XML (HTML)

PROF. JACQUES SAVOY
UNIVERSITY OF NEUCHATEL

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

What is XML?

XML stands for eXtensible Markup Language.

XML is a markup language much like HTML used to describe data and *structured documents*. It is essential to *indicate the structure*. *Declarative* approach.

XML tags are *not* predefined. You can define your own tags.

XML uses a Document Type Definition (DTD) or an XML Schema to describe the data.

XML with a DTD or XML Schema is designed to be self-descriptive.

An XML document needs both a *description* and a *style* to be visualized.

XML vs. HTML

XML was designed to carry data and proposed by the W3C (Jon Bosak, SUN microsystems).

HTML and XML were designed with different goals (but both are derived from SGML (Standard Generalized Markup Language) 1979, Charles Goldfach, IBM; ISO in 1986).

SGML was too complex, XML is now the entry point.

HTML is about displaying information, while XML is about describing information.

XML example

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

<note>
  <to> Tom </to>
    <from> Jane </from>
    <heading> Reminder </heading>
    <body> Meeting today noon </body>
 </note>
```

The XML example was created to structure and store information.

It does not DO anything because someone must write programs to send, receive or display it.

(ako JSON principle, but focusing mainly on documents)

How XML complements HTML

XML separates data from HTML.

Data can be exchanged between incompatible systems (e.g., DB, text corpora).

With XML, plain text files can be used to share data (Digital Libraries, DL).

Usually the description is given in a separate file.

The style (presentation, stylesheet) is also described in another file (XSL for eXtensible Stylesheet Language).

Hamlet in HTML

ACT I - SCENE I

Enter Barnardo and Francisco, two Sentinels, at several doors

BARN : Who's there?
FRAN : Nay, answer me. Stand and unfold yourself.
BARN : Long live the King!
FRAN : Barnardo?
BARN : He.
FRAN : You come most carefully upon your hour.
BARN : Tis now struck twelve. Get thee to bed, Francisco.

```
<html>
<body>
<h1>ACT I - SCENE I</h1>
   <p><i>Enter Barnardo and Francisco,
         two Sentinels, at several doors</i></p>
  <p><b>BARN</b> :  ...   Who's there?</p>
  <p><b>FRAN</b> : Nay, answer me. Stand and unfold yourself.</p>
  <p><b>BARN</b> : Long live the King!</p>
  <p><b>FRAN</b> :  ...   Barnardo?</p>
  <p><b>BARN</b> :  ...   He.</p>
  <p><b>FRAN</b> : You come most carefully upon your hour.</p>
  <p><b>BARN : Tis now struck twelve. Get thee to bed, Francisco.</p>
 </body>
 </html>
```

Hamlet in XML (TEI)

```
<div1 type="Act" n="I"><head>ACT I</head>
<div2 type="Scene" n="1"><head>SCENE I</head>
<stage rend=italic>Enter Barnardo and Francisco,
            two Sentinels, at several doors</stage>
<sp><speaker>Barn</speaker><l part=Y>Who's there?</l></sp>
<sp><speaker>Fran</speaker>
    <l>Nay, answer me.  Stand and unfold yourself.</l></sp>
<sp><speaker>Barn</speaker>
    <l part=i>Long live the King!</l></sp>
<sp><speaker>Fran</speaker><l part=m>Barnardo?</l></sp>
<sp><speaker>Barn</speaker><l part=f>He.</l></sp>
<sp><speaker>Fran</speaker>
     <l>You come most carefully upon your hour.</l></sp>
<sp><speaker>Barn</speaker>
      <l>Tis now struck twelve.
      Get thee to bed,Francisco.</l></sp>
... </div2> ........</div1>
```

XML examples

```
<WebPage>
 <size> 4567 </size>
 <url> http://www.unine.ch </url>
 <author> Tintin </author>
 <email> Tintin@unine.ch </email>
 </WebPage>

<Person>
 <name> Tintin </name>
 <firstName> Jules </firstName>
 <number> 762 </number>
 <dept> Development </dept>
 <email> Tintin@magic.com </email>
 </Person>
```

XML example (TEI)

Example with Part-Of-Speech (POS) tagging encoding.

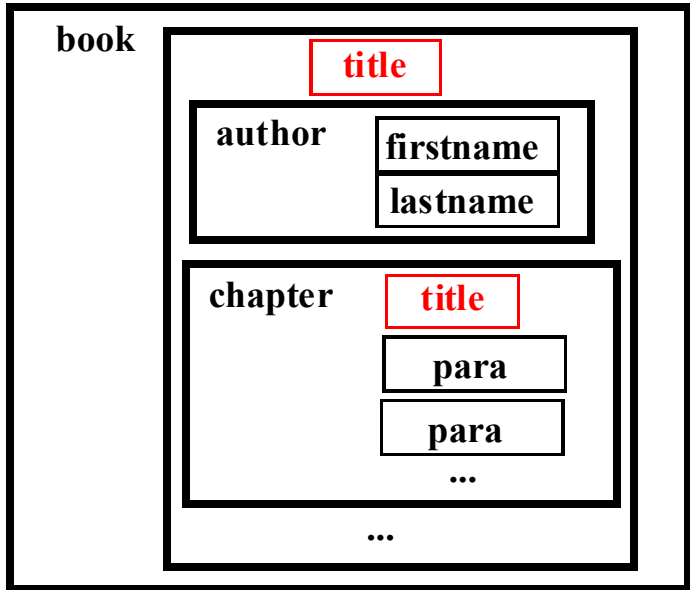
From the Text Encoding Initiative (TEI).

```
<text complete=Y  decls='CN000 HN001 QN000
  SN000'>
<div1 complete=Y org=SEQ>
<head  type=MAIN>
<s n=001>
<w NP0>CAMRA <w NN1>FACT <w NN1>SHEET
<w AT0>No <w CRD>1 </head>
<head r=it type=SUB>
<s n=002>
<w AVQ>How <w NN1>beer <w VBZ>is
<w AJ0-VVN>brewed </head>
<p><s n=003>
<w NN1>Beer <w VVZ>seems <w DT0>such
<w AT0>a <w AJ0>simple <w NN1>drink <w CJT>that
<w PNP>we <w VVB>tend <w TO0>to <w VVI>take
<w PNP>it <w CJS-PRP>for <w VVD-VVN>granted
<c PUN>.
```

XML example (TEI)

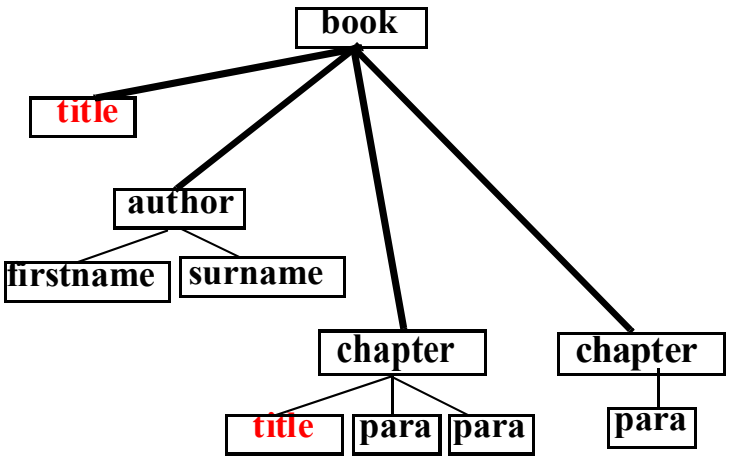
```
<u who=PS04Y>
<s n=01296><w ITJ>Mm <pause> <w ITJ>yes <pause dur=7>
<w PNP>I <w VVD>told <w NP0>Paul <pause>
<w CJT>that <w PNP>he <w VM0>can <w VVI>bring
<w AT0>a <w NN1>lady <w AVP>up <pause> <w PRP>at
<w NN1>Christmas-time<c PUN>.</u>
<u who=PS04U>
<s n=01297><w VBZ>Is <w PNP>he <w XX0>not
<w VVG>going <w AV0>home <w AV0>then<c PUN>?</u>
<u who=PS04Y>
<s n=01298><w ITJ>No <pause dur=8> <w CJC>and
<w UNC>erm <pause dur=7> <w PNP>I<w VBB>'m
<w VVG>leaving <w AT0>a <w NN1>turkey <w PRP>in
<w AT0>the <w NN1>freezer<c PUN>
<s n=01299><w NP0>Paul <w VBZ>is <w AV0>quite
<w AJ0>good <w PRP>at <w NN1-VVG>cooking <pause>
<w AJ0>standard <w NN1>cooking<c PUN>.</u>
```

The logical structure (box)




