



The Skinny¹ on Python and The eXtensible Markup Language

Brian A. Malloy
Computer Science Department

[1] Inside information; the real facts.



Slide 1 of 100

Go Back

Full Screen

Quit

1. Python and XML

- Scripting Language
- Full-scale Programming Language



Slide 2 of 100

Go Back

Full Screen

Quit

1.1. XML

- Not a markup language – a toolkit for markup languages
- XML will replace HTML and be absorbed into XHTML
- Can create new markup languages
- Works with stylesheets
- Unlike HTML browsers, XML parsers are supposed to throw an exception and stop when an XML document is broken. They are not permitted to guess what the XML document structure should be.



Slide 3 of 100

Go Back

Full Screen

Quit

1.2. Python/XML

- Excellent file processing
- Excellent OO
- Portable
- Robust rather than efficient



Slide 4 of 100

Go Back

Full Screen

Quit



1.3. Advantages of XML

- Human and machine readable
- Application & Language independent
- Platform independent
- Hierarchical
- User-defined tag names



Slide 5 of 100

Go Back

Full Screen

Quit

1.4. Python tools for XML

- Python standard lib: Expat for SAX & DOM
- PyXML – supports XPath and XSLT
- 4Suite – supports XSL transformations w/in Python



Slide 6 of 100

Go Back

Full Screen

Quit



1.5. XML Documents

- Comments: `<!-- comment -- >`
- Marks up data using tags: names in angle brackets.
The following XML document is well-formed:

```
<?xml version="1.0"?>
<greeting>
  Hello World
</greeting>
```

- The following XML document is not well-formed:

```
<?xml version="1.0"?>
<greeting> Hello World </greeting>
<greeting> Good Bye </greeting>
```



Slide 7 of 100

Go Back

Full Screen

Quit



1.6. Some XML Rules

- All elements must be properly nested
- All attributes must be quoted
- All tags are case sensitive
- All end-tags are required
- Empty tags can contain the end marker:
`<child age="6"/>`
- Can use DTDs and schemas to define additional constraints on an XML document.
- Well-formed vs valid:
 - Well-formed: follows rules for XML
 - Valid: an XML document that follows the rules specified in a DTD or schema



Slide 8 of 100

Go Back

Full Screen

Quit



1.7. XML Namespaces

- Python, C++ and XML have massive class libraries
- Namespaces distinguish tags with the same name:

```
<jackson:name>  
    The Lord of the Rings  
</jackson:name>
```



Slide 9 of 100

Go Back

Full Screen

Quit



1.8. Data in XML

- Can be placed between tags
- Can be placed in attributes

```
<letter>  
  <contact type="from">  
    <name>J. R. R. Tolkien</name>  
  </contact>  
</letter>
```

- Tags vs Elements:
 - Tag: text between angle brackets
 - Element: start tag, end tag and everything in-between.



Slide 10 of 100

Go Back

Full Screen

Quit



1.9. Empty elements

- Empty elements do not contain data between open/close element tags

```
<flag gender="M" />
```

```
<flag gender="F" > <flag/>
```



Slide 11 of 100

Go Back

Full Screen

Quit



1.10. Document Object Model – DOM

- Some XML parsers store data as tree – DOM
- Easy to modify data and write back
- DOM has single root: document root
- Each XML component stored as object/node in tree
- Each node: attributes & methods
- DOM must be stored in memory



Slide 12 of 100

Go Back

Full Screen

Quit



1.11. Simple API for XML

- Uses event-based model
- Generate notifications called events
- Software listens for these events
- Faster & less memory than DOM
- If data won't be modified – use SAX



Slide 13 of 100

Go Back

Full Screen

Quit



2. XSL Transformations

- Extensible Stylesheet Language Transformation (XSLT)
- Can transform XML into anything:
 - Another XML document
 - HTML
 - Latex
 - RTF
 - PDF
 - SVG
 - VRML
- Uses stylesheet to do what SAX does, only easier



