XML Processing

XML Parsing

- XML documents use structured markup

```
<contact>
  <name>Elwood Blues</name>
  <address>1060 W Addison</address>
  <city>Chicago</city>
  <zip>60616</zip>
</contact>
```

- Documents made up of elements

```
<name>Elwood Blues</name>
```

- Elements have starting/ending tags
- May contain text and other elements

XML Example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<recipe>
  <title>Famous Guacamole</title>
   <description>
     A southwest favorite!
   </description>
   <ingredients>
       <item num="2">Large avocados, chopped</item>
       <item num="1">Tomato, chopped</item>
       <item num="1/2" units="C">White onion, chopped</item>
       <item num="1" units="tbl">Fresh squeezed lemon juice</item>
       <item num="1">Jalapeno pepper, diced</item>
       <item num="1" units="tbl">Fresh cilantro, minced</item>
       <item num="3" units="tsp">Sea Salt</item>
       <item num="6" units="bottles">Ice-cold beer</item>
   </ingredients>
   <directions>
     Combine all ingredients and hand whisk to desired consistency.
     Serve and enjoy with ice-cold beers.
   </directions>
</recipe>
```

XML Parsing

- XML is a widely used data format
- To parse it, use xml.etree.ElementTree
- This is not the only approach, but it is often considered to be the easiest--especially for simple XML problems

ElementTree Parsing

- Parsing a document

```
from xml.etree.ElementTree import parse
doc = parse("recipe.xml")
```

- Finding one or more elements

```
elem  = doc.find("title")
for elem in doc.findall("ingredients/item"):
    statements
```

- Element attributes and properties

```
elem.tag                      # Element name
elem.text                     # Element text
elem.get(aname [,default])    # Element attributes
```

Obtaining Elements

```
<?xml version="1.0" encoding="iso-8859-1"?>
<recipe>
  <title>Famous Guacamole</title>
  <description>
    A southwest favorite!
  </description>
  <ingredients>
      <item num="2">La
      <item num="1">To
      <item num="1/2"
      <item num="1" ur
      <item num="1">Ja
      <item num="1" ur
      <item num="3" ur
      <item num="6" ur
  </ingredients>
  <directions>
    Combine all ingredients and hand whisk to desired consistency.
    Serve and enjoy with ice-cold beers.
  </directions>
</recipe>
```

```
doc = parse("recipe.xml")
desc_elem = doc.find("description")
desc_text = desc_elem.text
```

or

```
doc = parse("recipe.xml")
desc_text = doc.findtext("description")
```

Iterating over Elements

```
<?xml version="1.
<recipe>
  <title>Famous G
   <description>
     A southwest
   </description>
   <ingredients>
      <item num="2">Large avocados, chopped</item>
      <item num="1">Tomato, chopped</item>
      <item num="1/2" units="C">White onion, chopped</item>
      <item num="1" units="tbl">Fresh squeezed lemon juice</item>
      <item num="1">Jalapeno pepper, diced</item>
      <item num="1" units="tbl">Fresh cilantro, minced</item>
      <item num="3" units="tsp">Sea Salt</item>
      <item num="6" units="bottles">Ice-cold beer</item>
   </ingredients>
   <directions>
     Combine all ingredients and hand whisk to desired consistency.
     Serve and enjoy with ice-cold beers.
   </directions>
</recipe>
```

```
doc = parse("recipe.xml")
for item in doc.findall("ingredients/item"):
      statements
```

Element Attributes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<recipe>
  <title>Famous Guacamole</title>
   <description>
     A southwest favorite!
   </description>
   <ingredients>
```

```
for item in doc.findall("ingredients/item"):
    num   = item.get("num")
    units = item.get("units")
```

```
       <item num="1" units="tbl">Fresh cilantro, minced</item>
       <item num="3" units="tsp">Sea Salt</item>
       <item num="6" units="bottles">Ice-cold beer</item>
   </ingredients>
   <directions>
     Combine all ingredients and hand whisk to desired consistency.
     Serve and enjoy with ice-cold beers.
   </directions>
</recipe>
```

cElementTree

- There is a C implementation of the library that is significantly faster

```
import xml.etree.cElementTree
doc = xml.etree.cElementTree.parse("data.xml")
```

- For all practical purposes, you should use this version of the library given a choice
- Note : The C version lacks a few advanced customization features, but you probably won't need them

Advanced XML

- A discussion of additional XML topics
  - Advanced searching
  - XML namespaces
  - Document tree modification
  - Incremental parsing
  - Alternative packages

Search Wildcards

- Specifying a wildcard for an element name

```
items = doc.findall("*/item")
items = doc.findall("ingredients/*")
```