```
<book>
   <title>XML and its Applications
         </title>
   <author>
     <firstname>John</firstname>
     <lastname>Smith</lastname>
   </author>
  <chapter>
      <title>Introduction</title>
      <para>In this book, you will learn
              …  </para>
  </chapter>
  ...
</book>
```

The logical structure (tree)




```
<book>
   <title>XML and Its Applications
       </title>
   <author>
       <firstname>John</firstname>
       <surname>Smith</surname>
     </author>
  <chapter>
      <title>Introduction</title>
      <para>In this book, </para>
     </chapter>
  ...
</book>
```

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

XML

New tags and the corresponding document structure must be specified (no default values or structure).

XML is based on three concepts:

1. Elements (logical units)
   with attributes (properties attached to elements).
2. Entities (data stored)
3. Control information

XML Elements

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
  <to>Tom</to>
    <from>Jane</from>
    <heading align="center" style="bold">
        Reminder meeting at noon</heading>
    <body>Meeting</body>
 </note>
```

The first line in the document – the XML declaration – defines the XML version and character encoding used in the document.

XML Elements

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
```

- The document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set.
- *Elements*:
  `note, to, from, heading, body`
- The next line describes the root element of the document `<note>`
- The next 5 lines describe four child elements of the root: `<to>, <from>, <heading>` and `<body>`
- And finally the last line ends the root element `</note>`
- *Attributes*:
  `align` and `style` associated with the element `<heading>`

XML Elements

```
<?xml version="1.0" encoding="UTF-8"?>
<book>
  <title> Computational chemistry </title>
    <author> Adélaïde Laliberté </author>
    <img />
    <text> The main objective of this … </text>
 </book>
```

We are free to define your own tag names.

XML Elements

All XML elements *must* have a closing tag.

`<para>…</para>` or `<img/>` (`<img/>` meaning opening and closing tag)

In HTML, some elements do not have to have a closing tag, for example:

```
<p>This is a paragraph
```

XML tags are *case sensitive*, whereas HTML tags are case insensitive.

All XML elements must be properly nested and improper nesting of tags makes no sense to XML.

`<b><i>in bold and italic</b></i>` - Incorrect

`<b><i>in bold and italic</i></b>` - Correct

XML Elements

All XML documents must have a root element.

All other elements are within this root element.

All elements can have sub-elements (child elements) that are properly nested within their parent element:

```
<root>
   <child>
         <subchild>….</subchild>
   </child>
</root>
```

XML Elements

Can be organized into:

```
<book>
 <mainTitle>My First XML</mainTitle>
  <chapter><title>Introduction to XML</title>
   <para>What is HTML</para>
   <para>What is XML</para>
  </chapter>
  <chapter><title>XML Syntax     </title>
   <para>Elements must have a closing tag.</para>
   <para>Elements must be properly nested.</para>
  </chapter>
</book>
```

Be carful with the spaces and punctuation symbols that belong to the corresponding element

Element Naming

XML elements must follow these naming rules:

1. Names can contain letters, numbers, and other characters.
2. Names must not start with a number or punctuation character (but can start with "_").
   Be careful with the ":" (see later).
3. Names must not start with the sequence xml (or XML or Xml ..).
4. Names cannot contain spaces.
5. Names may contain - _ . : or digits.

Element Naming

Examples

`<italic>`	correct
`<Italic>`	correct (but different)
`<xmlIta>`	incorrect (start with "xml")
`<34Mem>`	incorrect
`<_italic>`	correct
`<Uni@fr>`	incorrect
`<titre-Sect>`	correct

Element Naming

Non-English letters like "森" are perfectly legal in XML element names, but watch out for problems if the software vendor doesn't support them.

`<modèle>`	correct
`<Äpfel>`	correct

The ":" should not be used in element names because it is reserved for *namespaces* (discussed later).

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

XML Attributes

XML elements can have *attributes* in name/value pairs.

The attribute value *must* always be *quoted* (double or single).

```
<?xml version = "1.0" encoding = "utf-8"?>
<note date = "6/1/2021">
  <to>Tom</to>
</note>
```

HTML was more permissive as, for example,

```
<img src = "/images/lake.gif">
<img src=/images/lake.gif>
```

XML Attributes

It is not always clear when using the attribute and when using an element.

Three examples with a date

```
<note date="6/1/2004">
<date>6/1/2004</date>
<date>
  <day>1</day>
  <month>6</month>
  <year>2004</year>
</date>
```

But the date format could be

US: month/day/year

FR: day/month/year

SV: year/month/day

XML Attributes

It is not always clear when using the attribute and when using an element.

```
<img src="../images/Peter.jpg" height="120" width= "200">
```

```
<img>
 <src>"../images/Peter.jpg" </src>
 <height>120</height>
 <width>200</width>
 </img>
```

```
<distance amount="120" unit="km">
   <from>Geneva</from>
   <to>Neuchâtel</to>
 </distance>
```

XML Attributes

Avoid using attributes? My suggestion: *Yes*

Use element instead of attribute!

1. Attribute cannot contain multiple values.
2. Attribute are not expandable (for future changes).
3. Attribute cannot describe the structures (like children elements can).
4. Attributes are more difficult to manipulate by program code.
5. Attributes values are not easy to test against a DTD.

But a good practice for attribute is the use of an identifier reference for a given XML documents. Using ID is a good practice.

```
<note ID="BG-89">
```

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

XML Control Information

Mainly three types of control information.

Comments: the syntax for writing comments in XML is similar to that of HTML:

```
<!-- this is a comment -->
```

Processing information:

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

Document type declarations: (see later)

```
<!DOCTYPE docType SYSTEM "aGrammar.dtd">
```

XML Entities

Entities are used to store data.

XML knows a set of predefined entities (corresponding to metacharacters used in the definition of the elements).

