4.5 Introduction to XML with Python

What is XML?

XML (Extensible Markup Language) is a way of creating structured documents. XML uses tags and elements to organize and structure a document.

XML also allows nesting of elements.

Shown below is an example XML text:

```
<?xml version = “1.0?”?>
<motorvehicles>
  <vehicle>
      <registration_no>CBB1456</registration_no>
      <make>Toyota</make>
      <model>Premio</model>
  </vehicle>
  <owner>
      <first_name>Amal</first_name>
      <last_name>Perera</last_name>
      <nic>900324770V</nic>
  </owner>
</motorvehicles>
```

In the above XML text, <motorvehicles> is the top level element. There must be a top level element in all XML documents.

<vehicle> is a starting tag and </vehicle> is an ending tag. Sandwiched between the starting and ending tags, we can have either data in text format or nested tags. <registration_no>,<make>, and <model> are elements nested under <vehicle> tag.

Unstructured text can also be contained mixed among XML elements. However, this can cause confusion and is discouraged for data representation.

Shown below is an example of unstructured text mixed with XML elements.

```
<?xml version = “1.0?”?>
<motorvehicles>
  <vehicle>
  This vehicle is a car.
      <registration_no>CBB1456</registration_no>
      <make>Toyota</make>
      <model>Premio</model>
  </vehicle>
  <owner>
      <first_name>Amal</first_name>
      <last_name>Perera</last_name>
      <nic>900324770V</nic>
  </owner>
</motorvehicles>
```

The text "This is a car" in the above example is mixed with XML elements.

XML elements can contain one or more attributes as well. For example, the below XML segment demonstrates <vehicle> element having two attributes, namely, "type" and "year".

```
<vehicle type="car" year="2018">
```

Data represented as attributes can be represented as sub elemets as well.

Python XML Processing

There are multiple libraries that support XML processing in Python. xml.dom.* and xml.sax.* are two standard libraries that Python has to support XML processing. In additon to that there is ElementTree (ET). In this lesson, we will learn how to use ET to process XML with Python.

For the purpose of this lesson, we will use the follow sample XML file (vehicle.xml) in the next sections.

```
<?xml version = “1.0?”?>
```

```
<motorvehicles>
  <vehicle type="car">
      <registration_no>CBB1456</registration_no>
      <make>Toyota</make>
      <model>Premio</model>
  </vehicle>
  <vehicle type="van">
      <registration_no>PR2245</registration_no>
      <make>Mazda</make>
      <model>Bongo</model>
      </vehicle>
</motorvehicles>
```

Importing data from file:

```
>>>import xml.etree.ElementTree as ET
>>>tree = ET.parse('vehicle.xml')
>>>root = tree.getroot()
```

Importing data from a string:

```
>>>root = ET.fromstring(vehicle_xml_data_as_string)
```

Each element has a tag and attributes (including root):

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

We can iterate the children nodes in a loop:

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

vehicle {'type': 'car'}
vehicle {'type': 'van'}
```

Child elements that are nested can be accessed by index like accessing array elements:

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

Accessing attributes:

```
for attr in element.attrib:
...  print(attr+ "=" + element.attrib[attr])
```

Simple searching using iter() function:

```
>>>for element in root.iter(tag='registration_no'):
...  print(element.text)

CBB1456
PR2245
```

In the above example we are providing a tag 'registration_no' to filter only elements with that particular tag. We can also use iter() function without any parameters in which case Python will retrieve all elements without any filtering.

findall() function is another way we can use to search XML in Python.

```
>>>for element in root.findall('vehicle'):
...   regno= element.find('registration_no').text
...   make= element.find('make').text
...   print(regno, make)
CBB1456 Toyota
PR2245 Mazda
```

Modifying XML:

```
>>> for element in root.iter(tag='make'):
...   newmake = 'Nissan'
...  element.text = 'Nissan'
>>> tree.write('output.xml')
```

Above example demostrates how to update the content associated with <make> tags to 'Nissan' and write the output to a new xml named output.xml.

Building an XML:

We can build an XML document by adding elements and subelements using ET.Element() and ET.Subelement() functions. After adding all elements and subelements ET.dump() outputs the XML tree created.

```
>>>a = ET.Element('a')
>>>b = ET.Subelement(a, 'b')
>>>c = ET.Subelement(a, 'c')
>>>d = ET.Subelement(c, 'd')
ET.dump(a)

<a><b /><c><d /></c></a>
```

The following trinket demonstrates all the samples discussed in the lecture video:

GET IN TOUCH

University of Moratuwa
Centre for Open & Distance Learning
CODL

011 308 2787/8

011 265 0301 ext. 3850,3851

open@uom.lk

University Website

CODL Website

Data retention summary