XML Processing

- **Steps to run the utility:**
    1. Enter the name of the XML file: file_name.xml
       Eg: movies.xml

    2. To search the value of tag by specific attribute:
       Enter a tag to search: (or 'q' to quit): tag_name
         Eg: movie
       Enter _attribute_:
         Eg: Trigun

       Enter a tag to search in movie with attrib {'title': 'Trigun'}: stars

    3. To search all the values of the tag
       Enter a tag to search: (or 'q' to quit): tag_name

- **XML processing**:

XML, or Extensible Markup Language, is a markup-language that is commonly used to structure, store, and transfer data between systems.

1. xml.etree.ElementTree: the ElementTree API is a simple and lightweight XML processor

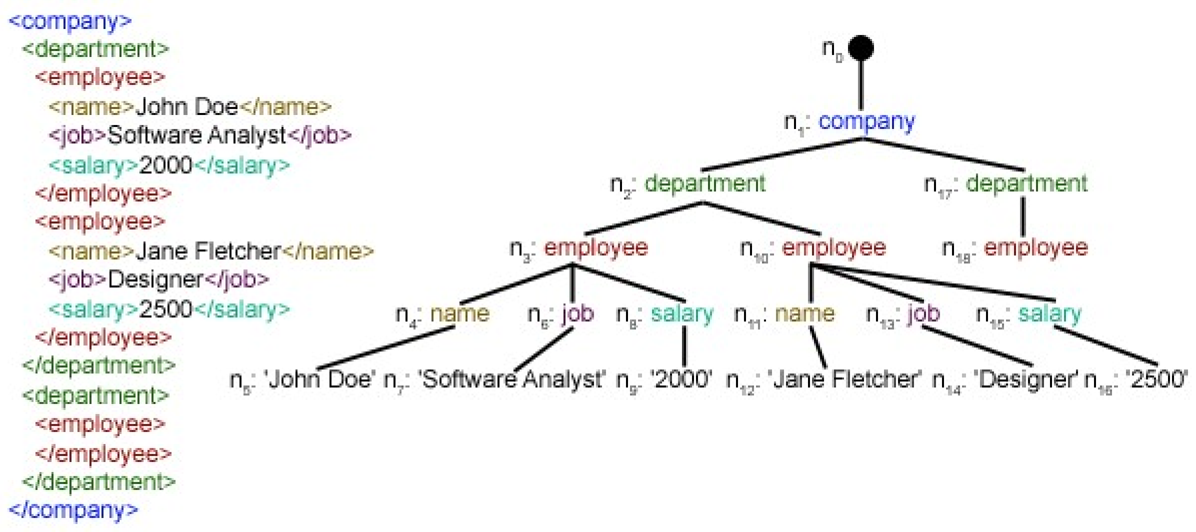
Import:

✓ import xml.etree.ElementTree as ET

The import command with the as keyword, allows to use a simplified name (ET in this case) for the module in the code.

Parsing XML

XML is an inherently hierarchical data format, and the most natural way to represent it is with a tree.



```
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```

Here in xml.etree.ElementTree (call it ET, in short) module, Element Tree has two classes for this purpose – ElementTree represents the whole XML document as a tree, and Element represents a single node in this tree.

Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level. Interactions with a single XML element and its sub-elements are done on the Element level.

- **Terms used in the utility:**

1. parseXML() function :

tree = ET.parse(xmlfile)

Here, we create an ElementTree object by parsing the passed xml file.

root = tree.getroot()

2. For Loops

You can easily iterate over subelements (commonly called “children”) in the root by using a simple “for” loop.

for elem in mytree.iter():

You can expand the use of the iter() function to help with finding particular elements of interest. root.iter() will list all subelements under the root that match the element specified.

3. attrib :
A dictionary containing the element's attributes. 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.

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

5. 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.

6. 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.