Harvard ‘Beyond TEI’ Workshop Cheatsheet

Sarah Lang

May 3, 2022

Contents

1 Preliminary Information

1.1 Digital Scholarly Editing

Is it a Digital Scholarly Edition?

1. **Is there a full representation** of the subject in question?
2. **Is it critical?** → processing rules stated and applied, scholarly knowledge included to make the document easier to understand, regarding material, document genesis/creation, context and reception?
3. **Is the edition of academic quality?** → transparent and rigorous edition process, responsibilities stated, enables future research on a reliable basis.
4. **Does the edition follow a digital paradigm?** → makes use of the possibilities of the digital, not printable without major loss of content or functionality.

1.2 XML & TEI

XML: eXtensible Markup Language

- W3Schools Tutorial
- paradigm of the separation of form and content
- XML is a metalanguage

.XML

- RSS, SOAP, XAML
- MathML, GraphML
- XHTML
- RDF
- KML
- Scalable Vector Graphics (SVG)

> “ **Extensible Markup Language (XML)** is a **markup language** and file format for storing, transmitting, and reconstructing arbitrary data. It defines a **set of rules for encoding documents** in a format that is **both human-readable and machine-readable.** (Wikipedia) ”

XML rules

XML can be checked for **validity** (validation if it complies with a standard) and **well-formedness** (following the rules of XML) → will only be parsed if well-formed. Thus: **Heed thy error messages!**

skip
There are rules on how elements can be named (you can look them up if relevant or will get informed by an error message).

`<key>value</key>` . . . . XML as a key value notation

Rules

- Hierarchical nesting below the root
- exactly one root element, i.e. one out-most russion doll
- start and end tag
- tag names are case-sensitive (!)
- empty elements allowed (& can be shortened)

Minimal example

```
<?xml version="1.0" ?>
<root>
  <element attribute="value">
    content
  </element>
  <!-- comment -->
</root>
```

XML rules

Prolog

`<xml version="1.0" encoding="utf-8">` . . . . XML declaration
`<?xsl-stylesheet type="text/xsl" href="my.xsl"?>` processing instructions (optional)

you can include document models (optional)
DTD, XML Schema, RelaxNG, Schematron

entities ‘protected’ characters that have a meta meaning in XML like:

`&lt;` . . . . <
`&gt;` . . . . >
`&amp;` . . . . &

Text Encoding Initiative (TEI)

.XML

XML-Standard, i.e. convention on how to use XML so that resulting data will be interoperable between different projects.
(founded in 1987, consortium since 2000)

> “ The Text Encoding Initiative (TEI) is a text-centric community of practice in the academic field of digital humanities, operating continuously since the 1980s. The community currently runs a mailing list, meetings and conference series, and maintains the TEI technical standard, a journal, a wiki, a GitHub repository and a toolchain. (Wikipedia) ”

TEI minimal example

```
<TEI> <!-- root element -->
    <teiHeader>
      <!-- author, title, dating,
           sources, edition rules, etc. -->
    </teiHeader>
    <text> ... </text>
</TEI>
```

Resources

- Learn TEI
- Teach Yourself
- P5 = 5. Proposal
- MEI for music
- CEI for charters
- http://www.tei-c.org/

TEI Header

fileDesc = bibliographical description of the contents of the document

encodingDesc = connection of electronic document to source

(i.e. transcription rules, etc.)

```
<TEI> <!-- root element -->
    <teiHeader>
        <fileDesc> ... </fileDesc> <!-- obligatory -->
        <encodingDesc> <!-- optional -->
        <profileDesc> <!-- optional -->
        <revisionDesc> <!-- optional -->
    </teiHeader>
    <text> ... </text>
</TEI>
```

profileDesc = decribes all non-bibliogaphical aspects of the text (i.e. creation, languages)

revisionDesc = tracks changes in the digital document

Using TEI

Gentle Intro to XML

TEI Core

- **div** (division)
- **p** (paragraph)
- **head** (heading)
- **lb** (linebreak)
- **pb** (page break / beginning)
- **hi** (highlight)
- **l** (line)
- **lg** (line group)
- **list**
- **item**
- **listBibl**
- **bibl** (bibliographical information)

Attributes

- **@n** (label)
- **@type** (typing)
- **xml:id** (unique identifier)
- **xml:lang** (language)
- **@rend** (rendering)
- **@ana** (interpretation)

Name spaces identified via URI

<prefix:name> e.g. `<tei:p>` ('I mean the `<p>` according to the TEI standard.')

declaration <element xmlns=“URI“> …
<prefix:element xmlns:prefix=“URI“> …
e.g.