Slide 14 of 100

Go Back

Full Screen

Quit



2.1. XSLT Specification

- Available from: <http://www.w3.org/TR/sxlt.html>
- Need 3 files and 1 app



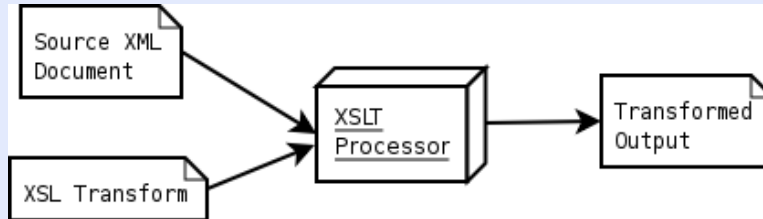
Slide 15 of 100

Go Back

Full Screen

Quit

2.2. Overview of transformation process



Slide 16 of 100

Go Back

Full Screen

Quit



2.3. XSLT Processors

- XSLT is a language, has nothing in particular to do with Python
- However, can use XSLT from within Python: 4XSLT



Slide 17 of 100

Go Back

Full Screen

Quit

2.4. 4XSLT Processor

- Open source
- Can be driven from command-line, or w/in Python
- Written in Python, w/ some C for performance
- Available from Fourthought, INC as part of 4Suite package <http://www.4suite.org/>



Slide 18 of 100

Go Back

Full Screen

Quit



2.5. Stylesheets

- XSLT stylesheets similar to CSS
- Three ways to use CSS (XSLT):
 - Simplified Stylesheet: style sheet included in document to be transformed.
 - Standalone: style sheet stored in a separate file
 - Embedded Stylesheet: embedded as special element in HTML specific styling info attached to elements



Slide 19 of 100

Go Back

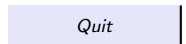
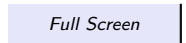
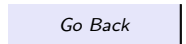
Full Screen

Quit



3. Simplified Stylesheets

- Similar to the Style attribute in HTML, but XSLT is much more powerful
- Some XSLT functionality is not permitted in simplified stylesheets
- Useful for queries on an XML document





3.1. Consider the Star Trek Spaceships example

- Instead of using DOM and XPath, we'll use a simplified XSLT
- We'll create a list of ships, sorted by registry number, formatted as an HTML table



Slide 21 of 100

Go Back

Full Screen

Quit



3.2. Example of Simplified Stylesheet

```
1 <html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
2     xsl:version="1.0">
3 <head>
4   <title>Ships of the
5     <xsl:value-of select="/shiptypes/@name" />
6 </title>
7 </head>
8 <body>
9   <table border="1">
10    <tr><th>Ship</th>
11      <th>Class</th>
12      <th>Registration</th>
13      <th>Captain</th>
14    </tr>
15    <xsl:for-each select="/shiptypes/ship">
16      <xsl:sort select="registry-code" />
17      <tr><td><xsl:value-of select="@name" /></td>
```



Slide 22 of 100

Go Back

Full Screen

Quit



```
18     <td><xsl:value-of select="class" /></td>
19     <td><xsl:value-of select="registry-code" /></td>
20     <td><xsl:value-of select="captain" /></td>
21 </tr>
22 </xsl:for-each>
23 </table>
24 </body>
25 </html>
```



Slide 23 of 100

Go Back

Full Screen

Quit

3.3. Result of Simplified Transform



The screenshot shows a Mozilla browser window titled "Ships of the United Federation of Planets - Mozilla". The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. Below the menu bar are navigation buttons (back, forward, home, stop) and a search box. The main content area displays a table with the following data:

Ship	Class	Registration	Captain
USS Enterprise	Constitution	NCC-1701	James T. Kirk
USS Enterprise	Galaxy	NCC-1701-D	Jean-Luc Picard
USS Enterprise		NCC-1701-D	
USS Enterprise	Sovereign	NCC-1701-E	Jean-Luc Picard
USS Voyager	Intrepid	NCC-74656	Kathryn Janeway
USS Sao Paulo	Defiant	NCC-75633	Benjamin L. Sisko