- The * wildcard only matches a <u>single</u> element
- Use multiple wildcards for nesting

```
<?xml version="1.0"?>
<top>
    <a>
       <b>
          <c>text</c>
       </b>
    </a>
</top>
```

```
c = doc.findall("*/*/c")
c = doc.findall("a/*/c")
c = doc.findall("*/b/c")
```

Search Wildcards

- Wildcard for multiple nesting levels (//)

```
items = doc.findall("//item")
```

- More examples

```
<?xml version="1.0"?>
<top>
    <a>
        <b>
            <c>text</c>
        </b>
    </a>
</top>
```

```
c = doc.findall("//c")
c = doc.findall("a//c")
```

XML Namespaces

- Example of a namespace specification

```
<doc xmlns:html="http://www.w3.org/1999/xhtml">
   <html:head>
       <html:title>Hello World</html:title>
   </html:head>
   <html:body>
        This is the body
        ...
   </html:body>
</doc>
```

- Use fully expanded namespace in queries

```
title = doc.findtext("{http://www.w3.org/1999/xhtml}title")
```

XML Namespaces

- Namespace suggestion: Use a dictionary

```
<doc xmlns:html="http://www.w3.org/1999/xhtml">
...

ns = {
  "html": "http://www.w3.org/1999/xhtml"
}

title = doc.findtext("{%(html)s}title" % ns)
```

- Reduces the amount of typing and makes it easier to make changes later (if needed)

Tree Modification

- ElementTree allows modifications to be made to the document structure
- To add a new child to a parent node

```
node.append(child)
```

- To insert a new child at a selected position

```
node.insert(index,child)
```

- To remove a child from a parent node

```
node.remove(child)
```

Tree Output

- If you modify a document, it can be rewritten
- There is a method to write XML

```
doc = xml.etree.ElementTree.parse("input.xml")
# Make modifications to doc
...
# Write modified document back to a file
f = open("output.xml","w")
doc.write(f)
```

- Individual elements can be turned into strings

```
s = xml.etree.ElementTree.tostring(node)
```

Iterative Parsing

- An alternative parsing interface

```
from xml.etree.ElementTree import iterparse
parse = iterparse("file.xml", ('start','end'))

for event, elem in parse:
    if event == 'start':
         # Encountered an start <tag ...>
         ...
    elif event == 'end':
         # Encountered an end </tag>
         ...
```

- This sweeps over an entire XML document
- Result is a sequence of start/end events and element objects being processed

Iterative Parsing

- If you combine iterative parsing and tree modification together, you can process large XML documents with almost no memory overhead
- General idea : Simply throw away the elements no longer needed during parsing

Iterative Parsing

- Programming pattern

```
from xml.etree.ElementTree import iterparse
parser = iterparse("file.xml",('start','end'))

for event,elem in parser:
    if event == 'start':
        if elem.tag == 'parenttag':
            parent = elem
    if event == 'end':
        if elem.tag == 'tagname':
            # process element with tag 'tagname'
            ...
            # Discard the element when done
            parent.remove(elem)
```

- The last step is the critical part

Alternative Libraries

- Also included in Python standard library
  - SAX parsing
  - DOM parsing
  - Both modeled after equivalent in Java

XML Parsing with SAX

- An event-driven parsing approach
- Very low-level, but sometimes used when dealing with very large XML documents

SAX Parsing

- Define a special handler class

```
import xml.sax

class MyHandler(xml.sax.ContentHandler):
    def startDocument(self):
        print "Document start"
    def startElement(self,name,attrs):
        print "Start:", name
    def characters(self,text):
        print "Characters:", text
    def endElement(self,name):
        print "End:", name
```

- In the class, you define methods that capture elements and other parts of the document

SAX Parsing

- To parse a document, you create an instance of the handler and give it to the parser

```
# Create the handler object
hand = MyHandler()

# Parse a document using the handler
xml.sax.parse("data.xml",hand)
```

- This reads the file and calls handler methods as different document elements are encountered (start tags, text, end tags, etc.)

SAX Example

- Print out the ingredient list from a recipe

```
class MyHandler(xml.sax.ContentHandler):
    def startDocument(self):
        self.initem = False
    def startElement(self,name,attrs):
        if name == 'item':
            self.num = attrs.get('num','1')
            self.units = attrs.get('units','none')
            self.text = []
            self.initem = True
    def endElement(self,name):
        if name == 'item':
            text = "".join(self.text)
            if self.units == 'none': self.units = ""
            unitstr = "%s %s" % (self.num, self.units)
            print "%-10s %s" % (unitstr,text.strip())
            self.initem = False
    def characters(self,data):
        if self.initem:
            self.text.append(data)
```

