Theory: XML in Python

37 minutes 0 / 5 problems solved

As you know from previous topics, XML is a format that stores data in a hierarchical structure. An **element** is a building block of XML and it consists of a **starting tag**, the corresponding **ending tag**, and their content. Now, we are going to learn how to work with XML in Python.

§1. Getting ready

There is a built-in Python submodule called `xml.etree` which can parse XML. However, we will use another library, `lxml`, and its same-name submodule `etree`. The reason is that the latter submodule processes XML documents faster, the core of this library being written in C language.

Since it is an external library, you should install it first:

```
pip install lxml
```

Then, import the `etree` module in your code. If you use PyCharm, you can write this line without installing `lxml`, and the IDE will suggest installing the library automatically.

```
from lxml import etree
```

We will work with two classes from this module: `Element` and `ElementTree`.

- An instance of the `Element` class represents **one element in the XML document**. It stores information about the tag name, attributes of the tag and references to child elements.
- `ElementTree` represents **the whole XML document**. It contains some general information about the XML document such as its encoding and the version of XML, and also has a reference to the root element of the document.

§2. From text to XML

Let's see how to parse XML documents: they can be parsed from a string or a file.

- **To parse a string**, just call the `fromstring()` function that returns the root element of the document.

```
xml_string = "<a><b>hello</b></a>"

root = etree.fromstring(xml_string)

print(type(root))  # <class 'lxml.etree._Element'>
```

- **To parse XML from a file**, use the `parse()` function. It returns an instance of the `ElementTree` class, so you should use the `getroot()` method of this class to obtain the root of the document.

```
xml_path = "xml_file.xml"

tree = etree.parse(xml_path)
print(type(tree))  # <class 'lxml.etree._ElementTree'>

root = tree.getroot()
print(type(root))  # <class 'lxml.etree._Element'>
```

Also, it might be useful to print your XML document so you can look at it. For this, there is the `dump()` function. It takes an element of the XML document and prints it with all its content in a beautiful way.

Current topic:

XML in Python

Topic depends on:

- ✗ XML
- ✓ Type casting 12★
- ✓ Dictionary 6★ Stage 1
- ✓ Nested lists
- ✓ Methods 3★ Stage 1
- ✗ Pip Stage 1

Table of contents:

↑ XML in Python

§1. Getting ready

§2. From text to XML

§3. Traversing the XML tree

§4. Accessing attributes

§5. From XML to text

§6. Conclusions

Feedback & Comments

```
xml_string = "<a><b>hello</b></a>"
root = etree.fromstring(xml_string)

etree.dump(root)
# <a>
#   <b>hello</b>
# </a>
```

Now let's see how to traverse an XML document and access the information in it.

§3. Traversing the XML tree

Since the important information is often stored not in the root element, you should be able to access child elements. In the lxml library it is very convenient because the `Element` class imitates well-known Python lists. Let's see an example.

First, we parse an XML document and print it to understand its structure.

```
xml_file = "xml_file.xml"
root = etree.parse(xml_file).getroot()
etree.dump(root)

# <country>
#   <name>United Stated of America</name>
#   <capital>Washington</capital>
#   <states>
#     <state>California</state>

#     <state>Texas</state>

#     <state>Florida</state>

#     <state>Hawaii</state>

#   </states>

# </country>
```

A child element can be accessed by specifying its **index** among other subelements in **square brackets**. Our root element, `<country>`, has three child elements: `<name>`, `<capital>`, and `<states>`. The tag containing the country's capital has index `1` (remember that indices start from 0), so you can access it in this way:

```
etree.dump(root[1])  # <capital>Washington</capital>
```

The structure of the entire XML document behaves like a **collection of lists** where each list except the root is nested into another. So, to print all states of the US that are mentioned in our document, we should first get the `<states>` element and then iterate over all its subelements. This can be done in the same way as when working with lists:

```
states = root[2]
for state in states:
    print(state.text)

# California
# Texas
# Florida
# Hawaii
```

Note that when an element contains text, it is stored in its `text` attribute.

§4. Accessing attributes

The data is not necessarily stored as raw text inside tags, there are also **attributes** storing some information inside the starting tags.

Let's load a new xml document with information contained in attributes.

```
xml_file = "xml_file1.xml"
root = etree.parse(xml_file).getroot()
etree.dump(root)

# <country name="United Stated of America" capital="Washington">
#   <states>
#     <state name="Hawaii"/>
#     <state name="Florida"/>
#     <state name="Texas"/>

#     <state name="California"/>

#   </states>

# </country>
```

The `Element` behaves like a list when we try to access its subelements. But when we want to get attributes of a tag, the element works like a **dictionary**. The `get()` method is used to access the specified attribute. If there is no such attribute, it returns `None`. Note, that unlike a dictionary, you can't specify the attribute in square brackets.

```
states = root[0]
for state in states:
    print(state.get('name'))

# Hawaii
# Florida
# Texas
# California
```

The `keys()` and `items()` methods can be used to get all attributes of a tag:

```
print(root.keys())     # ['name', 'capital']

print(root.items())    # [('name', 'United Stated of America'), ('capital', 'Washington')]
```

§5. From XML to text

Finally, after getting all information we need, we can save an XML document.

The function `tostring()` takes an element and returns a bytes object that can be later saved to a file.

```
xml_string = "<a><b>hello</b></a>"
root = etree.fromstring(xml_string)

print(etree.tostring(root))  # b'<a><b>hello</b></a>'
```

The method `write()` saves an instance of `ElementTree` directly to a file. If we have worked with an lxml `Element`, we should convert it to `ElementTree` first.

```
xml_string = "<a><b>hello</b></a>"
root = etree.fromstring(xml_string)


tree = etree.ElementTree(root)  # create an instance of ElementTree in order to save it
tree.write("xml_file.xml")
```

§6. Conclusions

- An element of an XML document in the `xml.etree` module is represented by an instance of the `Element` class.
- Work with `Element` as with a list if you want to access its subelements or as with a dictionary to access the attributes.
- Methods `fromstring()` and `parse()` are used to import XML object from a string or a file, while methods `tostring()` and `write()` allow us to save

XML objects back to files.

Report a typo

39 users liked this theory. **8** didn't like it. **What about you?**

Start practicing

Comments (0) Hints (0) Useful links (0) Show discussion