XML

eXtensible Markup Language

What is XML?

- A *markup language* much like HTML
- Designed to *describe data*
- Tags not predefined. *Define your own tags*
- Uses a Document Type Definition (DTD) or an XML Schema to describe the data
- Designed to be *self-descriptive*

XML does not DO anything

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Lunch this weekend!</body>
</note>
```

You must write software to send, receive or display it.

Benefits of XML

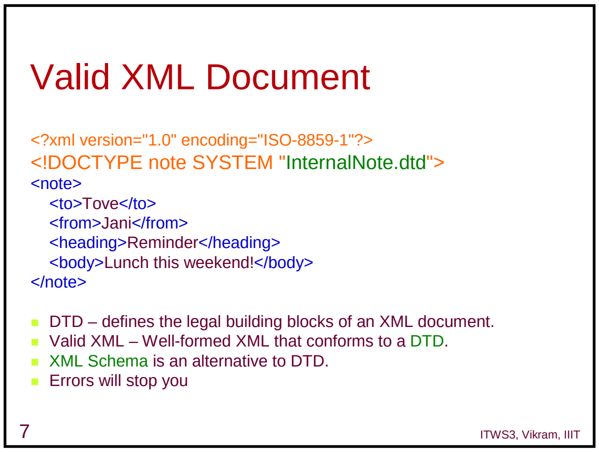
- Separate data from HTML
- Exchange data between incompatible systems
  - Plain text files
  - Between Word processors in different OS's
  - Between PDAs, Computers, Wireless devices
- Used to create new languages (e.g. WML)

Well-formed XML Document

- Document must have a root element
- Specify empty tags as empty
  - <p> </p>
  - <img src="file.jpg"/>
- Tags are case-sensitive (<Note> </Note>)
- Must be properly nested
  - <b><i> text </b></i> – wrong
- Attribute values must always be quoted
  - <table border="1">
- White space is preserved
  - Hello   this is xml data,
- <!-- This is a comment -->

Elements versus Attributes

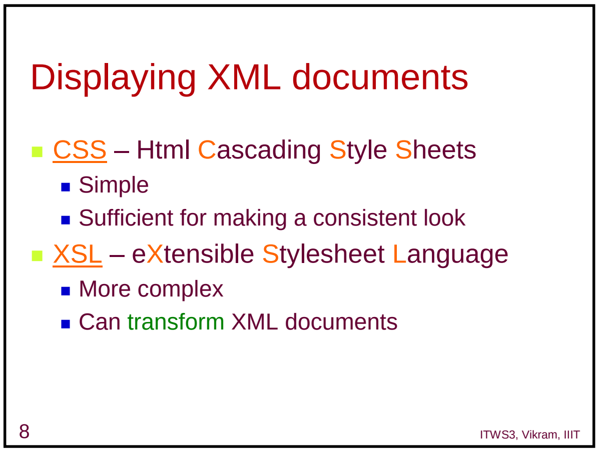
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note date="12/11/2002">
        <date>12/11/2002</date>
        <date>
                <day>12</day>
                <month>11</month>
                <year>2002</year>
        </date>
        <to>Tove</to>
        <from>Jani</from>
        <body>Lunch this weekend!</body>
</note>
```