DOM Parsing

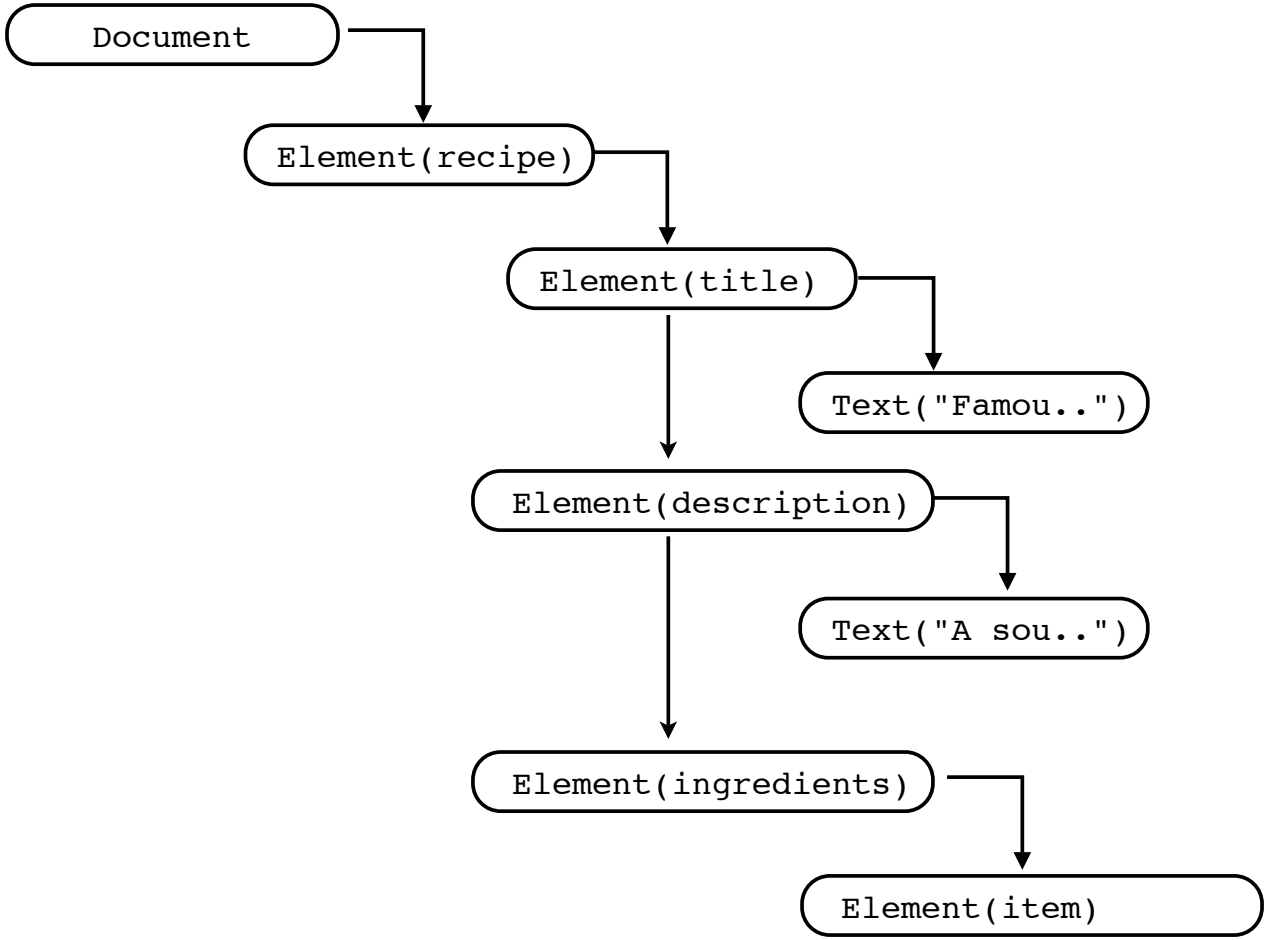
- Parses an XML document into a tree structure
- xml.dom.minidom module

```
>>> import xml.dom.minidom
>>> f = open("recipe.xml","r")
>>> doc = xml.dom.minidom.parse(f)
>>> doc
<xml.dom.minidom.Document instance at 0x585f8>
>>>
```

- Returned object holds the parsed XML data
- Useful for random access
- Or if you want to restructure the document

DOM Trees

- Document is stored as a tree



Example

- Print out an ingredient list for a recipe

```
import xml.dom.minidom
doc = xml.dom.minidom.parse("recipe.xml")

ingr = doc.getElementsByTagName("ingredients")[0]
items = ingr.getElementsByTagName("item")
for i in items:
    num = i.getAttribute("num")
    units = i.getAttribute("units")
    text = i.firstChild.data.strip()
    quantity = "%s %s" % (num,units)
    print "%-10s %s" % (quantity,text)
```

Final Comments

- Standard Python doesn't support advanced XML
  - Validation
  - Xinclude
  - XPath
  - XSLT
  - XML Schema
- Look at third party libs (start with lxml)