```
&lt;       >
&gt;       <
&amp;      &
&quot;     ”
&apos;     '
```

Entities types are explained later (with the DTD).

You may add others entities.

Special characters can be introduced like `©` (or `©`) for ©

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD (Document Type Definition)

Examples

Corpora / Tools

Examples with XML

Examples with HTML

Introduction to DTD

The purpose of a Document Type Definition is to define the legal building blocks of an XML document.

You don't expect that you will be an expert in XML. Show how this language was developed.

A DTD defines the document structure with a list of legal elements.

XML validated against a Document Type Definition (DTD) is *valid* XML.

The DTD specification is still the "king" (XML Schema is the next generation).

Each element appears between

`<!ELEMENT … … >`

within one can find an element name and a content.

Introduction to DTD

A DTD can be declared *inline* in your XML document or as an *external* reference (better).

If a DTD is included in your XML document, it should be wrapped in a DOCTYPE definition:

1. Inline

```
<!DOCTYPE root-element [element-declarations]>
```

2. External Reference

```
<!DOCTYPE note SYSTEM "Outside.dtd">
```

where note is the root element.

Introduction to DTD

An Inline DTD example:

```
<?xml version="1.0"?>
    <!DOCTYPE note [
    <!ELEMENT note (to, from, heading, body)>
      <!ELEMENT to      (#PCDATA)>
      <!ELEMENT from    (#PCDATA)>
      <!ELEMENT heading (#PCDATA)>
      <!ELEMENT body    (#PCDATA)>
    ]>
<note>
    <to>Bob</to>
    <from>Alice</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

Declaring a DTD Element

In the DTD, XML elements are declared with an *element* declaration:

```
<!ELEMENT element-name category>
```

or

```
<!ELEMENT element-name (element-content)>
```

There are various types of elements that we can declare in a DTD.

Declaring a DTD Element

The reserved keyword for element-content are:

- `PCDATA` Parsed character data (only text)

  `<!ELEMENT para PCDATA>`
- `ANY` PCDATA or and DTD element

  `<!ELEMENT note ANY>`

  （useful for image)
- `EMPTY` no content – just a placeholder

  `<!ELEMENT br EMPTY>`

Declaring a DTD Element

Elements with children (sequences)

- <!ELEMENT *element-name* (*child-element-name*, …) >
- Note that when children are declared in a sequence separated by commas, the children must appear in the *same sequence* in the document.

Introduction to DTD

An Inline DTD example:

```
1:<?xml version="1.0"?>
2:     <!DOCTYPE note [
3:    <!ELEMENT note (to, from, heading, body)>
4:       <!ELEMENT to      (#PCDATA)>
5:       <!ELEMENT from    (#PCDATA)>
6:       <!ELEMENT heading (#PCDATA)>
7:       <!ELEMENT body    (#PCDATA)>
8:     ]>
9: <note>
10:    <to>Bob</to>
11:     <from>Alice</from>
12:     <heading>Reminder</heading>
13:     <body>Don't forget me this weekend!</body>
14: </note>
```

Declaring a DTD Element

!DOCTYPE `note` (in line 2) defines that this is a document of type `note`.

!ELEMENT `note` (in line 3) defines the note element as having four elements:
`"to, from, heading, body"`

!ELEMENT `to` (in line 4) is defined to be of type "`#PCDATA`".

!ELEMENT `from` (in line 5) is defined to be of type "`#PCDATA`"

and so on.....

Another DTD Example

External Reference example:

```
<?xml version="1.0"?>
 <!DOCTYPE note SYSTEM "note.dtd">
 <note>
  <to>Bob</to>
      <from>Alice</from>
      <heading>Reminder</heading>
      <body>Don't forget me this weekend!</body>
 </note>
```

Another DTD Example

In the file Note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Types of DTD elements

Use of meta characters (like in Regexp)

Declaring only one occurrence of the same element

- <!ELEMENT *element-name* (*child-name*)>

Declaring minimum one (or more) occurrence of the same element

- <!ELEMENT *element-name* (*child-name+*)>

Declaring zero or more occurrences of the same element

- <!ELEMENT *element-name* (*child-name**)>

Declaring zero or one occurrence of the same element

- <!ELEMENT *element-name* (*child-name*?)>

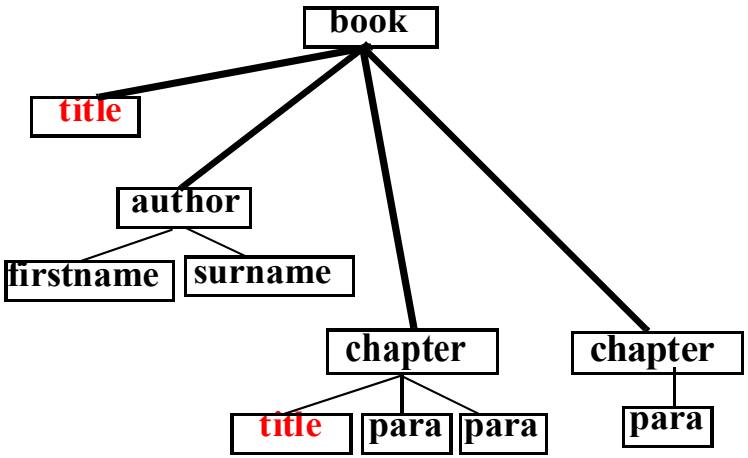
A more complex DTD

A DTD for a book:

```
<!ELEMENT book (title, (author | editor)?, img, chapter+)>
<!ELEMENT title    (#PCDATA)>
<!ELEMENT author   (#PCDATA)>
<!ELEMENT editor   (#PCDATA)>
<!ELEMENT img      (#EMPTY)>
<!ELEMENT chapter  (subtitle, para+)>
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT para     (#PCDATA)>
```

DTD Example

```
<!DOCTYPE book [
<!ELEMENT book        - - (title,
                 author*, chapter+)>
<!ELEMENT title       - - (#PCDATA)>
<!ELEMENT author      - -
(firstname?,
                          surname)>
<!ELEMENT firstname   - - (#PCDATA)>
<!ELEMENT surname     - - (#PCDATA)>
<!ELEMENT chap        - - (title?,
                          para+)>
<!ELEMENT para        - - (#PCDATA)>
]>
```



*	0-n
+	1-n
?	0-1

Types of DTD elements

Declaring either/or content

- `<!ELEMENT note (to, from, header, (message | body))>`

Declaring mixed content

- `<!ELEMENT note (#PCDATA|to|from|header|message)*>`

DTD attributes are declared with an ATTLIST declaration
useful to specify a presentation
or to impose conditions on values that are attached to a given attribute

Attribute

The main keyword for attribute are:

- `CDATA` The string type
- `ID` An identifier of the element unique in the document
- `IDREF` A reference to an identifier

  …

The predefined default value for attribute are:

- `#REQUIRED` A value must be supplied
- `#FIXED` *value* Attribute value is constant and must be equal to the default value
- `#IMPLIED` If no value is supplied, the processing system will define the value

Attribute

Use the following syntax <!ATTLIST *element-name attribute-name attribute-type default-value*>

In the DTD

```
<!ATTLIST author style
        (underlined | bold | italics) "italics")>
<!ATTLIST author align
        (left | center | right) "left")>
```

In the DTD

```
<!ATTLIST payment type CDATA "check">
```

In the XML file

```
<payment type="check" />
```

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded (any char except < > & ' ").

