

# XML

Week 4, day 4

# Extensible markup Language

- Structured markup
- Looks like but isn't HTML
- For containing data
- Non-executable - purely to be read and written

# Example

```
<SHOPPING_LIST>  
  <TO_GET>  
    <ITEM>Eggs</ITEM>  
    <ITEM>Milk</ITEM>  
    <ITEM>Bread</ITEM>  
  </TO_GET>  
  <DONT_NEED>  
    <ITEM>Tacos</ITEM>  
    <ITEM>Onions</ITEM>  
  </DONT_NEED>  
</SHOPPING_LIST>
```

# Structure

- A series of markup tags <in angled brackets>
- Tags are paired with closing tags
  - `<TAG> . . . </TAG>`
- Thus tags may contain other tags
- Tags can occur multiple times
- There are no standard tags

# Why


- Correctly formatted XML can be parsed without knowing the tags or their structure
- Can use generic tools to read & write a range of data - syntax is not semantics
- Data is portable across systems and languages
- “Solves” the data format & storage problem
- Extensible, customisable

# The rules by example: phylogeny


```
<PHYDATA>  
</PHYDATA>
```



```
<PHYDATA>  
  <PHYLOGENY>  
  </PHYLOGENY>  
</PHYDATA>
```



```
<PHYDATA>  
  <PHYDATA>  
  </PHYDATA>  
</PHYDATA>
```



1. A document has at least one **element**
2. The outer element is the **root**. All other elements must appear within it. It may not repeat or appear elsewhere.

# The rules by example 2

```
<PHYDATA>  
</PHYDATA> ✓
```

```
<PHYDATA>  
  <PHYLOGENY>  
  </PHYLOGENY>  
</Phydata> ✗
```

```
<TAXA>  
<NAME>  
</TAXA>  
</NAME> ✗
```

3. Start-tags and end-tags match. Case matters
4. If a start tag is “inside” another, so must the end tag be. No overlapping.

# The rules by example 3

```
<PHYDATA>  
</PHYDATA> ✓
```

```
<PHYDATA/> ✓
```

```
<TAXA>  
<NAME>  
</NAME> ✗
```

5. Start-tags must have an end-tags `/TAGNAME` if they “contain” other tags.

A tag without content can end with a trailing slash instead of the matching end tag.

# The rules by example 4

```
<PHYDATA>  
</PHYDATA> ✓
```

```
<PHY_DATA1/> ✓
```

```
<TAXA TITLE>  
</TAXA TITLE> ✗
```

6. Legal tag names contain letters, numbers, hyphens, underscores, periods. Names beginning with “XML” (any case) are reserved.

# The rules by example 5

```
<PHYDATA class="">  
</PHYDATA>
```



```
<PHY_DATA1  
class="extended" />
```



```
<TAXA  
style&name=linnean>  
</TAXA>
```



7. Tags may have **attributes**, properties given in the starting tag of the form `attribname="value"`. Single or double quotes are okay. Same name rules apply

# The rules by example 6

```
<PHYDATA>  
A sample file.  
</PHYDATA>
```



```
<TAXA>  
Hyaena & Crocuta  
</TAXA>
```



```
<TAXA>  
Hyaena & Crocuta  
</TAXA>
```



8. Text is a valid tag

9. But text cannot contain the characters `<` and `&`. They have to be escaped and written as `&lt;` and `&amp;`.

10. Whitespace is text too. Darn.

# The rules by example 7

```
<PHYDATA>  
<![CDATA[ &&&<<< ] ]>  
</PHYDATA>
```



11. **CDATA** can capture literal text

```
<?xml version="1.0  
encoding="utf-8"?>  
<PHYDATA>  
</PHYDATA>
```