Slide 24 of 100

Go Back

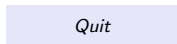
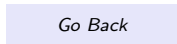
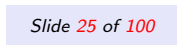
Full Screen

Quit



4. Standalone Stylesheets

- Stylesheets stored in separate file
- Most common usage of XSLT
- Offer more power and flexibility than simplified stylesheets
- Better modularization
- the next example was derived by:
4xslt ships.xml ships.xsl > ships.html





4.1. Example of Standalone stylesheet

```
1 <xsl:stylesheet >
3 <xsl:template match="/">
4 <html>
5 <head>
6 <title>Ships of the
7     <xsl:apply-templates mode="head" />
8 </title>
9 </head>
10 <body>
11 <xsl:apply-templates />
12 </body>
13 </html>
14 </xsl:template>
15
16 <xsl:template match="shiptypes" mode="head">
17 <xsl:value-of select="@name" />
18 </xsl:template>
```



Slide 26 of 100

Go Back

Full Screen

Quit



```
20 <xsl:template match="shiptypes">
21   <table border="1">
22     <tr><th>Ship</th>
23       <th>Class</th>
24       <th>Registration</th>
25       <th>Captain</th>
26   </tr>
27   <xsl:apply-templates />
28 </table>
29 </xsl:template>
30
31 <xsl:template match="ship">
32   <tr><td><xsl:value-of select="@name" /></td>
33     <td><xsl:value-of select="class" /></td>
34     <td><xsl:value-of select="registry-code" /></td>
35     <td><xsl:value-of select="captain" /></td>
36   </tr>
37 </xsl:template>
39 </xsl:stylesheet>
```



Slide 27 of 100

Go Back

Full Screen

Quit

4.2. Result of Standalone Transform



The screenshot shows a Mozilla browser window titled "Ships of the United Federation of Planets - Mozilla". The browser's address bar is empty, and the menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. The toolbar contains navigation buttons (back, forward, home, stop), a search box, and a print button. The main content area displays a table with the following data:

Ship	Class	Registration	Captain
USS Enterprise	Sovereign	NCC-1701-E	Jean-Luc Picard
USS Voyager	Intrepid	NCC-74656	Kathryn Janeway
USS Enterprise	Galaxy	NCC-1701-D	Jean-Luc Picard
USS Enterprise	Constitution	NCC-1701	James T. Kirk
USS Sao Paulo	Defiant	NCC-75633	Benjamin L. Sisko

The browser's status bar at the bottom shows the system tray with icons for network, volume, and power, and the text "Done".



Slide 28 of 100

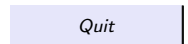
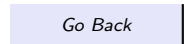
Go Back

Full Screen

Quit

5. The XSLT Elements

- Stylesheet element
- Template element
- Applying templates
- Getting the value of a node (value-of and select)
- Iteration over elements



5.1. The Stylesheet element

- Always the root element of a standalone stylesheet
- Contains some mandatory and some optional attributes
- The id attribute is optional: useful when collecting several stylesheets together into a larger composite document
- the version attribute is required
- namespace prefix optional, but recommended
- Important to specify the format of the output:
html, text, xml (default)
`<xsl:output method="xml" />`



Slide 30 of 100

Go Back

Full Screen

Quit



5.2. The Template Element

- Like a function, `xsl:template` does the work in the transformation process.
- It's an `xslt` instruction, usually specifies a pattern for its invocation, or specifies a name so that it can be called by other parts of the XSL document.
- Body of `xsl:template` element contains output markup when template is called or matched.
- The attributes of the template element define, optionally, its name and matching rule (`match`).
- In addition, `mode` and `priority` are also available.
- `mode` indicates a namespace prefix to be considered when the `xsl:apply-templates` instructions is used.
- `priority` is used when a collection of templates match the same pattern.



