': 'Systems Administrator'}, {'name': 'Eli Jones', 'username': 'ejones', 'phone': {'office': '684-348-1127'}, 'd
```

Remember that JSON elements can only represent simple data types. If you have complex Python objects, you won't be able to automatically serialize them as JSON. Take a look at this table to see in detail how Python objects are converted into JSON elements.

Completed **Go to next item**

Like **Dislike** **Report an issue**