Attribute

In the DTD

```
<!ATTLIST payment type (check | cash) "cash">
```

In the XML file

```
<payment type="cash" />
```

…

```
<payment type="check" />
```

In the DTD

```
<!ATTLIST person number CDATA #REQUIRED>
```

In the XML file

```
<person number="568" />
```

…

`<person />` → Invalid

Attribute

In the DTD

```
<!ATTLIST sender university CDATA #FIXED "UniNE">
```

In the XML file

```
<sender university="UniNE" />  …
<sender university="ETHZ" />
```

→ Invalid not UniNE

In the DTD

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

In the XML file

```
<contact fax="514-3437568" />
```

If you don't want to force the author to include an attribute, and you don't have an option for a default value.

Example of a DTD

An Inline DTD example:

```
<!ELEMENT book (title, (author | editor)?, img, chapter+)>

<!ELEMENT title (#PCDATA)>
<!ATTLIST title style
     (underlined | bold | italics) "bold")>
<!ELEMENT author  (#PCDATA)>
<!ATTLIST author style
     (underlined | bold | italics) "italics")>
<!ELEMENT editor  (#PCDATA)>
<!ELEMENT img     (#EMPTY)>
<!ATTLIST img src CDATA #REQUIRED>
<!ELEMENT chapter (subtitle para+)>
<!ATTLIST chapter number ID #REQUIRED>
<!ATTLIST chapter numberStyle
   (Arabic | Roman) "Roman">
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT para (#PCDATA)>
```

Attribute

Some attributes are already pre-defined by XML

Example

```
<para xml:lang="fr">Bonjour</para>
<para xml:lang="en">Hello</para>
<para xml:lang="de">Guten Tag</para>
```

The language code is given by the ISO-639 standard.

The second reserved attribute is

```
<para xml:space="default">…</para>
<para xml:space="preserve">…</para>
```

specifying if the space present in the element must be preserve or the parser is free to use them (default)

Naming space

Sometimes, we need to work in a distributed system or we must use predefined DTD.
In this case, we prefix the name by the needed name space (Uni in the example)

```
<Uni:Person>
   <Uni:name>Tintin</Uni:name>
   <Uni:firstName>Jules</Uni:firstName>
   <Uni:number> 762</Uni:number>
   <Uni:dept>biology</Uni:dept>
   <Uni:email>Tintin@magic.com</Uni:email>
</Uni:Person>
```

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

Théâtre français

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

<TEI.2>  <teiHeader> <fileDesc>  <titleStmt>

<title>ANDROMAQUE, TRAGÉDIE</title>

<author born="1639" born_location="La Ferté Millon" death="1699"
death_location="Paris" academie="1673">RACINE, Jean</author>

 </titleStmt>

 <publicationStmp>

  <p>publié par Paul FIEVRE, Mai 2006, revu décembre 2013</p>

</publicationStmp>
```

Théâtre français

```
<SourceDesc>
  <genre>Tragédie</genre>
      <inspiration>mythe grec</inspiration>
      <structure>Cinq actes</structure>
      <type>vers</type>
      <periode>1661-1670</periode>
      <taille>1500-1750</taille>

<permalien>http://gallica.bnf.fr/ark:/12148/bpt6k70167z</permalien>

<sources>
      <source id="1">
      <author>Euripide</author>
      <text>Andromaque</text>
      </source>
```

Théâtre français

```
<source id="2">
   <author>Virgile</author>
   <text>Enéïde</text>
  </source>
</sources>  </SourceDesc>  </fileDesc>  </teiHeader>
<text>  <front>  <docTitle>
 <titlePart type="main">ANDROMAQUE.</titlePart>
 <titlePart type="sub">TRAGÉDIE</titlePart>
 </docTitle>
 <docDate value="1668">M. DC. LXVIII. AVEC PRIVILÈGE DU ROI.</docDate>
 <docAuthor id="RACINE, Jean" bio="racine"></docAuthor>
 <docImprint>
<privilege id="1667-12-28"><head>PRIVILÈGE DU ROI</head>
```

Théâtre français

```
<p>[Ce privilège est celui de l'édition 1668 de l'exemplaire Rés. Yf-3206
de la BnF].</p>

<p>Louis par la Grâce de Dieu, Roi de France et de Navarre ; …

 Signé DEMALON.</p>

<p>Et le dit sieur Racine a cédé son droit de Privilège à Théodore Girard,
marchand libraire à Paris, suivant l'accord fait entre eux.</p>
</privilege>

<acheveImprime id="1668-01-20">Achevé d'imprimé seconde quinzaine de
janvier 1668.</acheveImprime>

<printer id="Théodore GIRARD">À Paris, Chez Théodore Girard, dans la
Grand' Salle du Palais, du côté de la cour de Aides, à l'Envie.</printer>
</docImprint>

…
```

Théâtre français

```
<performance>

<premiere date="1667-11-17" location="Appartement de la Reine">Représenté
la première fois 17 novembre 1667, dans l'Appartement de la Reine, et sans
doute reprise le 18 novembre 1667 (selon Georges Forestier) au Théâtre de
l'Hôtel de Bourgogne.</premiere>

</performance>

<div type="dedicace">

<adresse>À MADAME</adresse>

<head>MADAME,</head>

<p>Ce n'est pas sans sujet que je mets votre illustre nom à la tête de cet
ouvrage. Et de quel autre nom pourrais-je éblouir les yeux de mes lecteurs,
que de celui dont mes spectateurs ont été si …
```

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

Corpora

Linguistic Data Consortium (LDC)
http://www.ldc.upenn.edu

European Language Resources Association (ELRA)
http://www.elra.info

British National Corpus (BNC)
http://www.natcorp.ox.ac.uk/

The Text Encoding Initiative (TEI)
http://www.tei-c.org/

Information retrieval
TREC http://trec.nist.gov
CLEF http://www.clef-campaign.org
NTCIR http://research.nii.ac.jp/ntcir/

What is the BNC?

A snapshot of British English at the end of the 20th century

100 million words in approx. 4,000 different text samples,
both spoken (10%) and written (90%)

- written
  - medium (books, newspapers, unpublished…)
  - domain (informative, entertaining…)
- spoken
  - context-governed
  - demographically-sampled

synchronic (1990-4), sampled, general purpose corpus

A model for European corpus work (TEI)

Other tools

XSL (and XLST) Stylesheet

XPath use to locate nodes inside a document tree

`chap`	select the elements "chap", sons of the current node
`../chap`	select the elements "chap", sons of the father of the current node
`/`	root
`.`	current node
`..`	parent
`//`	sons
`@`	attribute

XLink and XPointer to locate specific section into a document

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

XML

```
>>> with open('data/sonnets/18.xml') as stream:
...     aSonnet = stream.read()   # ignore the xml tags
...

