Information Infrastructure II

INFO I211 – Spring 2014 – Sections 18530 & 22719

Lecture 24 – 2014.04.21 & 2014.04.22

Instructor:
Mitja Hmeljak,
http://mypage.iu.edu/~mitja
mitja@indiana.edu

What is *XML*?

E**X**tensible **M**arkup **L**anguage

Designed to carry data, not formatting information!

By itself, XML doesn't do anything.
It's a data structure, like a list!

We must write programs to use it.

When do we use XML?

XML is good for situations with complicated data:

- Student
  - Name, age, major, bursar balance, etc.
- Stock quote
  - Symbol, last trade, daily change, high, low, etc.
- Checking account
  - Account number, balance, amount withdrawn, etc.

What happens to our programs if the structure of the data changes?

What does XML look like?

We can tell what this data is:

```
<? xml version="1.0" ?>
<Student>
  <name>
   <first>Jack</first>
   <last>Sparrow</last>
  </name>
  <class>2013</class>
  <id>0019846768</id>
  <credit>10 </credit>
  <fees units="dollars">400</fees>
</Student>
```

It's much denser, but it's easier to parse by a program.

If the order changes or new data is added, we can still make sense of the old data

Structure of an XML Document

This is always the first line

Opening tags
(i.e. element)

Closing tags
(i.e. element)

Opening and closing must match and be in the appropriate order

Attributes (name & value pair)

```
<? xml version="1.0" ?>

<Student>

<name>
 <first>Jack</first>
 <last>Sparrow</last>
</name>

<class>2013</class>

<id>0019846768</id>

<credit>10 </credit>

<fees units="dollars">400</fees>
</Student>
```

Incorrect Structure

No matching closing tag!

Nesting is incorrect!

```
<? xml version="1.0" ?>

<Student>

  <name>
   <first>Jack
   <last>Sparrow
  </name>
  </last>

  <class>2013</class>

  <id>0019846768</id>

  <credit>10 </credit>

  <fees units="dollars">400</fees>
```

Proper XML

Proper XML has two requirements:

- **Well-formed**: Opening and closing tags match, nesting is correct
- **Valid**: The right tags are used in the right places
  - A <name> tag should only appear inside a <Student> tag

Validity changes by document, and we're not going to worry about it, but we do care about the document being well-formed!

Reading XML

Normally, we want to extract the data

Do this in a hierarchical fashion:

1. Get a Student element
   A. Read the first name
   B. Read the last name
   C. Read the class
   D. Read the amount of student fees

Parsing XML

The process of extracting data from an XML document is called **parsing**.

We can use **find()** or built-in XML parsers

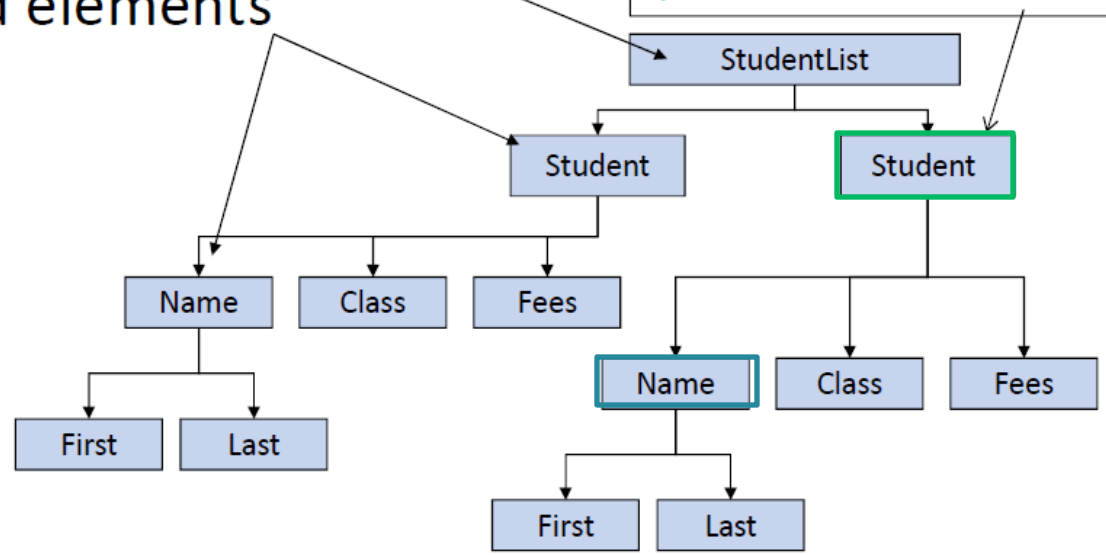
We'll be using a parser called **ElementTree**, which builds a tree structure for the tags.

