Data Serialization

If you have two programs that need to communicate with each other, how do you get that data from one place to another? We're going to talk about two aspects of that problem: what to send, and how to send it.

First, what do you send? When you have a conversation with another person, you don't send thoughts and memories directly between your brains. At least not yet! You first have to convert your thoughts into language, and then transmit that language to another person. They take that language, and convert it back into thoughts. It's the same with programs running in different places, or at different times.

In a previous course, we took a list of lists in memory and wrote it to disk as a ***Comma-Separated Value (CSV)*** file. This is one example of a technique called ***data serialization***. Data serialization is the process of taking an in-memory data structure, like a Python object, and turning it into something that can be stored on disk or transmitted across a network. Later, the file can be read, or the network transmission can be received by another program and turned back into an object again. Turning the serialized object back into an in-memory object is called ***deserialization***.

Data serialization is extremely useful for communicating with web services. A web service's ***API endpoint*** takes messages in a specific format, containing specific data. By the end of this module, we'll be sending messages to web services, but for now let's concentrate on how to serialize Python objects into some common formats.

Let's start with the contact information from one of our CSV examples. We'll keep just two entries to keep our examples short, but there's no limit to how long these can be.

```
1   name,username,phone,department,role
2   Sabrina Green,sgreen,802-867-5309,IT Infrastructure,System Administrator
3   Eli Jones,ejones,684-3481127,IT Infrastructure,IT specialist
```

Instead of having a list of lists, we could turn this information into a list of dictionaries. In each of these dictionaries, the key will be the name of the column, and the value will be the corresponding information in each row. It could look something like this:

```
1   people = [
2       {
3           "name": "Sabrina Green",
4           "username": "sgreen",
5           "phone": "802-867-5309",
6           "department": "IT Infrastructure",
7           "role": "Systems Administrator"
```

```
            "role": "Systems Administrator"
        },
        {
            "name": "Eli Jones",
            "username": "ejones",
            "phone": "684-348-1127",
            "department": "IT Infrastructure",
            "role": "IT Specialist"
        },
    ]

```

Using a structure like this lets us do interesting things with our information that's much harder to do with CSV files. For example, let's say we want to record more than one phone number for each person. Instead of using a single string for "phone", we could represent that data in another dictionary, like this:

```
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"
    },
]

```

Now, we can record multiple phone numbers per person, and give them descriptive names like "office" and "cell". This would be hard to store in a CSV file, because the data is not *flat*. To help us with that, there's a bunch of different formats that we can use to store our data when the structure isn't flat.

Up next, we'll look into a few of the most common ones.

✓ **Completed**