XML

PARSING

INDEX:

WHAT IS XML?

- **XML** : e**X**tensible **M**arkup **L**anguage
- It is used to transport, organize, and store data
- Allows you to create your own elements
- It carries and stores the data without worrying about how the data will be presented
- It is an industry standard created by the W3C
- It is self-describing
- It is a markup language , not a programming language
- Just information wrapped in tags
- It does not have pre-defined tags
- XML simplifies
    - Data sharing
    - Data transport
    - Data availability
    - Platform changes

XML SYNTAX:

- <?xml version=”1.0” encoding = “UTF-8”?> : The prolog is the first line in an XML document
- <rootElementName>

  </rootElementName>
- <example> is different tag than <Example>
- <text><bold> </text></bold> **(WRONG)**
- <text><bold> </bold></text> **(RIGHT)**
- <!—example comment-->

XML TREE AND ELEMENTS

- XML documents form a tree structure with branching and leaves

- **Syntax:**

```
<root>
    <child>
        <subchild>…..</subchild>
    </child>
</root>
```

- **Example :**

```
<store>
        <item category=”weapon”>
                <power> 50 </power>
                <cost> 30 </cost>
        </item>
</store>
```

PYTHON XML PASRSING MODULES

Python allows parsing these XML documents using two modules namely, the

- xml.etree.ElementTree module
- Minidom (Minimal DOM Implementation).

Parsing means to read information from a file and split it into pieces by identifying parts of that particular XML file. Let’s move on further to see how we can use these modules to parse XML data.

1. xml.etree.ElementTree Module:

This module helps us format XML data in a tree structure which is the most natural representation of hierarchical data. Element type allows storage of hierarchical data structures in memory and has the following properties:

Property	Description
Tag	It is a string representing the type of data being stored
Attributes	Consists of a number of attributes stored as dictionaries
Text String	A text string having information that needs to be displayed
Tail String	Can also have tail strings if necessary
Child Elements	Consists of a number of child elements stored as sequences

ElementTree is a class that wraps the element structure and allows conversion to and from XML

```
root = ET.fromstring(country_data_as_string)
```

fromstring() parses XML from a string directly into an Element, which is the root element of the parsed tree. Other parsing functions may create an ElementTree. Check the documentation to be sure.

As an Element, root has a tag and a dictionary of attributes:

```
>>> root.tag
'data'
>>> root.attrib
{}
```

It also has children nodes over which we can iterate:

```
>>> for child in root:
...     print(child.tag, child.attrib)
...
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

Children are nested, and we can access specific child nodes by index:

```
>>> root[0][1].text
'2008'
```

here are two ways to parse the file using 'ElementTree' module.

1. ***parse() function***
2. ***fromstring() function***.

The parse () function parses XML document which is supplied as a file whereas, fromstring parses XML when supplied as a string i.e within triple quotes.

Element Objects

class xml.etree.ElementTree.**Element**(*tag*, *attrib={}*, ***extra*)
Element class. This class defines the Element interface, and provides a reference implementation of this interface.

The element name, attribute names, and attribute values can be either bytestrings or Unicode strings. *tag* is the element name. *attrib* is an optional dictionary, containing element attributes. *extra* contains additional attributes, given as keyword arguments.

tag
A string identifying what kind of data this element represents (the element type, in other words).

text, **tail**
These attributes can be used to hold additional data associated with the element. Their values are usually strings but may be any application-specific object. If the element is created from an XML file, the *text* attribute holds either the text between the element's start tag and its first child or end tag, or None, and the *tail* attribute holds either the text between the element's end tag and the next tag, or None. For the XML data