XML as a Tree

XML data as a tree

- Root of the tree
- Child elements

```
<Student>
  <name>
    <first>Jack</first>
    <last>Sparrow</last>
  </name>
  <class>2013</class>
  <id>0019846768</id>
  <credit>10 </credit>
  <fees units="dollars">400</fees>
</Student>
```



XML

Download **students.xml** from Oncourse and open it in a web browser.

By itself, the data does nothing, but we can open and close the nested tags

This XML file does not appear to have any style information associated with

```
▼<StudentList>
  ▼<Student>
    ▼<name>
       <first>Katie</first>
       <last>Smith</last>
     </name>
     <class>2015</class>
     <credit>12</credit>
     <id>001987283</id>
     <fees units="dollars" c="usa">100</fees>
   </Student>
  ▼<Student>
    ▼<name>
       <first>Jack</first>
       <last>Sparrow</last>
     </name>
     <class>2016</class>
     <id>0019846768</id>
     <credit>10</credit>
     <fees units="dollars" c="usa">200</fees>
   </Student>
  ▼<Student>
    ▼<name>
       <first>Jason</first>
       <last>Bourne</last>
     </name>
     <class>2017</class>
     <id>0019846789</id>
     <credit>16</credit>
     <fees units="dollars" c="usa">400</fees>
   </Student>
  ▼<Student>
    ▼<name>
       <first>Rose</first>
       <last>Dawson</last>
     </name>
     <class>2016</class>
     <id>0019845768</id>
     <credit>9</credit>
     <fees units="dollars" c="usa">300</fees>
   </Student>
 </StudentList>
```

XML and Python

Try this out:

```
import xml.etree.ElementTree as ET
root = ET.parse(source="students.xml")
print root
```

Root is the root of the tree. By default, it will just print out the memory location, like this:

```
<xml.etree.ElementTree.ElementTree instance at
0x7fef092f6d88>
```

XML Parse Tree

Let's get a better look at the XML tree:

```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

elements = root.getiterator()     #creates a list of elements!
for elem in elements:          #prints out the details
    print "Tag Name:", elem.tag
    print "Tag Text:", elem.text
    print "Tag Attributes:", elem.items()
    print "Children:", list(elem)
    print "-"*20
```

An XML Element

Each element corresponds to an XML element

- Starts at the opening tag, ends at the closing tag

```
<Student>
  <name>
    <first>Katie</first>
    <last>Smith</last>
  </name>
  <fees units="dollars">100</fees>
</Student>
```

Four things:

- Tag – the name of the element
- Text – the content of the element (if any)
- Attributes (if any)
- Child elements (if any)

An XML Element

When we hit the fees element:

```
<Student>
  <name>
     <first>Katie</first>
     <last>Smith</last>
  </name>
  <fees units="dollars">100</fees>
</Student>
```

Content: Value:

elem.tag fees

elem.text 100

elem.items() ("units","dollars")

list(elem) No children

XML Parse Tree

We can choose which tags to process:

```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

elements = root.getiterator()
for elem in elements:
    if elem.tag == "name":#should get 4 matches!
        print "Tag Name:", elem.tag
        print "Tag Text:", elem.text
        print "Tag Attributes:", elem.items()
        print "Children:", list(elem)
        print "-"*20
```

XML

Attributes provide useful information about data

- Units of measurement
- Currency type
- Language

Each element **object** has a **method** called **items()**, which returns a list of:

- **Attribute** name, attribute **value** pairs (as tuples!)