Valid XML Document

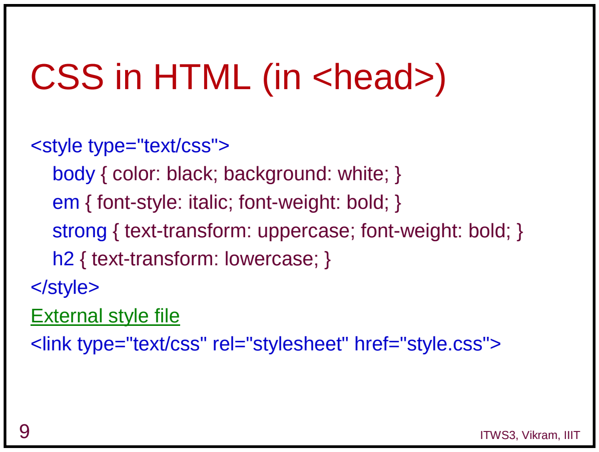
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE note SYSTEM "InternalNote.dtd">
<note>
   <to>Tove</to>
   <from>Jani</from>
   <heading>Reminder</heading>
   <body>Lunch this weekend!</body>
</note>
```

- DTD – defines the legal building blocks of an XML document.
- Valid XML – Well-formed XML that conforms to a DTD.
- XML Schema is an alternative to DTD.
- Errors will stop you



Displaying XML documents

- CSS – Html Cascading Style Sheets
  - Simple
  - Sufficient for making a consistent look
- XSL – eXtensible Stylesheet Language
  - More complex
  - Can transform XML documents

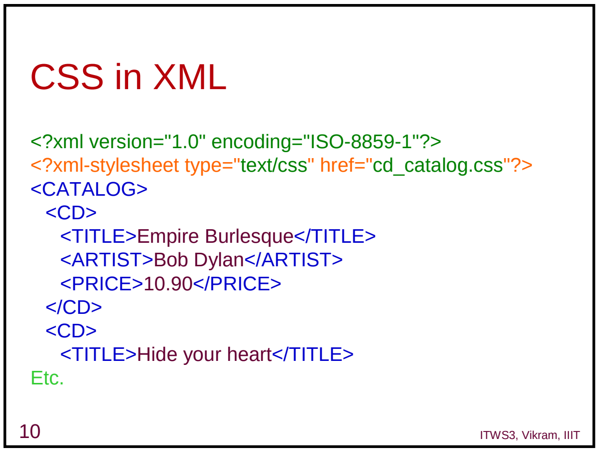


CSS in HTML (in <head>)

```
<style type="text/css">
   body { color: black; background: white; }
   em { font-style: italic; font-weight: bold; }
   strong { text-transform: uppercase; font-weight: bold; }
   h2 { text-transform: lowercase; }
</style>
```

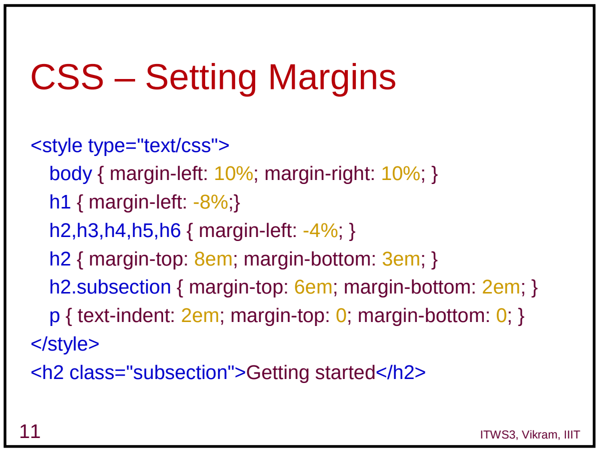
External style file

```
<link type="text/css" rel="stylesheet" href="style.css">
```



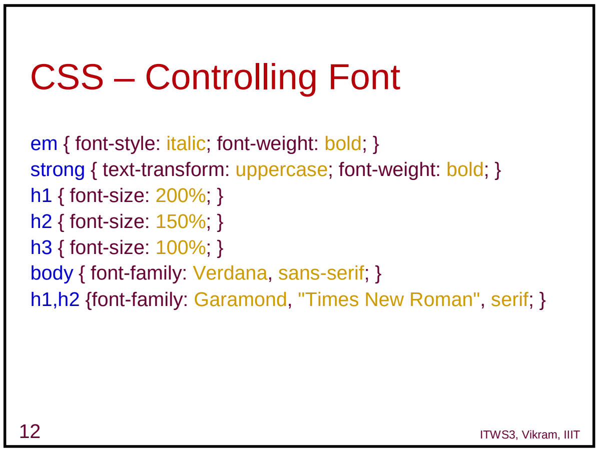
CSS in XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <PRICE>10.90</PRICE>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
Etc.
```



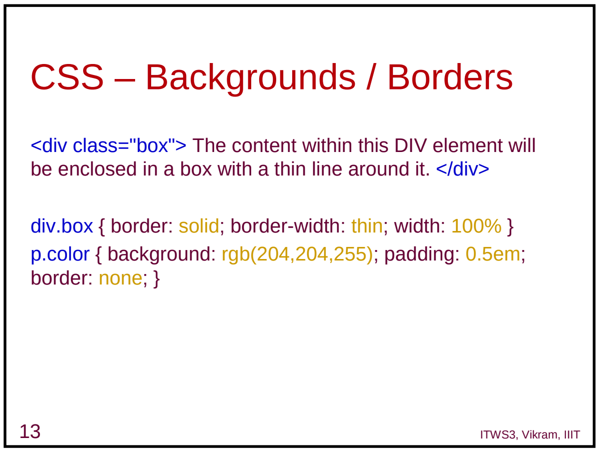
CSS – Setting Margins

```
<style type="text/css">
  body { margin-left: 10%; margin-right: 10%; }
  h1 { margin-left: -8%;}
  h2,h3,h4,h5,h6 { margin-left: -4%; }
  h2 { margin-top: 8em; margin-bottom: 3em; }
  h2.subsection { margin-top: 6em; margin-bottom: 2em; }
  p { text-indent: 2em; margin-top: 0; margin-bottom: 0; }
</style>
<h2 class="subsection">Getting started</h2>
```