```
<a><b>1<c>2<d/>3</c></b>4</a>
```

the *a* element has None for both *text* and *tail* attributes, the *b* element has *text* "1" and *tail* "4", the *c* element has *text* "2" and *tail* None, and the *d* element has *text* None and *tail* "3".

To collect the inner text of an element, see itertext(), for example "".join(element.itertext()).

Applications may store arbitrary objects in these attributes.

attrib
A dictionary containing the element's attributes. Note that while the *attrib* value is always a real mutable Python dictionary, an ElementTree implementation may choose to use another internal representation, and create the dictionary only if someone asks for it. To take advantage of such implementations, use the dictionary methods below whenever possible.

The following dictionary-like methods work on the element attributes.

clear()
Resets an element. This function removes all subelements, clears all attributes, and sets the text and tail attributes to None.

get(*key*, *default=None*)
Gets the element attribute named *key*.

Returns the attribute value, or *default* if the attribute was not found.

items()
Returns the element attributes as a sequence of (name, value) pairs. The attributes are returned in an arbitrary order.

keys()
Returns the elements attribute names as a list. The names are returned in an arbitrary order.

set(*key*, *value*)
Set the attribute *key* on the element to *value*.

The following methods work on the element's children (subelements).

append(*subelement*)
Adds the element *subelement* to the end of this element's internal list of subelements. Raises TypeError if *subelement* is not an Element.

extend(*subelements*)
Appends *subelements* from a sequence object with zero or more elements.
Raises TypeError if a subelement is not an Element.

find(*match, namespaces=None*)
Finds the first subelement matching *match*. *match* may be a tag name or a path. Returns an element instance or None. *namespaces* is an optional mapping from namespace prefix to full name. Pass '' as prefix to move all unprefixed tag names in the expression into the given namespace.

findall(*match, namespaces=None*)
Finds all matching subelements, by tag name or path. Returns a list containing all matching elements in document order. *namespaces* is an optional mapping from namespace prefix to full name. Pass '' as prefix to move all unprefixed tag names in the expression into the given namespace.

findtext(*match, default=None, namespaces=None*)
Finds text for the first subelement matching *match*. *match* may be a tag name or a path. Returns the text content of the first matching element, or *default* if no element was found. Note that if the matching element has no text content an empty string is returned. *namespaces* is an optional mapping from namespace prefix to full name. Pass '' as prefix to move all unprefixed tag names in the expression into the given namespace.

insert(*index, subelement*)
Inserts *subelement* at the given position in this element. Raises TypeError if *subelement* is not an Element.

iter(*tag=None*)
Creates a tree iterator with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order. If *tag* is not None or '*', only elements whose tag equals *tag* are returned from the iterator. If the tree structure is modified during iteration, the result is undefined.

iterfind(*match, namespaces=None*)
Finds all matching subelements, by tag name or path. Returns an iterable yielding all matching elements in document order. *namespaces* is an optional mapping from namespace prefix to full name.

itertext()
Creates a text iterator. The iterator loops over this element and all subelements, in document order, and returns all inner text.

makeelement(*tag, attrib*)
Creates a new element object of the same type as this element. Do not call this method, use the SubElement() factory function instead.

remove(*subelement*)
Removes *subelement* from the element. Unlike the find* methods this method compares elements based on the instance identity, not on tag value or contents.

ElementTree Objects

***class* xml.etree.ElementTree.ElementTree(*element=None, file=None*)**
ElementTree wrapper class. This class represents an entire element hierarchy, and adds some extra support for serialization to and from standard XML.

element is the root element. The tree is initialized with the contents of the XML *file* if given.

_setroot(*element*)
Replaces the root element for this tree. This discards the current contents of the tree, and replaces it with the given element. Use with care. *element* is an element instance.

find(*match, namespaces=None*)
Same as Element.find(), starting at the root of the tree.

findall(*match, namespaces=None*)
Same as Element.findall(), starting at the root of the tree.

findtext(*match, default=None, namespaces=None*)
Same as Element.findtext(), starting at the root of the tree.

getroot()
Returns the root element for this tree.

iter(*tag=None*)
Creates and returns a tree iterator for the root element. The iterator loops over all elements in this tree, in section order. *tag* is the tag to look for (default is to return all elements).

iterfind(*match, namespaces=None*)
Same as Element.iterfind(), starting at the root of the tree.

parse(*source, parser=None*)
Loads an external XML section into this element tree. *source* is a file name or file object. *parser* is an optional parser instance. If not given, the standard XMLParser parser is used. Returns the section root element.

write(*file, encoding='us-ascii', xml_declaration=None, default_namespace=None, method='xml', *, short_empty_elements=True*)
Writes the element tree to a file, as XML. *file* is a file name, or a file object opened for writing. *encoding* [1] is the output encoding (default is US-ASCII). *xml_declaration* controls if an XML declaration should be added to the file

- # Using fromstring() function:

You can also use fromstring() function to parse your string data. In case you want to do this, pass your XML as a string within triple quotes as follows:

```
import xml.etree.ElementTree as ET
data='''<?xml version="1.0" encoding="UTF-8"?>
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description>
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
</metadata>
'''
myroot = ET.fromstring(data)
#print(myroot)
print(myroot.tag)
```

the above code will return the same output as the previous one. Please note that the XML document used as a string is just one part of ‘Sample.xml’ which I have used for better visibility. You can use the complete XML document as well.

You can also retrieve the root tag by using the ‘tag’ object as follows:

EXAMPLE:

print(myroot.tag)

OUTPUT: metadata

You can also slice the tag string output by just specifying which part of the string you want to see in your output.

EXAMPLE:

print(myroot.tag[0:4])

OUTPUT: meta

As mentioned earlier, tags can have dictionary attributes as well. To check if the root tag has any attributes you can use the ‘attrib’ object as follows:

EXAMPLE:

print(myroot.attrib)

OUTPUT: {}

the output is an empty dictionary because our root tag has no attributes.

XML PARSNG USING fromstring() METHOD:

```
from xml.etree import ElementTree as et

root = et.fromstring("""<?xml version="1.0"?>
<store>
    <item category="weapon">
        <power>50</power>
        <cost>30</cost>
    </item>
    <item category="spell">
        <power>80</power>
        <cost>100</cost>
    </item>
</store>
 """)

print(root.tag)
```

```
print(root.attrib)

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

OUTPUT:

C:\Users\vlab\PycharmProjects\XML_Parsing\.venv\Scripts\python.exe
C:\Users\vlab\PycharmProjects\XML_Parsing\xml_parse.py

store

{}

item {'category': 'weapon'}

item {'category': 'spell'}

Process finished with exit code 0

MODIFYING THE XML USING fromstring():

```
import xml.etree.ElementTree
from xml.etree import ElementTree as et

root = et.fromstring("""<?xml version="1.0"?>
<store>
    <item category="weapon">
        <power>50</power>
        <cost>30</cost>
    </item>
    <item category="spell">
        <power>80</power>
        <cost>100</cost>
    </item>
</store>
 """)


root.tag = "shop"
print(root.tag)

new_ele = xml.etree.ElementTree.SubElement(root,"item")
new_ele.attrib = {"category":"shield"}

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

```
print("after removing newly added sub element")
root.remove(new_ele)
for child in root:
    print(child.tag,child.attrib)
```

OUTPUT:

C:\Users\vlab\PycharmProjects\XML_Parsing\.venv\Scripts\python.exe
C:\Users\vlab\PycharmProjects\XML_Parsing\modify_xml.py

shop

item {'category': 'weapon'}

item {'category': 'spell'}

item {'category': 'shield'}

after removing newly added sub element

item {'category': 'weapon'}

item {'category': 'spell'}

Process finished with exit code 0

- ## Using parse() function:

As mentioned earlier, this function takes XML in file format to parse it. Take a look at the following example:

EXAMPLE:

import xml.etree.ElementTree as ET

mytree = ET.parse('sample.xml')

myroot = mytree.getroot()

sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description>
```

```
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
<food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description>
    Plain paper dosa with chutney
    </description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
    <description>
    Rava upma with bajji
    </description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description>
   Bisi Bele Bath with sev
    </description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description>
    Sweet rava with saffron
    </description>
    <calories>950</calories>
</food>
</metadata>
```

Parse_xmlfile.py

```
import xml.etree.ElementTree as ET
mytree = ET.parse("XML_files/sample.xml")
myroot = mytree.getroot()
print(myroot)
#print(myroot.attrib)

for child in myroot:
    for c in child:
        print("text :",c.text)
        print("tail :",c.tail)
        print("atrribute :",c.attrib)
        print("tag :",c.tag)
```

OUTPUT:

C:\Users\vlab\PycharmProjects\XML_Parsing\.venv\Scripts\python.exe
C:\Users\vlab\PycharmProjects\XML_Parsing\parse_XMLfile.py

<Element 'metadata' at 0x00000136DA4A53F0>

text : Idly

tail :

atrribute : {'name': 'breakfast'}

tag : item

text : $2.5

tail :

atrribute : {}

tag : price

text :