>>> print(aSonnet)
<?xml version="1.0"?>
<sonnet author="William Shakepeare" year="1609">
  <line n="1">Shall I compare thee to a summer's <rhyme>day</rhyme>?</line>
  <line n="2">Thou art more lovely and more <rhyme>temperate</rhyme>:</line>
  <line n="3">Rough winds do shake the darling buds of <rhyme>May</rhyme>,</line>
  …
  <line n="14">So long lives this, and this gives life to <rhyme>thee</rhyme>.</line>
</sonnet>
```

Not the correct spelling?

Root element

child element with an attribute denoted "n"

XML

View of the file 18.xml

With the BBEdit editor

18.xml

Free Mode

~/Documents/Cours/Cours-DH/Exercises/02-getting-data/data/sonnets/18.xml (functions)

```xml
<?xml version="1.0"?>
<sonnet author="William Shakepeare" year="1609">
    <line n="1">Shall I compare thee to a summer's <rhyme>day</rhyme>?</line>
    <line n="2">Thou art more lovely and more <rhyme>temperate</rhyme>:</line>
    <line n="3">Rough winds do shake the darling buds of <rhyme>May</rhyme>,</line>
    <line n="4">And summer's lease hath all too short a <rhyme>date</rhyme>:</line>
    <line n="5">Sometime too hot the eye of heaven <rhyme>shines</rhyme>,</line>
    <line n="6">And often is his gold complexion <rhyme>dimm'd</rhyme>;</line>
    <line n="7">And every fair from fair sometime <rhyme>declines</rhyme>,</line>
    <line n="8">By chance, or nature's changing course, <rhyme>untrimm'd</rhyme>;</line>
    <volta/>
    <line n="9">But thy eternal summer shall not <rhyme>fade</rhyme></line>
    <line n="10">Nor lose possession of that fair thou <rhyme>ow'st</rhyme>;</line>
    <line n="11">Nor shall Death brag thou wander'st in his <rhyme>shade</rhyme>,</line>
    <line n="12">When in eternal lines to time thou <rhyme>grow'st</rhyme>;</line>
    <line n="13">So long as men can breathe or eyes can <rhyme>see</rhyme>,</line>
    <line n="14">So long lives this, and this gives life to <rhyme>thee</rhyme>.</line>
</sonnet>
```

L: 1 C: 1 XML Saved: 03.12.19, 14:21:17 1'210 / 209 / 18 100%

Manipulating a XML file

```
>>> import lxml.etree        # Use the dedicated package
>>> tree = lxml.etree.parse('data/sonnets/18.xml’)
>>> print(tree)
<lxml.etree._ElementTree object at 0x7fae28028280
```

Decoding is needed to transform the bytes object into an actual string

```
>>> print(lxml.etree.tostring(tree).decode())

 <sonnet author="William Shakepeare" year="1609">
<line n="1">Shall I compare thee to a summer’s <rhyme>day</rhyme>?</line>
<line n="2">Thou art more lovely and more <rhyme>temperate</rhyme>:</line>
…
```

Manipulating a XML file

```
<sonnet author="William Shakepeare" year="1609">
<line n="1">Shall I compare thee to a summer’s <rhyme>day</rhyme>?</line>
<line n="2">Thou art more lovely and more <rhyme>temperate</rhyme>:</line>
<line n="3">Rough winds do shake the darling buds of <rhyme>May</rhyme>,</line>
<line n="4">And summer's lease hath all too short a <rhyme>date</rhyme>:</line>
<line n="5">Sometime too hot the eye of heaven <rhyme>shines</rhyme>,</line>
<line n="6">And often is his gold complexion <rhyme>dimm'd</rhyme>;</line>
<line n="7">And every fair from fair sometime <rhyme>declines</rhyme>,</line>
<line n="8">By chance, or nature's changing course, <rhyme>untrimm'd</rhyme>;</line>
<line n="9">But thy eternal summer shall not <rhyme>fade</rhyme></line>
<line n="10">Nor lose possession of that fair thou <rhyme>ow'st</rhyme>;</line>
<line n="11">Nor shall Death brag thou wander'st in his <rhyme>shade</rhyme>,</line>
<line n="12">When in eternal lines to time thou <rhyme>grow'st</rhyme>;</line>
<line n="13">So long as men can breathe or eyes can <rhyme>see</rhyme>,</line>
<line n="14">So long lives this, and this gives life to <rhyme>thee</rhyme>.</line>
</sonnet>
```

To indicate a change

<volta/>

Manipulating a XML file

```
>>> for rhyme in tree.iterfind('//rhyme'):
...     print(f'element: {rhyme.tag} -> {rhyme.text}')
...
element: rhyme -> day
element: rhyme -> temperate
element: rhyme -> May
element: rhyme -> date
element: rhyme -> shines
element: rhyme -> dimm’d
element: rhyme -> declines
element: rhyme -> untrimm'd
element: rhyme -> fade
element: rhyme -> ow'st
element: rhyme -> shade
element: rhyme -> grow'st
element: rhyme -> see
element: rhyme -> thee
```

Manipulating a XML file

```
>>> root = tree.getroot()
>>> print(root.tag)                    # the root name
    sonnet
>>> print(root.attrib['year'])         # access an attribute value
    1609

>>> print(len(root))
    15

>>> children = [child.tag for child in root]
>>> children

   ['line', 'line', 'line', 'line', 'line', 'line', 'line', 'line', 'volta',
'line', 'line', 'line', 'line', 'line', 'line']

>>> print('\n'.join(child.text or '' for child in root))

    Shall I compare thee to a summer’s
    Thou art more lovely and more
    Rough winds do shake the darling buds of
    And summer's lease hath all too short a …
```

Manipulating a XML file

```
>>> print(''.join(root[0].itertext()))    # extract the content of a XML node
 Shall I compare thee to a summer's day?

>>> for node in root:
...     if node.tag == 'line':      # need to know the element name
...         print(f"line {node.attrib['n']: >2}:{''.join(node.itertext())}")
...

 line  1: Shall I compare thee to a summer's day?
 line  2: Thou art more lovely and more temperate:
 line  3: Rough winds do shake the darling buds of May,
 line  4: And summer's lease hath all too short a date:
 line  5: Sometime too hot the eye of heaven shines,
 line  6: And often is his gold complexion dimm’d;
…
 line 13: So long as men can breathe or eyes can see,
 line 14: So long lives this, and this gives life to thee.
```

Creating a XML file

```
>>> root = lxml.etree.Element('sonnet')   # create an XML document
>>> root.attrib['author'] = 'William Shakespeare'   # create an attribute
>>> root.attrib['year'] = '1609'
>>> tree = lxml.etree.ElementTree(root)
>>> stringified = lxml.etree.tostring(tree)
>>> print(stringified)
  b'<sonnet author="William Shakespeare" year="1609"/>'
>>> print(type(stringified))
  <class 'bytes'>
>>> print(stringified.decode('utf-8'))
  <sonnet author="William Shakespeare" year="1609"/>
```

TXT

View of the file 116.txt

116.txt

```
Let me not to the marriage of true minds
Admit impediments. Love is not love
Which alters when it alteration finds,
Or bends with the remover to remove:
O no; it is an ever-fixed mark,
That looks on tempests, and is never shaken;
It is the star to every wandering bark,
Whose worth's unknown, although his height be taken.
Love's not Time's fool, though rosy lips and cheeks
Within his bending sickle's compass come;
Love alters not with his brief hours and weeks,
But bears it out even to the edge of doom.
If this be error and upon me proved,
I never writ, nor no man ever loved.
```

Creating a XML file