CSS – Controlling Font

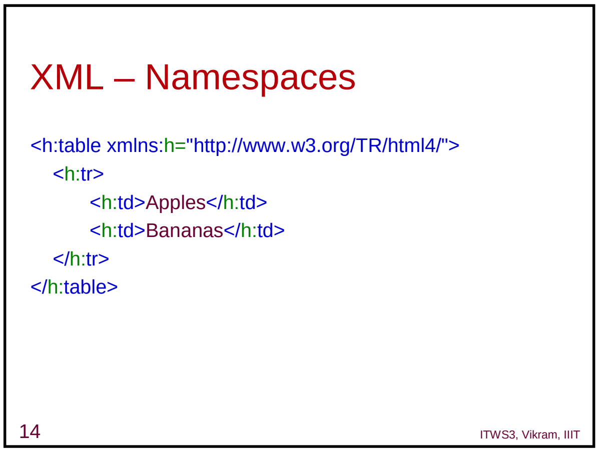
```
em { font-style: italic; font-weight: bold; }
strong { text-transform: uppercase; font-weight: bold; }
h1 { font-size: 200%; }
h2 { font-size: 150%; }
h3 { font-size: 100%; }
body { font-family: Verdana, sans-serif; }
h1,h2 {font-family: Garamond, "Times New Roman", serif; }
```



CSS – Backgrounds / Borders

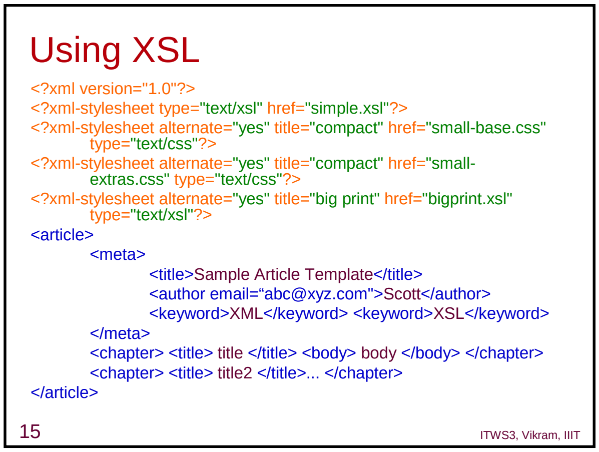
```
<div class="box"> The content within this DIV element will
be enclosed in a box with a thin line around it. </div>

div.box { border: solid; border-width: thin; width: 100% }
p.color { background: rgb(204,204,255); padding: 0.5em;
border: none; }
```