Two idly's with chutney

tail :

atrribute : {}

tag : description

text : 553

tail :

atrribute : {}

tag : calories

text : Paper Dosa

tail :

atrribute : {'name': 'breakfast'}

tag : item

text : $2.7

tail :

atrribute : {}

tag : price

text :

Plain paper dosa with chutney

tail :

atrribute : {}

tag : description

text : 700

tail :

atrribute : {}

tag : calories

text : Upma

tail :

atrribute : {'name': 'breakfast'}

tag : item

text : $3.65

tail :

atrribute : {}

tag : price

text :

Rava upma with bajji

tail :

atrribute : {}

tag : description

text : 600

tail :

atrribute : {}

tag : calories

text : Bisi Bele Bath

tail :

atrribute : {'name': 'breakfast'}

tag : item

text : $4.50

tail :

atrribute : {}

tag : price

text :

Bisi Bele Bath with sev

tail :

atrribute : {}

tag : description

text : 400

tail :

atrribute : {}

tag : calories

text : Kesari Bath

tail :

atrribute : {'name': 'breakfast'}

tag : item

text : $1.95

tail :

atrribute : {}

tag : price

text :

Sweet rava with saffron

tail :

atrribute : {}

tag : description

text : 950

tail :

atrribute : {}

tag : calories

Process finished with exit code 0

As you can see, The first thing you will need to do is to import the xml.etree.ElementTree module. Then, the parse() method parses the 'Sample.xml' file. The getroot() method returns the root element of 'Sample.xml'.

When you execute the above code, you will not see outputs returned but there will be no errors indicating that the code has executed successfully. To check for the root element, you can simply use the print statement as follows:

EXAMPLE:

import xml.etree.ElementTree as ET

mytree = ET.parse('sample.xml')

myroot = mytree.getroot()

print(myroot)

OUTPUT: <Element 'metadata' at 0x033589F0>

The above output indicates that the root element in our XML document is 'metadata'.

if you want to display all the items with their particular price, you can make use of the get() method. This method accesses the element's attributes.

```
import xml.etree.ElementTree as ET
mytree = ET.parse("XML_files/sample.xml")
myroot = mytree.getroot()

for x in myroot.findall('food'):
    item =x.find('item').text
    price = x.find('price').text
    print(item, price)
```

OUTPUT:

Idly $2.5

Paper Dosa $2.7

Upma $3.65

Bisi Bele Bath $4.50

Kesari Bath $1.95

MODIFYING THE XML USING parse() of .xml FILE:

The elements present your XML file can be manipulated. To do this, you can use the set() function. Let us first take a look at how to add something to XML.

Adding to XML:
The following example shows how you can add something to the description of items.

To add a new subtag, you can make use of the SubElement() method.

```
import xml.etree.ElementTree as ET
mytree = ET.parse("XML_files/sample.xml")
myroot = mytree.getroot()

for description in myroot.iter('description'):
    new_desc = str(description.text) + 'wil be served'
    description.text = str(new_desc)
    description.set('updated', 'yes')

mytree.write('XML_files/new.xml')

ET.SubElement(myroot[0], 'speciality')
for x in myroot.iter('speciality'):
    new_desc = 'South Indian Special'
    x.text = str(new_desc)

mytree.write('XML_files/new_subElement.xml')
```

new.xml

```
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description updated="yes">
   Two idly's with chutney
   wil be served</description>
```

```
    <calories>553</calories>
</food>
<food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description updated="yes">
    Plain paper dosa with chutney
    wil be served</description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
    <description updated="yes">
    Rava upma with bajji
    wil be served</description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description updated="yes">
   Bisi Bele Bath with sev
    wil be served</description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description updated="yes">
    Sweet rava with saffron
    wil be served</description>
    <calories>950</calories>
</food>
</metadata>
```

New_subelement.xml

```
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description updated="yes">
   Two idly's with chutney
   wil be served</description>
    <calories>553</calories>
<speciality>South Indian Special</speciality></food>
<food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description updated="yes">
    Plain paper dosa with chutney
    wil be served</description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
    <description updated="yes">
    Rava upma with bajji
```

```
    wil be served</description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description updated="yes">
   Bisi Bele Bath with sev
    wil be served</description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description updated="yes">
    Sweet rava with saffron
    wil be served</description>
    <calories>950</calories>
</food>
```

Deleting from XML:

To delete attributes or sub-elements using ElementTree, you can make use of the pop() method. This method will remove the desired attribute or element that is not needed by the user.