```
>>> for nb, line in enumerate(open('data/sonnets/116.txt')):
...     node = lxml.etree.Element('line')
...     node.attrib['n'] = str(nb + 1)
...     node.text = line.strip()
...     root.append(node)
...                                 # voltas typically, but not always occur between the octave and sextet
...     if nb == 8:
...         node = lxml.etree.Element('volta')
...         root.append(node)
```

```
>>> print(lxml.etree.tostring(tree, pretty_print=True).decode())
<sonnet author="William Shakespeare" year="1609">
   <line n="1">Let me not to the marriage of true minds</line>
   <line n="2">Admit impediments. Love is not love</line>
   <line n="3">Which alters when it alteration finds,</line>
…
  <line n="14">I never writ, nor no man ever loved.</line>
</sonnet>
```

Manipulating a XML file

The rhyme elements are missing

```
>>> for node in root:                    # Loop over all nodes in the tree
...     if node.tag == 'volta':           # Leave the volta node alone
...         Continue
...     punctuation = ''                  # We chop off and store verse-final punctuation
...     if node.text[-1] in ',:;.':       # Punctuation must appear after the rhyme element
...         punctuation = node.text[-1]
...         node.text = node.text[:-1]
...     words = node.text.split()                 # Make a list of words
...     other_words, rhyme = words[:-1], words[-1]    # We split rhyme words and others
        # Replace the node's text with all text except the rhyme word
...     node.text = ' '.join(other_words) + ' '
...     elt = lxml.etree.Element('rhyme')         # We create the rhyme element
...     elt.text = rhyme
...     elt.tail = punctuation
...     node.append(elt)      # We add the rhyme to the line
```

XML

View of the file 116.xml

116.xml

Free Mode

Currently Open Documents

~/Documents/Cours/Cours-DH/Exercises/02-getting-data/data/sonnets/116.xml

(functions)

```xml
<?xml version='1.0' encoding='utf-8'?>
<sonnet author="William Shakespeare" year="1609">
  <line n="1">Let me not to the marriage of true <rhyme>minds</rhyme></line>
  <line n="2">Admit impediments. Love is not <rhyme>love</rhyme></line>
  <line n="3">Which alters when it alteration <rhyme>finds</rhyme>,</line>
  <line n="4">Or bends with the remover to <rhyme>remove</rhyme>:</line>
  <line n="5">O no; it is an ever-fixed <rhyme>mark</rhyme>,</line>
  <line n="6">That looks on tempests, and is never <rhyme>shaken</rhyme>;</line>
  <line n="7">It is the star to every wandering <rhyme>bark</rhyme>,</line>
  <line n="8">Whose worth's unknown, although his height be <rhyme>taken</rhyme>.</line>
  <line n="9">Love's not Time's fool, though rosy lips and <rhyme>cheeks</rhyme></line>
  <volta/>
  <line n="10">Within his bending sickle's compass <rhyme>come</rhyme>;</line>
  <line n="11">Love alters not with his brief hours and <rhyme>weeks</rhyme>,</line>
  <line n="12">But bears it out even to the edge of <rhyme>doom</rhyme>.</line>
  <line n="13">If this be error and upon me <rhyme>proved</rhyme>,</line>
  <line n="14">I never writ, nor no man ever <rhyme>loved</rhyme>.</line>
</sonnet>
```

L: 1 C: 1 XML Saved: 04.05.21, 11:40:31 1'201 / 208 / 19 100%

Manipulating a XML file

```
>>> tree = lxml.etree.ElementTree(root)
>>> print(lxml.etree.tostring(tree, pretty_print=True).decode())
<sonnet author="William Shakespeare" year="1609">
   <line n="1">Let me not to the marriage of true<rhyme>minds</rhyme></line>
   <line n="2">Admit impediments. Love is not <rhyme>love</rhyme></line>
   <line n="3">Which alters when it alteration <rhyme>finds</rhyme>,</line>
   <line n="4">Or bends with the remover to <rhyme>remove</rhyme>:</line>
   <line n="5">O no; it is an ever-fixed <rhyme>mark</rhyme>,</line>
   <line n="6">That looks on tempests, and is never
<rhyme>shaken</rhyme>;</line>
   <line n="7">It is the star to every wandering <rhyme>bark</rhyme>,</line>
   …
   <line n="14">I never writ, nor no man ever <rhyme>loved</rhyme>.</line>
</sonnet>
```

Manipulating a XML file

```
>>> with open('data/sonnets/116.xml', 'w') as f:
...     f.write(
...         lxml.etree.tostring(
                  root,
                  xml_declaration=True,
                  pretty_print=True,
                  encoding='utf-8').decode())

1201
```

More Complete Example

```
>>> import os
>>> import lxml.etree    # Use the dedicated package
>>> import tarfile
>>> tf=tarfile.open('data/theatre-classique.tar.gz','r')
>>> tf.extractall('data')    # need to be done once

>>> subgenres = ('Comédie','Tragédie','Tragi-comédie')
>>> plays, titles, genres = [], [], []
>>> authors, years = [], []
```

Manipulating an XML file

```
for fn in os.scandir('data/theatre-classique’):
   if not fn.name.endswith('.xml’): # Only XML files
      continue
   tree   = lxml.etree.parse(fn.path)
   genre  = tree.find('//genre’)      # the genre element
   title  = tree.find('//title')
   author = tree.find('//author')
   year   = tree.find('//date')
   if genre is not None and genre.text in subgenres:
      lines = []
      for line in tree.xpath('//l|//p'):
         lines.append(' '.join(line.itertext()))
         text = '\n'.join(lines)
         plays.append(text)
         genres.append(genre.text)
         titles.append(title.text)
         authors.append(author.text)
         if year is not None:
            years.append(year.text)
```

Manipulating an XML file

