



XML, JSON

XML File Handling

- Introduction to XML and Parsing
- Python XML Parsing Modules
- `xml.etree.ElementTree` Module
 - Parsing
 - Finding Elements
 - Modifying Elements

XML :

XML stands for eXtensible Markup Language. It was designed to store and transport small to medium amounts of data and is widely used for sharing structured information.

- XML parser is a software library or a package that provides interface for client applications to work with XML documents.
- It checks for proper format of the XML document and may also validate the XML documents. Modern day browsers have built-in XML parsers. The goal of a parser is to transform XML into a readable code.

XML Modules

To create XML file

The tree is a hierarchical structure of elements starting with root followed by other elements. Each element is created by using Element() function of this module.

Introduction to ElementTree

The XML tree structure makes navigation, modification, and removal relatively simple programmatically. Python has a built in library, `ElementTree`, that has functions to read and manipulate XMLs (and other similarly structured files).

First, import `ElementTree`. It's a common practice to use the alias of `ET`:

```
import xml.etree.ElementTree as ET
```

XML file : "sample.xml"

```
<metadata>
  <food>
    <items name="special_Idli">Idli</items>
    <price>70Rs</price>
  </food>
  <food>
    <items name="special_Dosa">Masala Dosa</items>
    <price>80Rs</price>
  </food>
  <food>
    <items name="special_Poori">Poori</items>
    <price>70Rs</price>
  </food>
</metadata>
```

Parse XML file

```
import xml.etree.ElementTree as ET

tree = ET.parse("sample.xml")
root = tree.getroot()
```

Every part of a tree (root included) has a tag that describes the element. In addition, as you have seen in the introduction, elements might have attributes, which are additional descriptors

At the top level, you see that this XML is rooted in the **metadata** tag.

`root.tag` → Root tag element

`'metadata'` `root.attrib` → to check the attributes of tag elements

```
for child in root:  
    print(child.tag, child.attrib)
```

The tag 'food' not contain any attributes.

```
food {}  
food {}  
food {}
```

Typically it is helpful to know all the elements in the entire tree. One useful function for doing that is `root.iter()`. You can put this function into a "for" loop and it will iterate over the entire tree.

```
for child in root.iter():  
    print(child.tag, child.attrib)
```

```
metadata {}  
food {}  
items {'name': 'special_Idli'}  
price {}  
food {}  
items {'name': 'special_Dosa'}  
price {}  
food {}  
items {'name': 'special_Poori'}  
price {}
```

```
for child in root.iter("items"):
    print(child.tag, child.attrib)
    print("Text Value of items Tag --> : ",child.text, "\n")
```

```
items {'name': 'special_Idli'}
```

```
Text Value of items Tag --> : Idli
```

```
items {'name': 'special_Dosa'}
```

```
Text Value of items Tag --> : Masala Dosa
```

```
items {'name': 'special_Poori'}
```

```
Text Value of items Tag --> : Poori
```


Use findall method to check specific tag Element

```
for child in root.findall("food"):
    item = child.find('items').text
    price = child.find("price").text
    print(item, "--->", price)
```

Idli ---> 70Rs

Masala Dosa ---> 80Rs

Poori ---> 70Rs

Adding tag element in XML file → root[0] means only for the first food tag element

```
ET.SubElement(root[0], "description")

for x in root.iter("description"):
    data = "South Indian Special"
    x.text = data

tree.write("sample.xml")
```

```
<metadata>
  <food>
    <items name="special_Idli">Idli</items>
    <price>70Rs</price>
    <description>South Indian Special</description></food>
  <food>
    <items name="special_Dosa">Masala Dosa</items>
    <price>80Rs</price>
  </food>
```

```
for x in root:
    ET.SubElement(x, "description")
```

```
for x in root.iter("description"):
    data = "South Indian Special"
    x.text = data
```

```
tree.write("sample.xml")
```

```
<metadata>
  <food>
    <items name="special_Idli">Idli</items>
    <price>70Rs</price>
    <description>South Indian Special</description></food>
  <food>
    <items name="special_Dosa">Masala Dosa</items>
    <price>80Rs</price>
    <description>South Indian Special</description></food>
  <food>
    <items name="special_Poori">Poori</items>
    <price>70Rs</price>
    <description>South Indian Special</description></food>
</metadata>
```

What is JSON?

JSON (Java Script Object Notation) is a popular file format for storing and transmitting data in web applications.

```
{
  "squadName" : "Super Hero Squad",
  "homeTown" : "Metro City",
  "formed" : 2016,
  "secretBase" : "Super tower",
  "active" : true,
  "members" : [
    {
      "name" : "Molecule Man",
      "age" : 29,
      "secretIdentity" : "Dan Jukes",
      "powers" : [
        "Radiation resistance",
        "Turning tiny",
        "Radiation blast"
      ]
    }
  ]
}
```

The process of converting python to JSON is done by serialization.

The term 'serialization' refers to the transformation of data into a series of bytes (hence serial) to be stored.

Since the JSON is read by other languages, various Python-specific objects are converted to a standard JSON acceptable format. For example, a list and a tuple, which are python specific, are converted to arrays when storing it as JSON format.

Likewise, a JSON object is imported and parsed as a python `dict`.

Python	JSON
dict	object
list, tuple	array
str	string
int, long, float	number
True	true
False	false
None	Null

Example 1: Python JSON to dict

You can parse a JSON string using `json.loads()` method. The method returns a dictionary.

```
import json

person = '{"name": "Bob", "languages": ["English", "French"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```

Here, `person` is a JSON string, and `person_dict` is a dictionary.

Read JSON File

```
import json

with open('path_to_file/person.json', 'r') as f:
    data = json.load(f)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print(data)
```

Here, we have used the `open()` function to read the json file. Then, the file is parsed using `json.load()` method which gives us a dictionary named `data`.


Example 3: Convert dict to JSON

```
import json

person_dict = {'name': 'Bob',
               'age': 12,
               'children': None}
person_json = json.dumps(person_dict)

# Output: {"name": "Bob", "age": 12, "children": null}
print(person_json)
```

Example 5: Python pretty print JSON



```
import json

person_string = '{"name": "Bob", "languages": "English", "numbers": [2, 1.6, null]}'

# Getting dictionary
person_dict = json.loads(person_string)

# Pretty Printing JSON string back
print(json.dumps(person_dict, indent = 4, sort_keys=True))
```

When you run the program, the output will be:

```
{
    "languages": "English",
    "name": "Bob",
    "numbers": [
        2,
        1.6,
        null
    ]
}
```

Example 4: Writing JSON to a file

```
import json

person_dict = {"name": "Bob",
               "languages": ["English", "French"],
               "married": True,
               "age": 32
              }

with open('person.txt', 'w') as json_file:
    json.dump(person_dict, json_file)
```

Difference between JSON and XML

JSON is easy to learn.	XML is quite more complex to learn than JSON.
It is simple to read and write.	It is more complex to read and write than JSON.
It is data-oriented.	It is document-oriented.
JSON is less secure in comparison to XML.	XML is highly secured.
It doesn't provide display capabilities.	It provides the display capability because it is a markup language.
It supports the array.	It doesn't support the array
<p>Example :</p> <pre>[{ "name" : "Peter", "employed id" : "E231", "present" : true, "numberofdayspresent" : 29 }, { "name" : "Jhon", "employed id" : "E331", "present" : true, "numberofdayspresent" : 27 }]</pre>	<p>Example :</p> <pre><name> <name>Peter</name> </name></pre>