EXAMPLE:

```
myroot[0][0].attrib.pop('name', None)

# create a new XML file with the results
mytree.write('output5.xml')
```

```
import xml.etree.ElementTree as ET
mytree = ET.parse("XML_files/sample.xml")
myroot = mytree.getroot()
myroot[0][0].attrib.pop('name', None)
mytree.write('XML_files/pop_nameAttribute.xml')
```

OUTPUT: pop_nameAttribute.xml

We observe that name attribute has been removed from item tag

```
<food>
    <item>Idly</item>
    <price>$2.5</price>
    <description updated="yes">
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
```

The above image shows that the name attribute has been removed from the item tag. To remove the complete tag, you can use the same pop() method as follows:

```
myroot[0].remove(myroot[0][0])
mytree.write('output6.xml')
```

```
import xml.etree.ElementTree as ET
mytree = ET.parse("XML_files/sample.xml")
myroot = mytree.getroot()

myroot[0].remove(myroot[0][0])
mytree.write('XML_files/pop_tag.xml')
```

OUTPUT: pop_tag.xml

We observe that the item tag(first subelement) has been removed from food tag

```
<food>
    <price>$2.5</price>
    <description updated="yes">
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
```

The output shows that the first subelement of the food tag has been deleted. In case you want to delete all tags, you can make use of the clear() function as follows:

```
myroot[0].clear()
mytree.write('output7.xml')
```

When the above code is executed, the first child of food tag will be completely deleted including all the subtags

```
myroot[0].clear()
mytree.write('XML_files/clear_1stSubelement.xml')
```

OUTPUT: clear_1stSubelement.xml

```
<metadata>
<food /><food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description updated="yes">
    Plain paper dosa with chutney
    wil be served</description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
```

```
    <description updated="yes">
    Rava upma with bajji
    wil be served</description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description updated="yes">
   Bisi Bele Bath with sev
    wil be served</description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description updated="yes">
    Sweet rava with saffron
    wil be served</description>
    <calories>950</calories>
</food>
</metadata>
```

2. xml.dom.minidom Module:

This module is basically used by people who are proficient with DOM (Document Object module). DOM applications often start by parsing XML into DOM. in xml.dom.minidom, this can be achieved in the following ways:

- **Using parseString() Method:**

This method is used when you want to supply the XML to be parsed as a string.

EXAMPLE:

p3 = minidom.parseString('<myxml>Using<empty/> parseString</myxml>')

You can parse XML using any of the above methods. Now let us try to fetch data using this module.

```
from xml.dom import minidom
p1 = minidom.parseString("""<?xml version="1.0"?>
<store>
    <item category="weapon">
        <power>50</power>
        <cost>30</cost>
    </item>
    <item category="spell">
        <power>80</power>
        <cost>100</cost>
    </item>
</store>
 """)
```

```
print(p1)

tagname = p1.getElementsByTagName("item")
print(tagname[0])

print(tagname[0].attributes["category"].value)

print("number of items in menu :",len(tagname))
```

OUTPUT:

C:\Users\vlab\PycharmProjects\XML_Parsing\.venv\Scripts\python.exe
C:\Users\vlab\PycharmProjects\XML_Parsing\xml.dom.minidom\minidom_parseString.py

<xml.dom.minidom.Document object at 0x00000115446F6510>

<DOM Element: item at 0x115441e7260>

weapon

number of items in menu : 2

Process finished with exit code 0

- ## Using the parse() function:

The first method is to make use of the parse() function by supplying the XML file to be parsed as a parameter. For example:

EXAMPLE:

```
from xml.dom import minidom
p1 = minidom.parse("sample.xml");
```

Once you execute this, you will be able to split the XML file and fetch the required data. You can also parse an open file using this function..

EXAMPLE:

```
tagname= dat.getElementsByTagName('item')[0]
print(tagname)
```

If I try to fetch the first element using the GetElementByTagName method, I will see the following output:

OUTPUT:

<DOM Element: item at 0xc6bd00>

Please note that just one output has been returned because I have used [0] subscript for convenience which will be removed in the further examples.

