A 5-Minute Crash Course in XML

If you already know about xml, you can skip this section.

XML is a generalized way of describing hierarchical structured data. An XML *document* contains one or more *elements*, which are delimited by *start and end tags*. This is a complete (albeit boring) xml document:

```
<foo>   <!-- ① -->
</foo>  <!-- ② -->
```

① This is the *start tag* of the `foo` element.

② This is the matching end tag of the `foo` element. Like balancing parentheses in writing or mathematics or code, every start tag must be *closed* (matched) by a corresponding end tag.

Elements can be *nested* to any depth. An element `bar` inside an element `foo` is said to be a subelement or *child* of `foo`.

```
<foo>
  <bar></bar>
</foo>
```

The first element in every xml document is called the *root element*. An xml document can only have one root element. The following is **not an xml document**, because it has two root elements:

```
<foo></foo>
<bar></bar>
```

Elements can have *attributes*, which are name-value pairs. Attributes are listed within the start tag of an element and separated by whitespace. *Attribute* names can not be repeated within an element. *Attribute values* must

be quoted. You may use either single or double quotes.

```
<foo lang='en'>                          <!-- ① -->
  <bar id='papayawhip' lang="fr"></bar>  <!-- ② -->
</foo>
```

① The `foo` element has one attribute, named `lang`. The value of its `lang` attribute is `en`.

② The `bar` element has two attributes, named `id` and `lang`. The value of its `lang` attribute is `fr`. This doesn't conflict with the `foo` element in any way. Each element has its own set of attributes.

If an element has more than one attribute, the ordering of the attributes is not significant. An element's attributes form an unordered set of keys and values, like a Python dictionary. There is no limit to the number of attributes you can define on each element.

Elements can have *text content*.

```
<foo lang='en'>
  <bar lang='fr'>PapayaWhip</bar>
</foo>
```

Elements that contain no text and no children are *empty*.

```
<foo></foo>
```

There is a shorthand for writing empty elements. By putting a / character in the start tag, you can skip the end tag altogther. The xml document in the previous example could be written like this instead:

```
<foo/>
```

Like Python functions can be declared in different *modules*, xml elements can be declared in different *namespaces*. Namespaces usually look like URLs. You use an `xmlns` declaration to define a *default namespace*. A namespace declaration looks similar to an attribute, but it has a different purpose.

```
<feed xmlns='http://www.w3.org/2005/Atom'>  <!-- ① -->
  <title>dive into mark</title>             <!-- ② -->
</feed>
```

① The `feed` element is in the `http://www.w3.org/2005/Atom` namespace.

② The `title` element is also in the `http://www.w3.org/2005/Atom` namespace. The namespace declaration affects the element where it's declared, plus all child elements.

You can also use an `xmlns:prefix` declaration to define a namespace and associate it with a *prefix*. Then each element in that namespace must be explicitly declared with the prefix.

```
<atom:feed xmlns:atom='http://www.w3.org/2005/Atom'>  <!-- ① -->
  <atom:title>dive into mark</atom:title>             <!-- ② -->
</atom:feed>
```

① The `feed` element is in the `http://www.w3.org/2005/Atom` namespace.

② The `title` element is also in the `http://www.w3.org/2005/Atom` namespace.

As far as an XML parser is concerned, the previous two xml documents are *identical*. Namespace + element name = XML identity. Prefixes only exist to refer to namespaces, so the actual prefix name (atom:) is irrelevant. The namespaces match, the element names match, the attributes (or lack of attributes) match, and each element's text content matches, therefore the XML documents are the same.

Finally, XML documents can contain character encoding information on the first line, before the root element. (If you're curious how a document can contain information which needs to be known before the document can be parsed, Section F of the XML specification details how to resolve this Catch-22.)

```
<?xml version='1.0' encoding='utf-8'?>
```

And now you know just enough XML to be dangerous!