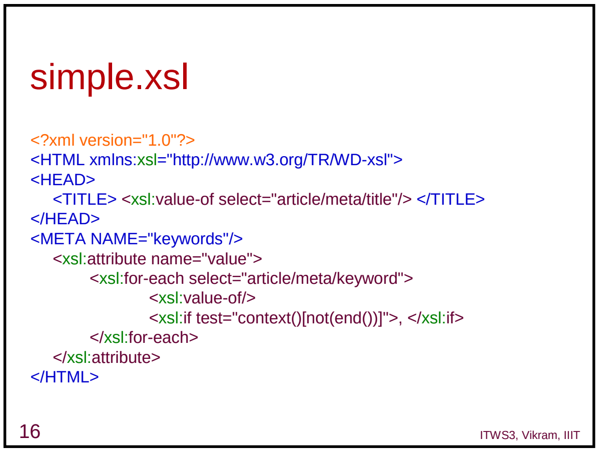
XML – Namespaces

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
   <h:tr>
        <h:td>Apples</h:td>
        <h:td>Bananas</h:td>
   </h:tr>
</h:table>
```



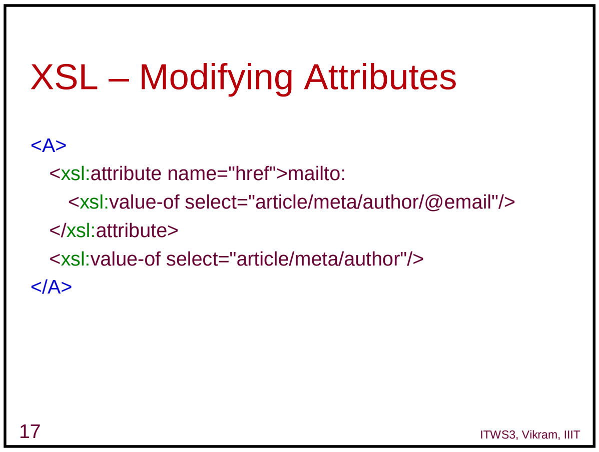
Using XSL

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="simple.xsl"?>
<?xml-stylesheet alternate="yes" title="compact" href="small-base.css"
        type="text/css"?>
<?xml-stylesheet alternate="yes" title="compact" href="small-
        extras.css" type="text/css"?>
<?xml-stylesheet alternate="yes" title="big print" href="bigprint.xsl"
        type="text/xsl"?>
<article>
        <meta>
                <title>Sample Article Template</title>
                <author email=“abc@xyz.com">Scott</author>
                <keyword>XML</keyword> <keyword>XSL</keyword>
        </meta>
        <chapter> <title> title </title> <body> body </body> </chapter>
        <chapter> <title> title2 </title>... </chapter>
</article>
```



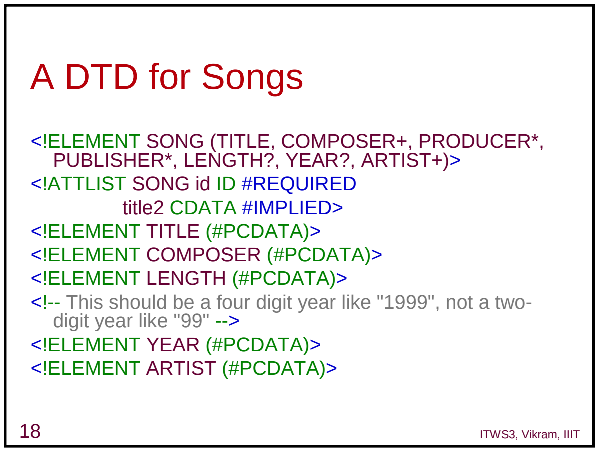
simple.xsl

```
<?xml version="1.0"?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<HEAD>
   <TITLE> <xsl:value-of select="article/meta/title"/> </TITLE>
</HEAD>
<META NAME="keywords"/>
   <xsl:attribute name="value">
        <xsl:for-each select="article/meta/keyword">
                <xsl:value-of/>
                <xsl:if test="context()[not(end())]">, </xsl:if>
        </xsl:for-each>
   </xsl:attribute>
</HTML>
```



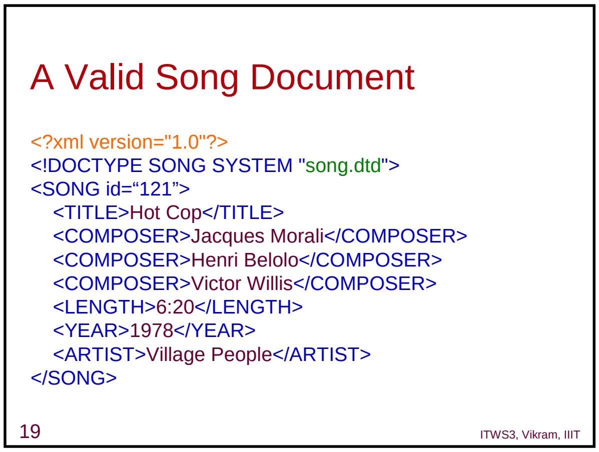
XSL – Modifying Attributes

```
<A>
  <xsl:attribute name="href">mailto:
    <xsl:value-of select="article/meta/author/@email"/>
  </xsl:attribute>
  <xsl:value-of select="article/meta/author"/>
</A>
```