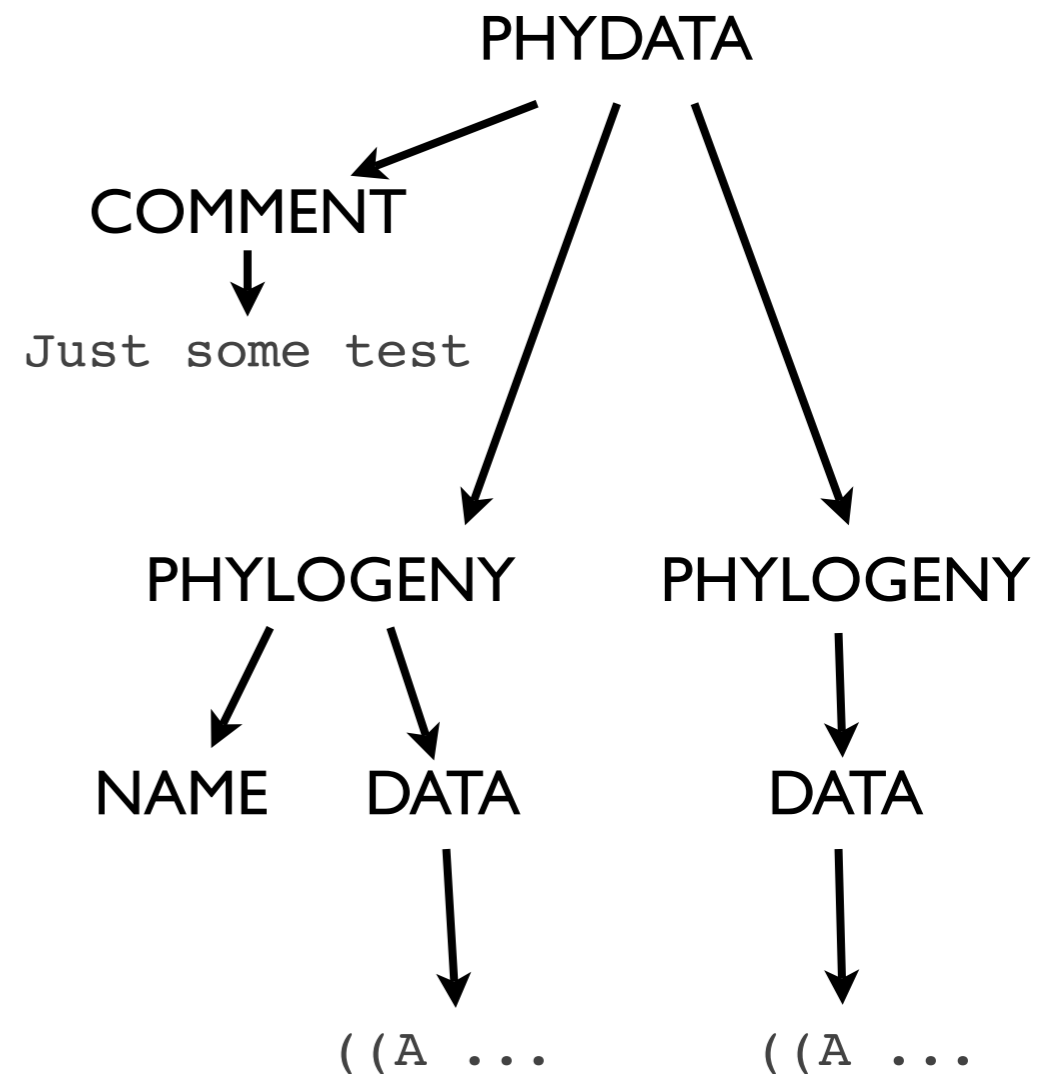
12. A opening “tag” giving info about XML is normal

# The upshot

- A **well-formed** XML document is an ordered labelled tree
- A tree of **nodes** that are:
  - **character data** (text) ...
  - or **elements** with a **type** or **name** and labels or **attributes**
- We can traverse a tree in order

# The tree

```
<PHYDATA>
<COMMENT>Just some test
data</COMMENT>
<PHYLOGENY>
<NAME />
<DATA format="newick">
((A,B), (C,D))</DATA>
</PHYLOGENY>
<PHYLOGENY>
<DATA format="ntx">
((A,B,C), D)</DATA>
</PHYLOGENY>
</PHYDATA>
```



# Parsing

- SAX
  - Simple API for XML
  - Element by element as it is read
- DOM
  - Document Object Model
  - Read whole document and then process

# XML & Python

- `import xml`
- `PyXML`
  - `and Elementtree / cElementree ...`
  - `pyfo ...`
  - `PyXSLT ...`
  - `etcetera etcetera ....`

# Creating an XML document

```
import xml.dom.minidom

doc = Document()

root = doc.createElementNS ("http://
nbn.ac.za", "PHYDATA")
doc.appendChild (root)

c = doc.createComment ("An example file")
t = doc.createTextNode ("This is text data")
root.appendChild (t)
root.appendChild (c)
```

# Creating an XML document 2

```
[...]
```

```
newElem = doc.createElementNS ( "", "PHYDATA" )  
root.appendChild (newElem)
```

```
newElem.setAttributeNS( "", "name", "Hyrax" )
```

# The calls

- `import xml.dom.minidom`
- `Document()`
- `doc.createElementNS (namespace, tag)`
- `elem.appendChild (node)`
- `doc.createComment (str)`
- `doc.createTextNode (str)`

# Writing it out

```
# print to screen in nice format
import xml.dom.ext
xml.dom.ext.PrettyPrint (doc)

# to file
outfile = open ("out.xml")
xml.dom.ext.PrettyPrint (doc, outfile)
outfile.close()

# unpretty print
outfile = open ("out2.xml")
xml.dom.ext.Print (doc, outfile)
outfile.close()
```

# Why not

- Verbose
- Hard for human to read
- Some things are hard to express in tags (mathematical notation)
- Binary data
- Hard for human to write
- Doesn't solve the data format & storage problem

# Resources

- Python XML & HTML tutorials at <<http://awaretek.com/tutorials.html#html>>
- PyXML at SourceForge <<http://sourceforge.net/projects/pyxml/>>
- Python & XML <[http://www.boddie.org.uk/python/XML\\_intro.html](http://www.boddie.org.uk/python/XML_intro.html)>