Searching For Nodes Within An XML Document

So far, we've worked with this XML document "from the top down," starting with the root element, getting its child elements, and so on throughout the document. But many uses of XML require you to find specific elements. Etree can do that, too.

```
import xml.etree.ElementTree as etree
tree = etree.parse('feed.xml')
root = tree.getroot()

print (root.findall('{http://www.w3.org/2005/Atom}entry'))    #①
#[<Element {http://www.w3.org/2005/Atom}entry at e2b4e0>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b510>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b540>]

print (root.tag)
#'{http://www.w3.org/2005/Atom}feed'

print (root.findall('{http://www.w3.org/2005/Atom}feed'))     #②
#[]

print (root.findall('{http://www.w3.org/2005/Atom}author'))   #③
#[]
```

① The `findall()` method finds child elements that match a specific query. (More on the query format in a minute.)

② Each element — including the root element, but also child elements — has a `findall()` method. It finds all matching elements among the element's children. But why aren't there any results? Although it may not be obvious, this particular query only searches the element's children. Since the root `feed` element has no child named `feed`, this query returns an empty list.

③ This result may also surprise you. There is an author element in this document; in fact, there are three (one in each entry). But those `author` elements are not *direct children* of the root element; they are "grandchildren" (literally, a child element of a child element). If you want to look for author

elements at any nesting level, you can do that, but the query format is slightly different.

```
import xml.etree.ElementTree as etree
tree = etree.parse('feed.xml')

print (tree.findall('{http://www.w3.org/2005/Atom}entry'))   #①
#[<Element {http://www.w3.org/2005/Atom}entry at e2b4e0>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b510>,
# <Element {http://www.w3.org/2005/Atom}entry at e2b540>]

print (tree.findall('{http://www.w3.org/2005/Atom}author'))  #②
#[]
```

① For convenience, the tree object (returned from the `etree.parse()` function) has several methods that mirror the methods on the root element. The results are the same as if you had called the `tree.getroot().findall()` method.

② Perhaps surprisingly, this query does not find the `author` elements in this document. Why not? Because this is just a shortcut for `tree.getroot().findall('{http://www.w3.org/2005/Atom}author')`, which means "find all the `author` elements that are children of the root element." The `author` elements are not children of the root element; they're children of the `entry` elements. Thus the query doesn't return any matches.

There is also a `find()` method which returns the first matching element. This is useful for situations where you are only expecting one match, or if there are multiple matches, you only care about the first one.

```
import xml.etree.ElementTree as etree
tree = etree.parse('feed.xml')

entries = tree.findall('{http://www.w3.org/2005/Atom}entry')          #①
print (len(entries))
#3

title_element = entries[0].find('{http://www.w3.org/2005/Atom}title')  #②
print (title_element.text)
#Dive into history, 2009 edition

foo_element = entries[0].find('{http://www.w3.org/2005/Atom}foo')      #③
foo_element
```

```
print (type(foo_element))
#<class 'NoneType'>
```

① You saw this in the previous example. It finds all the `atom:entry` elements.

② The `find()` method takes an `ElementTree` query and returns the first matching element.

③ There are no elements in this entry named `foo`, so this returns `None`.

> *There is a “gotcha” with the **find()** method that will eventually bite you. In a boolean context, ElementTree element objects will evaluate to **False** if they contain no children (i.e. if **len(element)** is 0). This means that **if element.find('...')** is not testing whether the **find()** method found a matching element; it's testing whether that matching element has any child elements! To test whether the find() method returned an element, use if **element.find('...') is not None**.*

There *is* a way to search for *descendant* elements, i.e. children, grandchildren, and any element at any nesting level.

```
import xml.etree.ElementTree as etree
tree = etree.parse('feed.xml')

all_links = tree.findall('//{http://www.w3.org/2005/Atom}link')          #①
print (all_links)
#[<Element {http://www.w3.org/2005/Atom}link at e181b0>,
# <Element {http://www.w3.org/2005/Atom}link at e2b570>,
# <Element {http://www.w3.org/2005/Atom}link at e2b480>,
# <Element {http://www.w3.org/2005/Atom}link at e2b5a0>]

print (all_links[0].attrib)                                              #②
#{'href': 'http://diveintomark.org/',
# 'type': 'text/html', 'rel': 'alternate'}

print (all_links[1].attrib )                                             #③
#{'href': 'http://diveintomark.org/archives/2009/03/27/dive-into-history-2009-edition',
# 'type': 'text/html', 'rel': 'alternate'}

print (all_links[2].attrib)
#{'href': 'http://diveintomark.org/archives/2009/03/21/accessibility-is-a-harsh-mistress',
# 'type': 'text/html', 'rel': 'alternate'}

print (all_links[3].attrib)
#{'href': 'http://diveintomark.org/archives/2008/12/18/give-part-1-container-formats',
# 'type': 'text/html', 'rel': 'alternate'}
# 'type': 'text/html', 'rel': 'alternate'}
```

① This query — `//{http://www.w3.org/2005/Atom}link` — is very similar to the previous examples, except for the two slashes at the beginning of the query. Those two slashes mean "don't just look for direct children; I want any elements, regardless of nesting level." So the result is a list of four `link` elements, not just one.

② The first result *is* a direct child of the root element. As you can see from its attributes, this is the feed-level alternate link that points to the html version of the website that the feed describes.

③ The other three results are each entry-level alternate links. Each `entry` has a single `link` child element, and because of the double slash at the beginning of the query, this query finds all of them.

Overall, ElementTree's `findall()` method is a very powerful feature, but the query language can be a bit surprising. It is officially described as "limited support for XPath expressions." XPath is a W3C standard for querying XML documents. ElementTree's query language is similar enough to XPath to do basic searching, but dissimilar enough that it may annoy you if you already know XPath. Now let's look at a third-party XML library that extends the ElementTree api with full XPath support.