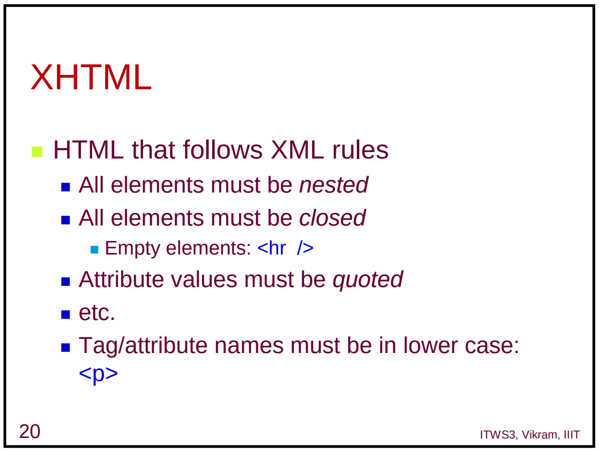
A DTD for Songs

```
<!ELEMENT SONG (TITLE, COMPOSER+, PRODUCER*,
   PUBLISHER*, LENGTH?, YEAR?, ARTIST+)>
<!ATTLIST SONG id ID #REQUIRED
            title2 CDATA #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT COMPOSER (#PCDATA)>
<!ELEMENT LENGTH (#PCDATA)>
<!-- This should be a four digit year like "1999", not a two-
   digit year like "99" -->
<!ELEMENT YEAR (#PCDATA)>
<!ELEMENT ARTIST (#PCDATA)>
```



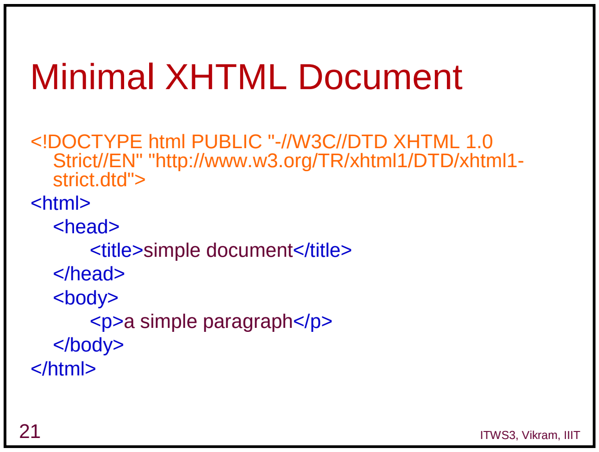
A Valid Song Document

```
<?xml version="1.0"?>
<!DOCTYPE SONG SYSTEM "song.dtd">
<SONG id=“121”>
   <TITLE>Hot Cop</TITLE>
   <COMPOSER>Jacques Morali</COMPOSER>
   <COMPOSER>Henri Belolo</COMPOSER>
   <COMPOSER>Victor Willis</COMPOSER>
   <LENGTH>6:20</LENGTH>
   <YEAR>1978</YEAR>
   <ARTIST>Village People</ARTIST>
</SONG>
```



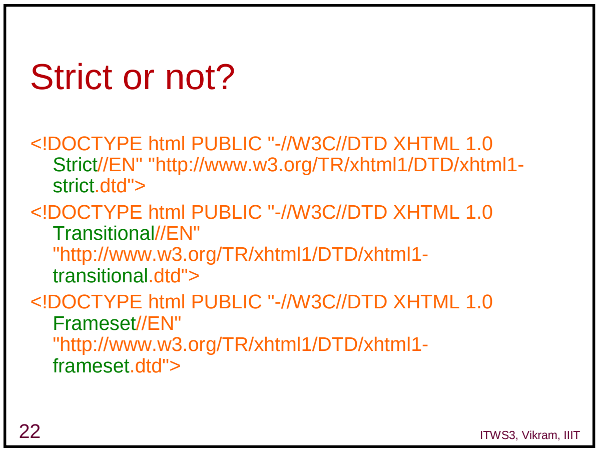
XHTML

- HTML that follows XML rules
  - All elements must be *nested*
  - All elements must be *closed*
    - Empty elements: <hr />
  - Attribute values must be *quoted*
  - etc.
  - Tag/attribute names must be in lower case: <p>