XML

Examining attributes:

```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

elements = root.getiterator()
for elem in elements:
    if elem.tag == "fees":
        attributes = list(elem.items())

        print "Attributes:", attributes
        for item in attributes:
            print item[0], item[1]
        print "-"*20
```

XML as a string

If we place an XML document in a string,
we can parse it using the **XML** method!

(not all URLs will return XML as we might like it)

XML as a string

```
import xml.etree.ElementTree as ET

root2 = ET.XML("<top attrib1='val1'><middle>Words!</middle></top>")
print root2

elements = root2.getiterator()
for elem in elements:
    print "Tag Name:", elem.tag
    print "Tag Text:", elem.text
    print "Tag Attributes:", elem.items()
    print "Children:", list(elem)
    print "-"*20
```

XML

Some useful methods:

find(query) – this method returns the first sub-element whose tag matches **query**

findall(query) – returns a list of all sub-elements whose tag matches **query**

query should be an element name or path

XML

Try this:

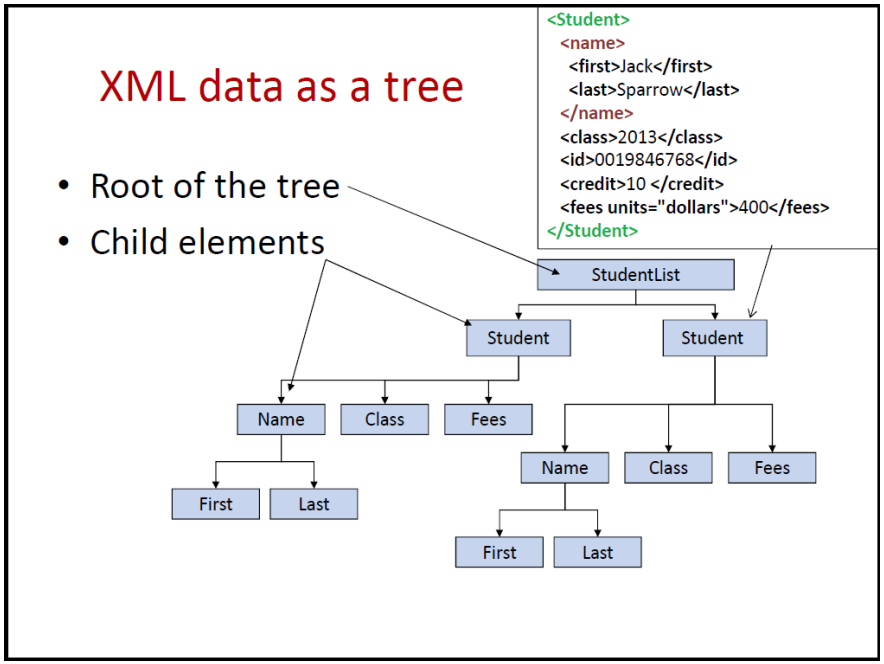
```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

names = root.findall("name")

for name in names:
    print name.text
```

But why doesn't it print out anything?

XML

Remember the tree:

root.findall("name") is trying to find an element called name that's **directly related** to the root!



