[API with JSON](https://www.linkedin.com/learning/learn-api-documentation-with-json-and-xml/why-document-json-and-xml?leis=OCN&resume=false&u=246983906)

**Why document JSON and XML?**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's talk about why you would document JSON and XML, and how it relates to API documentation. API stands for application programming interface. It defines how two pieces of software communicate with each other. There are several types of APIs, but for this course, I'm going to focus on web APIs. I'll give you an example of a web API. Let's say you're using a Facebook app on your smartphone. When you open the app, the first thing it does, is retrieve your friends' statuses. To do this, it sends an API request to the Facebook server, with data that identifies you, and says that you want to retrieve the most recent statuses. Then the Facebook server responds, and returns the data, with all of the statuses to display. This is different than when you go to the Facebook webpage on a browser, the server is only returning the data, not a full webpage. If you are documenting a web API, what do you need to cover? Any time a developer builds something that uses an app, they need to know what the request will look like, so that the server can understand it, and what the response will look like, so that they can make sense of the data. The first web API was created by eBay. If you think about using the eBay website, it works great for individuals, but suppose you're a company like a thrift store, and you have software that keeps track of your inventory. You have to manually move information back and forth between two places, the eBay website, and the accounting software. Wouldn't it be great if stores could use accounting software that put things up for bid on eBay, and kept a record of the final sale? eBay came up with this idea in 2005, now more than half of the items sold on eBay are through the API, and not the website. What makes APIs so powerful? There are several features I could mention, but let's just start with mashups. APIs let developers create apps that bring together data from different sources. For example, let's say you wanted to create an app that shows how to get to the nearest dog park. How would you use APIs to build this? Several cities now provide APIs that list where all the dog parks are. Google Maps provides an API that says how to get to a given location. Your app first calls the city API, then, once it has a location, it calls the Google API, and gets the directions. So with APIs, apps can bring data together in new ways that no-one has thought of before. There's a website called programmableweb.com, it used to keep track of all of the web APIs that are publicly available to use. Starting with eBay in 2005, there's been extreme growth up to 2013, where there are over 10,000 APIs, and these are only the publicly available ones, there are also ones just for company partners or internal use. I've talked to people in the industry, who believe that these numbers represent less than 1/10 of the APIs out there. And guess what? All of those APIs need documentation. Why do we need good API documentation? It's the same reason as why we need good end user documentation. Software development and writing are two very different skills, most software developers hate to write, and most are not very good at it. API documentation is somewhat different than end user documentation. You are writing for a software developer audience, and they learn and understand differently than the general population. Also, the documentation is almost entirely text. The most widely used text format for web APIs are JSON and XML. They're used to describe structured data, which is data that doesn't necessarily fit neatly into a table. If you think about a database table, or an Excel spreadsheet, you have two dimensions of rows and columns, where the numbers of cells are the same for each row, and each column. Structured data is more like a tree, where data can branch off from other data any number of times. Of the two formats, JSON is simpler, and uses less text for the same amount of data, but XML came first, and is still heavily used in the industry.

**Data types and structured data**

[Presenter] Let's take a closer look at data types and structured data. As an API writer, one of the main tasks you'll need to do is describing data, so this lesson will explain how data is defined, especially for use in web APIs. When you are developing software, each piece of data has a type. Common types include integers, which are whole numbers, decimals, which can have digits to the right of the decimal place, text, true/false values, also known as Boolean, and so on. You can also define your own custom types, like a type that captures all the information about a user, or a device, or a bank account, and so on. This course focuses on two formats: JSON and XML. I'll go into more detail about them later, but for now, I'll mention that JSON uses three basic types: numbers, strings, and Booleans, and XML only has one type, which is strings. Let me explain what all of this means. The word "string" is a technical term meaning "text." It's essentially a group of text characters that are strung together. Usually, strings are enclosed in quotation marks, either with double or single quotes. They can have letters, numbers, punctuation, math symbols, and so on. They can even have spaces. Here are some examples. A single name, a code, and a sentence. Booleans may sound extremely technical, but they are, in fact, very simple. Long ago, a mathematician named George Boole came up with a radical idea. If we can do math with numbers, why can't we do math with logic? He came up with a number system with only two values: true and false. With just these two values, you can do all sorts of things, just like you do with numbers. For documentation purposes, all you need to know is that a Boolean value can be either true or false. Let's do a little exercise here. This is a screenshot from a user profile. Once you make changes and submit it, it will send data back to the server. Of course, the data will have data types. Let's see if we can find all three of the data types that I've talked about. Well, string is easy. The current city and hometown are both strings. Numbers are easy, too. The day and the year of the birthday are clearly numbers. What about Booleans? Well, any checkbox is a Boolean. It has a value of true if it's checked, and a value of false if it's not. In addition to individual data values, you can have groups of data called collections. There are two common types. Arrays are lists of data values, and dictionaries are what are called lookup tables. Let me explain each of these in more detail. Arrays, as I mentioned before, are a list of data values. That means they have a size, which is how many are in the list, and they have an order, which describes which data values come first, second, and then third, and so on. For example, you could have a list of numbers, or you could have a list of strings. You can even have lists of lists. Dictionaries are a little more complicated. They're a list of data keys and values. The idea is you use the key to look up the value. Think about how you use a dictionary. Each word in the dictionary has a definition. If your dictionary is electronic, then you give it the word, and it returns the definition. The word is the key, and the definition is the value. So, for example, let's say that you had data that divided the United States into several regions, and you wanted to have a dictionary where you could give it the region name, and it would return the number of U.S. states in that region. Your dictionary might look like this. In front of the colon is the key, a string in quotes, and after the colon is the value. Structured data combines data types in collections. Collections can consist of other collections. In other words, you can have dictionaries of lists, a list of dictionaries, a dictionary of dictionaries, and so on. As many levels as you need. What you end up with is a tree structure where each collection is a branch and the end of each branch is a data value. You may be wondering what this has to do with APIs. The answer is very simple. As I mentioned in the introductory lesson, APIs have requests and responses. The responses almost always returned information from the server in the form of structured data, and sometimes the requests use structured data to pass information to the server.