Slide 31 of 100

Go Back

Full Screen

Quit



5.3. Template Element: match

- Most important attribute
- Contains an XPath expression used to determine when the processor has hit the target element in the source XML document.
- For example:
`<xsl:template match="/addr-record/address1">`
starts at root `/addr-record` and then selects child `address1`.
- The `xsl:apply-templates` allows processing of nested rules within each other to produce deeply nested documents that are transformed as expected.



Slide 32 of 100

Go Back

Full Screen

Quit



5.4. Example of Nesting: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<deep-nest>
  <title>Sample Text</title>
  <big>T</big>his is an example of
  <red>Fancy Text</red> that comes in
  <blue>m<big>u</big></blue><green>l<big>
t</big></green><blue>i<big>p</big>
</blue><green>l<big>e</big></green>
  colors. Many of <bold>these</bold>
  elements are <big><green>N</green>
  <blue>E</blue><green>S</green><blue>T</blue>
  <green>E</green><blue>D</blue></big> within
  each other.
</deep-nest>
```



Slide 33 of 100

Go Back

Full Screen

Quit



5.5. Example of Nesting: XSLT

```
<xsl:stylesheet>

<xsl:output method="html"/>

<xsl:template match="deep-nest">
  <html><body><xsl:apply-templates/></body></html>
</xsl:template>

<xsl:template match="title">
  <h1> <xsl:apply-templates/> </h1>
</xsl:template>

<xsl:template match="big">
  <font size="5"><xsl:apply-templates/></font>
</xsl:template>

<xsl:template match="red">
```



Slide 34 of 100

Go Back

Full Screen

Quit

```
<font size="3" color="#FF0000">
  <u><xsl:apply-templates/></u>
</font>
</xsl:template>
<xsl:template match="blue">
  <font color="#0000FF">
    <xsl:apply-templates/>
  </font>
</xsl:template>
<xsl:template match="green">
  <font color="#00FF00">
    <b><xsl:apply-templates/></b>
  </font>
</xsl:template>
<xsl:template match="bold">
  <b><i><xsl:apply-templates/></i></b>
</xsl:template>
</xsl:stylesheet>
```



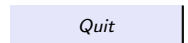
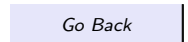
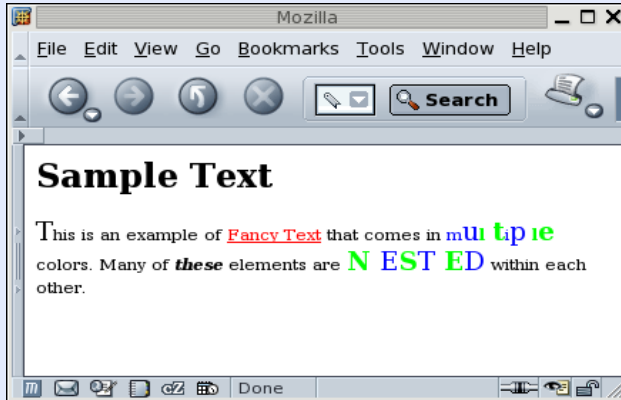
Slide 35 of 100

Go Back

Full Screen

Quit

5.6. Result of Nested Templates



5.7. Getting the value of a node

- The `xsl:value-of` element generates output from an expression.
- Has 2 possible attributes: `select` and `disable-output-escaping`
- `select` is mandatory and takes an XPath expression
- The `disable-output-escaping` causes XSLT processor to suppress encoding of characters that could be confused with markup.



Slide 37 of 100

Go Back

Full Screen

Quit



5.8. Example of value-of/select: XML

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <a>
3   <b>
4     <c id="an_attribute">
5       This is a sentence
6     <doc>A &amp; B</doc>
7   </c>
8 </b>
9 </a>
***** Results *****
$ 4xslt example.xml style.xsl
an_attributeA & B
```