```
>>> print (len(plays), len(genres), len(titles), len(authors), len(years))
   498 498 498 498 208
>>> genres[0]
   'Comédie'
>>> titles[0]
   'LE MARI DIRECTEUR ou LE DÉMÉNAGEMENT DU COUVENT, COMÉDIE EN UN ACTE et
    EN VERS LIBRES.'
>>> authors[0]
   'CARBON DE FLINS, Claude de'
>>> years[0]
   '1707'
```

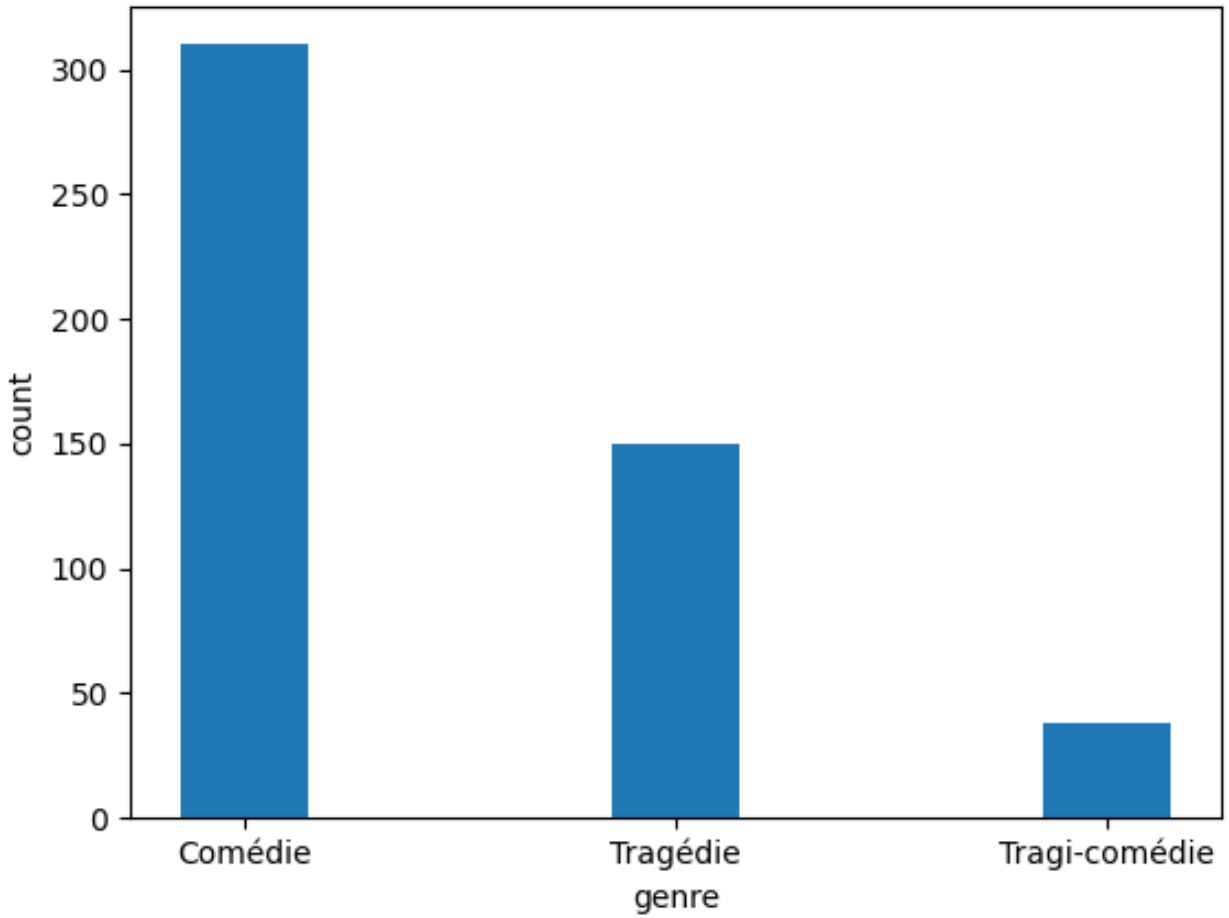
Manipulating an XML file

```
>>> import collections
>>> import matplotlib.pyplot as plt      # overview statistics
>>> counts = collections.Counter(genres)

>>> fig, ax = plt.subplots()
>>> ax.bar(counts.keys(), counts.values(), width=0.3)
   <BarContainer object of 3 artists>
>>> ax.set(xlabel="genre", ylabel="count");
   [Text(0.5, 0, 'genre'), Text(0, 0.5, 'count')]
>>> fig.show()
```

Manipulating

```
>>> fig.show()
>>> …
```



Manipulating an XML file

Some problems: normalization of the authors' names

```
>>> authors

['CARBON DE FLINS, Claude de', 'SAINT-AIGNAN, Etienne', …
'VOLTAIRE', 'VOLTAIRE', 'CHAMPMÉSLÉ, Charles Chevillet dit',
'CHABANON, Michel Paul Guy de', 'VOLTAIRE', 'MARIVAUX', 'MOLIÈRE',
'Molière', 'FERIOL DE PONT-DE-VEYLE, Antoine de', 'MARIVAUX',
'Anonyme', 'VOLTAIRE', 'CHABANON, Michel Paul Guy de', 'VOLTAIRE',
"TRISTAN L'HERMITE, François", 'VOLTAIRE', 'VOLTAIRE, François-Marie
AROUET de', 'VOLTAIRE', 'Anonyme', 'BIEVRE, François George Maréchal,
marquis de', 'MARIVAUX', 'Molière', …
'MOLIÈRE', 'MARIVAUX', 'DU RYER, Pierre', 'FLORIAN, Jean Pierre Claris
de', 'CRÉBILLON, Prosper J. de', "AIGUEBERRE, Jean Dumas d'",
'BOUCHER, Pierre', 'BOISSY, Louis de', 'SEDAINE, Michel-Jean',
'SCARRON, Paul', 'SAINT-AIGNAN, Etienne', 'CAMPISTRON, Jean G. de']
```

Manipulating TEI

TEI is strongly related to XML. Adopted by the EU for its digital archives.

Some consistency inside a given dataset as the Folger Digital texts (TEI-inspired) using the XML namespaces

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
```

You specify this namespace at the root level (file `Oth.xml`).

```
<?xml version="1.0" encoding="utf-8"?>
 <?xml-stylesheet type="text/xsl" href="fdt.xsl"?>
 <TEI xmlns="http://www.tei-c.org/ns/1.0">
 <teiHeader>  <fileDesc>  <titleStmt>
 <title>Othello</title>
 <author>William Shakespeare</author>
```

Manipulating TEI

```
>>> tree = lxml.etree.parse('data/folger/xml/Oth.xml’)
```

prefix the tags with the specified prefix of the namespace

```
>>> print(tree.getroot().find('.//{http://www.tei-c.org/ns/1.0}title').text)
Othello
>>> print(tree.getroot().find('title’))   # fail without the prefix
None
```

```
>>> NSMAP = {'tei': 'http://www.tei-c.org/ns/1.0’}  # as a global variable
```

the prefix as one of the argument

```
>>> print(tree.getroot().find('.//tei:title', namespaces=NSMAP).text)
Othello
```

Overview

What is XML / HTML?

XML Elements

XML Attributes

XML Control Information

DTD

Examples

Corpora / Tools

Examples with XML

Examples with HTML

Manipulating HTML

As HTML is a cousin of XML, you can manipulate HTML documents with the BeautifulSoup package.

Some examples with a short document (Henry IV, Part I)

```
>>> import bs4 as bs
```

Then parse the document and use functions to extract part of the text

This is an alias for the package name

Manipulating HTML

```
>>> html_doc = """     # define the HTML document
<html> <head> <title>Henry IV, Part I</title> </head>
   <body> <div>
      <p class="speaker">KING</p>
      <p id="line-1.1.1"> <a id="ftln-0001">FTLN 0001</a>
         So shaken as we are, so wan with care,  </p>
      <p id="line-1.1.2"> <a id="ftln-0002">FTLN 0002</a>
         Find we a time for frighted peace to pant </p>
       <p id="line-1.1.3"> <a id="ftln-0003">FTLN 0003</a>
         And breathe short-winded accents of new broils </p>
      <p id="line-1.1.4"> <a id="ftln-0004">FTLN 0004</a>
         To be commenced in strands afar remote. </p>
      </div>
   </body> </html> """
```