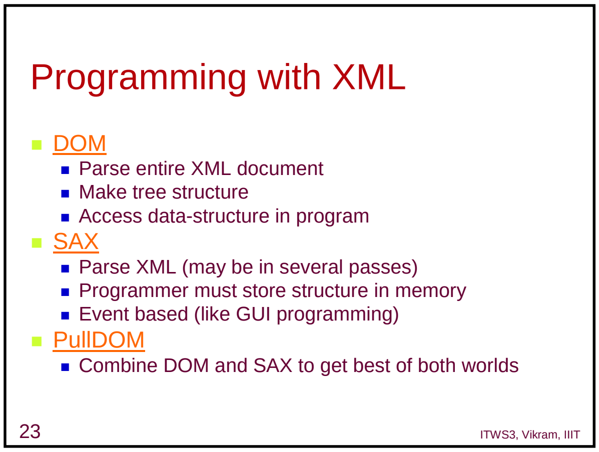
Minimal XHTML Document

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   strict.dtd">
<html>
   <head>
        <title>simple document</title>
   </head>
   <body>
        <p>a simple paragraph</p>
   </body>
</html>
```



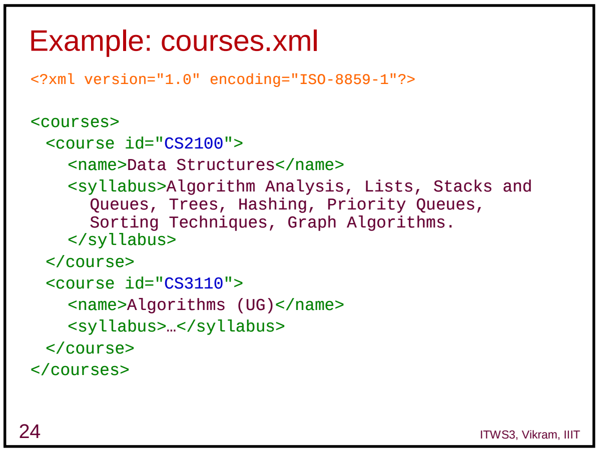
Strict or not?

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   strict.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Frameset//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   frameset.dtd">
```



Programming with XML

- DOM
  - Parse entire XML document
  - Make tree structure
  - Access data-structure in program
- SAX
  - Parse XML (may be in several passes)
  - Programmer must store structure in memory
  - Event based (like GUI programming)
- PullDOM
  - Combine DOM and SAX to get best of both worlds



Example: courses.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<courses>
  <course id="CS2100">
     <name>Data Structures</name>
     <syllabus>Algorithm Analysis, Lists, Stacks and
        Queues, Trees, Hashing, Priority Queues,
        Sorting Techniques, Graph Algorithms.
     </syllabus>
  </course>
  <course id="CS3110">
     <name>Algorithms (UG)</name>
     <syllabus>…</syllabus>
  </course>
</courses>
```

Example DOM in Python

```
from xml.dom import minidom
dom = minidom.parse('courses.xml')
courses = dom.getElementsByTagName('course')
for node in courses:
  courseid = node.attributes['id'].value
  coursenameNodes = node.getElementsByTagName('name')
  coursename = coursenameNodes[0].firstChild.data
  syllabiNodes = node.getElementsByTagName('syllabus')
  syllabus = syllabiNodes[0].toxml()
  print 'Finished.'
```

Another Example DOM in Python

```
>>> from xml.dom import minidom
>>> x='''<x a="b" d="e   f g" num="38"/>'''
>>> d = minidom.parseString(x)
>>> d.firstChild.attributes.items()
 [(u'a', u'b'), (u'num', u'38'), (u'd', u'e
 f g')]
```

Example SAX in Python

```
from xml.sax.handler import ContentHandler
from xml.sax import parse

class CourseNames(ContentHandler):
  passthrough = 0
  def startElement(self, name, attrs):
     if name == 'name': self.passthrough = 1
     elif self.passthrough: print name,
  def endElement(self, name):
     if name == 'name':
        self.passthrough = 0
        print “\n”
     elif self.passthrough: print name,
  def characters(self, chars):
     if self.passthrough: print(chars)

parse('courses.xml', CourseNames())
```

Example PullDOM in Python

```
from xml.dom import pulldom
events = pulldom.parse('courses.xml')
for (event,node) in events:
  if event == pulldom.START_ELEMENT:
    if node.tagName == 'course':
      events.expandNode(node)
      courseid = node.attributes['id'].value
      coursenameNodes =
            node.getElementsByTagName('name')
      coursename = coursenameNodes[0].firstChild.data
      syllabusNodes =
            node.getElementsByTagName('syllabus')
      syllabus = syllabusNodes[0].toxml()
  elif event == pulldom.END_ELEMENT:
    if node.tagName == 'course':
      print 'Finished.'
```

References

- XML tutorial: http://www.w3schools.com/xml/xml_whatis.asp
- CSS tutorial: http://www.w3.org/MarkUp/Guide/Style
- XHTML tutorial: http://www.w3schools.com/xhtml/default.asp
- XSL tutorial: http://www.w3schools.com/xsl/
- XML-Python programming: http://pyxml.sourceforge.net/topics/howto/xml-howto.html
- Several other tutorials (check google)