Slide 38 of 100

Go Back

Full Screen

Quit



5.9. Example of value-of/select: XSLT

```
<?xml version="1.0"?>
<xsl:stylesheet>
<xsl:output method="text" indent="no"
  encoding="ISO-8859-1"/>
<xsl:strip-space elements="*/>
<xsl:template match="/">
  <xsl:value-of select="a/b/c/@id" />
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="c">
  <xsl:apply-templates select="doc"/>
</xsl:template>
<xsl:template match="doc">
  <xsl:value-of select="text()"
    disable-output-escaping="yes" />
</xsl:template>
</xsl:stylesheet>
```



Slide 39 of 100

Go Back

Full Screen

Quit



5.10. Iteration over Elements

- `xsl:for-each` allows iteration over elements
- Has a mandatory `select` attribute that defines the node set to be iterated over.
- `select` can contain anything that results in a collection of elements or nodes, can be as simple as an element name or another path expression.
- Helpful when working with mixed content.



Slide 40 of 100

Go Back

Full Screen

Quit



5.11. Example of iteration: products xml

```
<?xml version="1.0"?>
<purchases>
  <product name ="floppy disk" price="3.50"/>
  <product name ="web updates" price="6.95"/>
  <product name ="ink-jet cartridge" price="19.95"/>
  <product name ="consulting" price="150h"/>
</purchases>
```



Slide 41 of 100

Go Back

Full Screen

Quit



5.12. Iteration: the style sheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0">
<xsl:output method="html"/>
<xsl:template match="/">
  <html>
    <body><xsl:apply-templates/></body>
  </html>
</xsl:template>
<xsl:template match="purchases">
  <xsl:for-each select="product">
    <p>
      Product: <xsl:value-of select="./@name"/>
      Price: <xsl:value-of select="./@price"/>
    </p>
  </xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```



Slide 42 of 100

Go Back

Full Screen

Quit



5.13. Iteration: HTML output

```
<html>
  <body>
    <p> Product: floppy disk
      Price: 3.50
    </p>
    <p> Product: web updates
      Price: 6.95
    </p>
    <p> Product: ink-jet cartridge
      Price: 19.95
    </p>
    <p> Product: consulting
      Price: 150h
    </p>
  </body>
</html>
```



Slide 43 of 100

Go Back

Full Screen

Quit



6. Embedding XSLT in Python

- XML is frequently used to store the "core" version of a document while transformations are used to integrate the data into other systems.
- Ability to transform XML at runtime is key to its versatility
- Uses Uniform Resource Identifiers (URI), the generic term for all types of names and addresses that refer to objects on the World Wide Web. A URL is one kind of URI.
- 4xslt package works from within Python:
<http://4suite.org/docs/CoreManual.xml>
- XML and Python are frequently used with CGI



Slide 44 of 100

Go Back

Full Screen

Quit



6.1. XSL transformations w/in Python

```
from Ft.Xml.Xslt import Processor
# We use the InputSource architecture
from Ft.Xml import InputSource
# path to URI conversions
from Ft.Lib.Uri import OsPathToUri

processor = Processor.Processor()

# Prepare an InputSource for the source document
# Convert from local file to uri
srcAsUri = OsPathToUri('story.xml')
source = InputSource.DefaultFactory.fromUri(srcAsUri)

# Prepare an InputSource for the stylesheet
# Convert from local file to uri
ssAsUri = OsPathToUri('story.xsl')
transform = InputSource.DefaultFactory.fromUri(ssAsUri)
```



Slide 45 of 100

Go Back

Full Screen

Quit



```
processor.appendStylesheet(transform)
result = processor.run(source)
```

```
# result is a string representation of
# the serialized transformation
print result
```



Slide 46 of 100

Go Back

Full Screen

Quit

6.2.

-



Slide 47 of 100

Go Back

Full Screen

Quit

Contents



Slide 48 of 100

Go Back

Full Screen

Quit