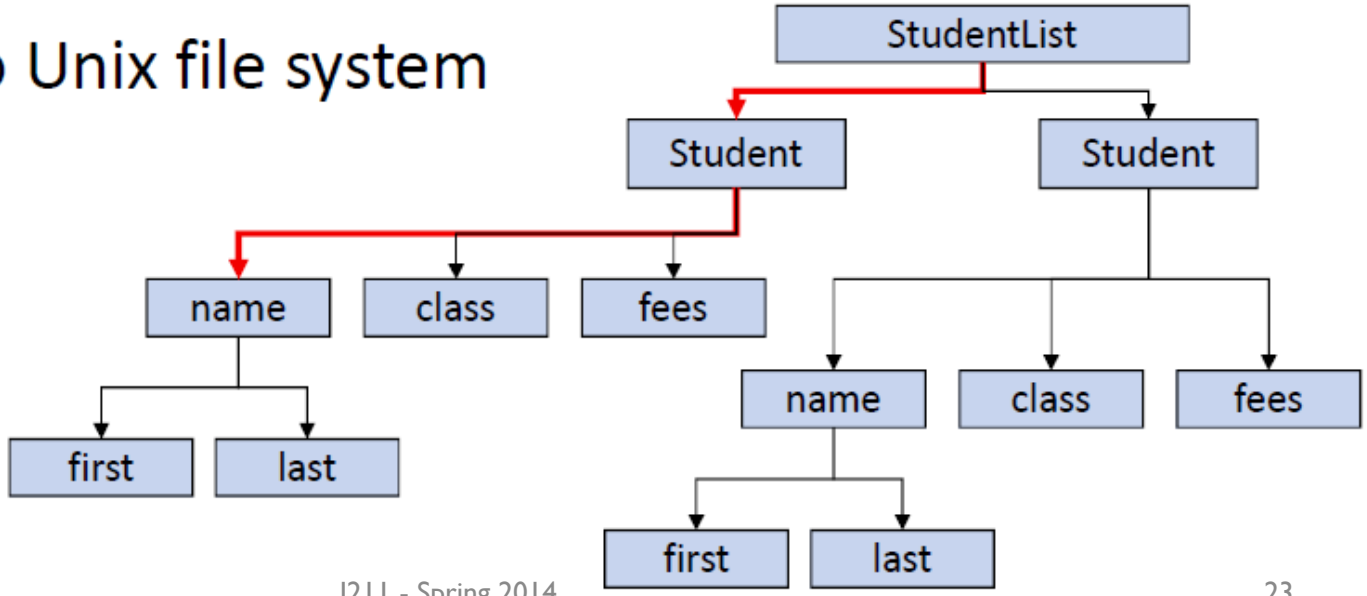
But that's not where the names are:

We need to specify a path to the names.

XML path

- Starting at the root "StudentList", the path to a "name" element is in red

- The path is: **"Student/name"**
  - Similar to Unix file system



XML

So we could get all the first names for example thus:

```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

names = root.findall("Student/name/first")

for elem in names:
    print elem.text
```

XML

find() is handy, but you lose the context of the data!

If that doesn't matter, then use **find()**

Use **findall()** when you need to do something like add up all elements of type X.

If you need the context, use **getiterator()**

Student Info (Group Work)

Write a program to compute and display the following info from the students file:

```
>>>
The unique tags are: ['StudentList', 'Student', 'name', 'first', 'last', 'class', 'credit', 'id', 'fees']

The students are:
Katie Smith
Jack Sparrow
Jason Bourne
Rose Dawson

The total amount of fees is: $ 1000
>>>
```

Student Info (Solution part 1)

```
import xml.etree.ElementTree as ET

root = ET.parse(source="students.xml")

tags = []

elements = root.getiterator()

for elem in elements:
    if elem.tag not in tags:
        tags.append(elem.tag)

print "The unique tags are:", tags, "\n"
```

Student Info (Solution part 2)

```
print "The students are:"

for elem in elements:
    if elem.tag == "Student":
        name = elem.find("name/first").text + " " +
    elem.find("name/last").text
        print name

fees = root.findall("Student/fees")

total = 0
for elem in fees:
    total += int(elem.text)

print "\nThe total amount of fees is: $", total
```

XML Links

XML Tutorial (now cached on archive.org):

http://web.archive.org/web/20120516081149/http://www.javacommerce.com/displaypage.jsp?name=intro.sql&id=18238

Element Tree in Python 2.6:

http://docs.python.org/2.6/library/xml.etree.elementtree.html