To access the value of the attributes, I will have to make use of the value attribute as follows:

EXAMPLE:

```
dat = minidom.parse('sample.xml')
tagname= dat.getElementsByTagName('item')
print(tagname[0].attributes['name'].value)
```

OUTPUT: breakfast

To retrieve the data present in these tags, you can make use of the data attribute as follows:

EXAMPLE:

```
print(tagname[1].firstChild.data)
```

OUTPUT: Paper Dosa

You can also split and retrieve the value of the attributes using the value attribute.

EXAMPLE:

```
print(items[1].attributes['name'].value)
```

OUTPUT: breakfast

To print out all the items available in our menu, you can loop through the items and return all the items.

EXAMPLE:

```
for x in items:
    print(x.firstChild.data)
```

OUTPUT:

Idly
Paper Dosa
Upma
Bisi Bele Bath
Kesari Bath

To calculate the number of items on our menu, you can make use of the len() function as follows:

EXAMPLE:

```
print(len(items))
```

OUTPUT: 5

The output specifies that our menu consists of 5 items

```
from xml.dom import minidom
p1 = minidom.parse("../XML_files/sample.xml")

print(p1)

tagname = p1.getElementsByTagName("item")
print(tagname[0])

print(tagname[0].attributes["name"].value)
print(tagname[1].firstChild.data)

for x in tagname:
    print(x.firstChild.data)

print("number of items in menu :",len(tagname))
```

OUTPUT:

C:\Users\vlab\PycharmProjects\XML_Parsing\.venv\Scripts\python.exe
C:\Users\vlab\PycharmProjects\XML_Parsing\xml.dom.minidom\minidom_parse.py

<xml.dom.minidom.Document object at 0x0000025A2CDE64B0>

<DOM Element: item at 0x25a2c877380>

breakfast

Paper Dosa

Idly

Paper Dosa

Upma

Bisi Bele Bath

Kesari Bath

number of items in menu : 5

Process finished with exit code 0

XML TO CSV

INPUT:

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description>
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
<food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description>
    Plain paper dosa with chutney
    </description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
    <description>
    Rava upma with bajji
    </description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description>
   Bisi Bele Bath with sev
    </description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description>
    Sweet rava with saffron
    </description>
    <calories>950</calories>
</food>
</metadata>
```

CODE:

```
# Importing the required libraries
import xml.etree.ElementTree as Xet
import pandas as pd
```

```
cols = ["item", "price", "description", "calories"]
rows = []

# Parsing the XML file
xmlparse = Xet.parse('../XML_files/sample.xml')
root = xmlparse.getroot()
for i in root:
    item = i.find("item").text
    price = i.find("price").text
    description = i.find("description").text
    calories = i.find("calories").text

    rows.append({"item": item,
                 "price": price,
                 "description": description,
                 "calories": calories})

df = pd.DataFrame(rows, columns=cols)

# Writing dataframe to csv
df.to_csv('../CSV_files/food.csv')
```

OUTPUT: Saved in food.csv

0,Idly,$2.5,Two idly's with chutney,553

1,Paper Dosa,$2.7, Plain paper dosa with chutney,700

2,Upma,$3.65,Rava upma with bajji,600

3,Bisi Bele Bath,$4.50,Bisi Bele Bath with sev,400

4,Kesari Bath,$1.95,Sweet rava with saffron,950

XML TO JSON:

INPUT:

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
<food>
    <item name="breakfast">Idly</item>
    <price>$2.5</price>
    <description>
   Two idly's with chutney
   </description>
    <calories>553</calories>
</food>
<food>
    <item name="breakfast">Paper Dosa</item>
    <price>$2.7</price>
    <description>
    Plain paper dosa with chutney
    </description>
    <calories>700</calories>
</food>
<food>
    <item name="breakfast">Upma</item>
    <price>$3.65</price>
    <description>
    Rava upma with bajji
    </description>
    <calories>600</calories>
</food>
<food>
    <item name="breakfast">Bisi Bele Bath</item>
    <price>$4.50</price>
    <description>
   Bisi Bele Bath with sev
    </description>
    <calories>400</calories>
</food>
<food>
    <item name="breakfast">Kesari Bath</item>
    <price>$1.95</price>
    <description>
    Sweet rava with saffron
    </description>
    <calories>950</calories>
</food>
</metadata>
```