Manipulating HTML

```
>>> import bs4 as bs                    # parse the document
>>> html = bs.BeautifulSoup(html_doc, 'html.parser’)
>>> print(html.title)                   # extract the entire title section (from head)
   <title>Henry IV, Part I</title>
>>> print(html.p)                       # print the first p section
   <p class="speaker">KING</p>
>>> print(html.title.text)              # extract the only the content of the title
  Henry IV, Part I
>>> print(html.p.parent.name)           # extract the parent tag name of the (first) p section
  div
```

Manipulating HTML

```
>>> print(html.p.parent)    # extract the parent of the (first) p section

<div> <p class="speaker">KING</p> <p id="line-1.1.1">
   <a id="ftln-0001">FTLN 0001</a>
         So shaken as we are, so wan with care,  </p> <p id="line-1.1.2">
<a id="ftln-0002">FTLN 0002</a>
         Find we a time for frighted peace to pant </p> <p id="line-1.1.3">
<a id="ftln-0003">FTLN 0003</a>
         And breathe short-winded accents of new broils </p>
<p id="line-1.1.4"> <a id="ftln-0004">FTLN 0004</a>
         To be commenced in strands afar remote. </p> </div>

>>> print(html.find_all('a'))  # print all p sections

[<a id="ftln-0001">FTLN 0001</a>, <a id="ftln-0002">FTLN 0002</a>,
<a id="ftln-0003">FTLN 0003</a>, <a id="ftln-0004">FTLN 0004</a>]

>>> print(html.find('p', id='line-1.1.3’))  # extract a specific section with ID
<p id="line-1.1.3"> <a id="ftln-0003">FTLN 0003</a>
         And breathe short-winded accents of new broils </p>
```

Manipulating HTML

```
>>> def html2txt(fpath):        # fpath: a string pointing to the filename
# Extract the text fo a HTML file and return a string.

    with open(fpath) as f:

        html = bs.BeautifulSoup(f, 'html.parser')

    return html.get_text()

>>> fp = 'data/folger/html/1H4.html'
>>> text = html2txt(fp)         # Return the text present in an HTML document
>>> start = text.find('Henry V')
>>> print(text[start:start + 500])

Henry V, Romeo and Juliet, and others. Editors choose which version to use
as their base text, and then amend that text with words, lines or speech
prefixes from the other versions that, in their judgment, make for a better
or more accurate text.

Other editorial decisions involve choices about whether  …
```

Manipulating HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html xmlns:html="http://www.w3.org/1999/xhtml"><head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Henry IV, Part I</title>
<meta name="author" content="Folger Shakespeare Library">
...</head>

<body oncopy="smartCopy();">...
<div id="contents" class="page div1">
<div class="frontHeader">Contents</div>
<table class="contents">  <tbody>
<tr>  <td class="act">Front Matter</td>  <td><ul class="nav">
<li class="scene"><a id="locFromTheDirector" href="#FromTheDirector"
class="scene">From the Director of the Folger Shakespeare Library</a></li>
<li class="scene"><a id="locTextualIntroduction" href="#TextualIntroduction"
class="scene">Textual Introduction</a></li>
<li class="scene"><a id="locsynops" href="#synopsis"
class="scene">Synopsis</a></li>
</ul></td>
```

Manipulating HTML

```
>>> with open(fp) as f:
...     html = bs.BeautifulSoup(f, 'html.parser’)
...

>>> toc = html.find('table', attrs={'class': 'contents'})

>>> toc    # the table of content (with the tags table, tr, td)
<table class="contents"><tbody>
<tr>
<td class="act">Front Matter</td>
<td><ul class="nav">
<li class="scene"><a class="scene" href="#FromTheDirector"
id="locFromTheDirector">From the Director of the Folger Shakespeare
Library</a></li>
<li class="scene"><a class="scene" href="#TextualIntroduction"
id="locTextualIntroduction">Textual Introduction</a></li>
<li class="scene"><a class="scene" href="#synopsis"
id="locsynops">Synopsis</a></li> …
```

Manipulating HTML

```
>>> def toc_hrefs(html):
    # Return a list of hrefs from a document's table of contents.

...     toc = html.find('table', attrs={'class': 'contents'})
...     hrefs = []     # In a list, return the hrefs
...     for tr in toc.find_all('tr'):
...         for td in tr.find_all('td'):
...             for a in td.find_all('a'):
...                 hrefs.append(a.get('href'))
...     return hrefs

>>> items = toc_hrefs(html)

>>> print(items[:5])    # the first five hrefs

['#FromTheDirector', '#TextualIntroduction', '#synopsis', '#characters',
'#line-1.1.0']
```

Manipulating HTML

```
>>> def get_href_div(html, href):
# Retrieve the <div> element corresponding to the given href.
...     href = href.lstrip('#')
...     div = html.find('div', attrs={'id': href})   # is it the ID?
...     if div is None:    # if not, the name
...         div = html.find('a', attrs={'name': href}).findNext('div')
...     return div
```

```
>>> def html2txt(fname, concatenate=False):
# Convert text from a HTML file into a string or sequence of strings.
...   with open(fname) as f:
...       html = bs.BeautifulSoup(f, 'html.parser')
...   texts = [get_href_div(html, href).get_text() for href in toc_hrefs(html)]
...   return '\n'.join(texts) if concatenate else texts
```

Manipulating HTML

```
>>> texts = html2txt(fp)     # Get the text of a play
>>> print(texts[6][:200])

Scene 3

Enter the King, Northumberland, Worcester, Hotspur,

and Sir Walter Blunt, with others.

KING , to Northumberland, Worcester, and Hotspur
FTLN 0332 My blood hath been too cold and tempera…


>>> import urllib.request   # to access HTML pages in the Internet

>>> page =
urllib.request.urlopen('https://en.wikipedia.org/wiki/William_Shakespeare')

>>> html = page.read()
```

Manipulating HTML

```
>>> import bs4   # as always with BeautifulSoup
>>> soup = bs4.BeautifulSoup(html, 'html.parser’)
>>> print(soup.get_text().strip()[:300])
William Shakespeare - Wikipedia
```

```
William Shakespeare
From Wikipedia, the free encyclopedia

Jump to navigation
Jump to search
English poet, playwright and actor


This article is about the poet and playwright. For other persons of the same
name, see William Shakespe
```

Manipulating HTML

```
>>> import re   # we need to remove non pertinent stuff (e.g., tags)

>>> for script in soup(['script', 'style']):
...     script.extract()

>>> text = soup.get_text()
>>> text = re.sub('\s*\n+\s*', '\n', text)   # remove multiple line breaks:
>>> print(text[:300])

William Shakespeare - Wikipedia
William Shakespeare
From Wikipedia, the free encyclopedia
Jump to navigation
Jump to search
English poet, playwright and actor
This article is about the poet and playwright. For other persons of the same
name, see William Shakespeare (disambiguation). For other uses
```

Manipulating HTML

```
>>> links = soup.find_all('a')    # extract all hyperlinks from the retrieved page

>>> print(links[9].prettify())    # links between web pages or inside the same page

   <a href="/wiki/Chandos_portrait" title="Chandos portrait">

    Chandos portrait

  </a>

>>>
```

References

DTD Tutorial	http://www.w3schools.com/dtd/default.asp
XML Tutorial	http://www.xmlfiles.com/xml/
	http://www.w3schools.com/xml/default.asp
HTML Tutorial	http://www.w3schools.com/html/default.asp
The Reference	http://www.w3.org/XML/