CODE:

```
import json
import xmltodict


with open("../XML_files/sample.xml") as xml_file:
```

```
    data_dict = xmltodict.parse(xml_file.read())
    xml_file.close()

    json_data = json.dumps(data_dict, indent=4)

    with open("../Json_files/data.json", "w") as json_file:
        json_file.write(json_data)
        json_file.close()
```

OUTPUT:


```
{
    "metadata": {
        "food": [
            {
                "item": {
                    "@name": "breakfast",
                    "#text": "Idly"
                },
                "price": "$2.5",
                "description": "Two idly's with chutney",
                "calories": "553"
            },
            {
                "item": {
                    "@name": "breakfast",
                    "#text": "Paper Dosa"
                },
                "price": "$2.7",
                "description": "Plain paper dosa with chutney",
                "calories": "700"
            },
            {
                "item": {
                    "@name": "breakfast",
                    "#text": "Upma"
                },
                "price": "$3.65",
                "description": "Rava upma with bajji",
                "calories": "600"
            },
            {
                "item": {
                    "@name": "breakfast",
                    "#text": "Bisi Bele Bath"
                },
                "price": "$4.50",
                "description": "Bisi Bele Bath with sev",
                "calories": "400"
            },
            {
                "item": {
                    "@name": "breakfast",
                    "#text": "Kesari Bath"
                },
                "price": "$1.95",
                "description": "Sweet rava with saffron",
                "calories": "950"
            }
        ]
```

```
    }
}
```

CSV TO XML:

INPUT:

name,age,email

John,30,john@example.com

Alice,25,alice@example.com

Bob,35,bob@example.com

CODE:

```
import pandas as pd
import xml.etree.ElementTree as ET
import xml.dom.minidom


def csv_to_xml(csv_file, xml_file):
    # Read CSV file into a pandas DataFrame
    df = pd.read_csv(csv_file)

    # Create root element for XML tree
    root = ET.Element('data')

    # Iterate over rows in DataFrame
    for index, row in df.iterrows():
        # Create child element for each row
        item = ET.SubElement(root, 'item')

        # Add data from DataFrame row as child elements
        for col in df.columns:
            ET.SubElement(item, col).text = str(row[col])

    # Create XML tree
    tree = ET.ElementTree(root)

    # Convert XML tree to string
    xml_string = ET.tostring(root, encoding='unicode')

    # Prettify XML string
    dom = xml.dom.minidom.parseString(xml_string)
    prettified_xml = dom.toprettyxml(indent="  ")

    # Write prettified XML to file
    with open(xml_file, "w") as f:
        f.write(prettified_xml)


# Example usage
csv_to_xml('../CSV_files/email.csv', '../XML_files/email.xml')
```

OUTPUT:

```
<?xml version="1.0" ?>
<data>
  <item>
    <name>John</name>
    <age>30</age>
    <email>john@example.com</email>
  </item>
  <item>
    <name>Alice</name>
    <age>25</age>
    <email>alice@example.com</email>
  </item>
  <item>
    <name>Bob</name>
    <age>35</age>
    <email>bob@example.com</email>
  </item>
</data>
```

JSON TO XML:

INPUT:

```
{
  "employee": {
  "name": "John Doe",
  "age": "35",
  "job": {
    "title": "Software Engineer",
    "department": "IT",
    "years_of_experience": "10"},
  "address": {"street": "123 Main St.",
    "city": "San Francisco",
    "state": "CA",
    "zip": "94102"}
  }
}
```

CODE:

```
import json
import xmltodict

with open("../Json_files/person.json", "r") as file:
    python_dict = json.load(file)

with open("../XML_files/person.xml", "w") as xml_file:
    xmltodict.unparse(python_dict, output=xml_file, pretty=True)
```

OUTPUT:

```
<?xml version="1.0" encoding="utf-8"?>
<employee>
    <name>John Doe</name>
    <age>35</age>
    <job>
       <title>Software Engineer</title>
       <department>IT</department>
       <years_of_experience>10</years_of_experience>
    </job>
    <address>
       <street>123 Main St.</street>
       <city>San Francisco</city>
       <state>CA</state>
       <zip>94102</zip>
    </address>
</employee>
```