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

How to find information on TEI elements

…and teach yourself how to use new elements:

- General TEI guidelines (XML Primer, Learn the TEI page, etc.)
- web-search TEI + (element you want to know about), i.e. “tei teiHeader” and you will get:
  1. definition page
  2. list of all examples for that element → directly over websearch or click 'show all' in the examples on the 'definitons page'
  3. sometimes even an module overview text for things as big as `<teiHeader>` (has its own module)

Relevant TEI modules

all All modules

5 Characters, Glyphs, and Writing Modes,

10 Manuscript Description,

11 Representation of Primary Sources,

12 Critical Apparatus,

13 Names, Dates, People, and Places.

Also: The TEI guidelines are documentation and reference, not necessarily ideal teaching tools → overwhelming. Maybe try other tutorials like the TEI by example page, for example the tutorials on **Primary Sources**

and **Critical Editing**

.

Oxygen tricks

- If the TEI schema is linked to your document and you have internet, you can hover over elements and click to be redirected to the relevant info page.
- If you open a tag (by just typing '<'), the editor will suggest a list of elements currently allowed where you're standing (for example, `<teiHeader>` is very picky about the sequence).

1.3 Metadata, datatypes, etc.

1.3.1 Annotation and other datastructures

Markdown

→ Markdown in 60s
simple text formatting

.md // presentational

`*Italic*`	*Italic*
`**Bold**`	**Bold**
`## Heading 1`	Heading 1
`#### Heading 2`	etc.
`[Link](http://a.com)`	'hidden' link
`![Image](http://url/a.png`	image
`> Blockquote`	quote
`- List`	list (can be nested, 2 spaces)
`* List`	alternative list
`1. enumerate`	numbered list
`--`	separator line
`‘Inline code‘`	Code ('backticks')
`‘‘‘code block‘‘‘`	code block

LaTeX

- .tex Typesetting with TeX using Lamport macros (i.e. shortcuts to make complicated code easy)
- LaTeX reads in those macros (with telling names like `\emph{}` for 'emphasis', an 'intelligent' command which can be redefined for the whole document)
- feels like markup for those producing the code (illusion of descriptive markup)
- placeholder for complex procedural language
- **WYIWYG**-Editors (*what you see is what you get*, i.e. MS Word) vs. **WYSIWYM** (*what you see is what you mean* → LaTeX)

Example commands

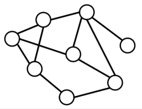
Commands below without the spaces

`\textit {Italic}`	*Italic* (presentational.)
`\emph {Italic}`	*Italic* (semantic/'intelligent')
`\textbf {Bold}`	**bold face**
`\section {Title}`	Heading 1
`\subsection {Subtitle}`	etc.
`\href {http://a.com}{Link}`	'hidden' link
`\includegraphics {bla.png}`	image

Data structures: Graphs

Applications

- **Resource Description Framework (RDF):**
  - e.g. Blazegraph (Graph-DB)
  - Query: SPARQL
- Labeled Property Graphs
  - e.g. Neo4j (Graph-DB)
  - Query: Cypher



RDF/Turtle-Notation (`.ttl`)

```
@prefix ex: <http://example.com/#> .

ex:Graz a ex:city;
ex:name "Graz" ;
ex:inhabitants 288806 ;
ex:location [ ex:lat 47.4; ex:long 5.26 ] .

ex:Wien a ex:city;
ex:name "Wien" ;
ex:inhabitants 1897491 ;
ex:location [ ex:lat 47.12; ex:long 16.22 ] .
```

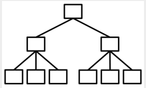
SPARQL query

```
@prefix ex: <http://example.com/#> .

SELECT ?name, ?population
WHERE {
    ?city ex:inhabitants ?population .
}
```

Data structures: tree hierarchy 1 / JSON

Applications: **JavaScript Object Notation (JSON)**

- e.g. MongoDB
- no standardized query language, just JavaScript (`.js`)



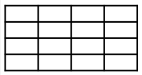
JSON & JS (w3s)

```
'{"name": "John", "age": 30, "car": null,
  "tree" : [
    "key": "value"
  ]
}'
const obj = JSON.parse('{"name":"John", "age":30,
                         "city":"New York"}');
obj.age = obj.age.toString();
```

Data structures: tabular data / relational databases

Applications

- e.g. SQLite, MySQL (relational dbs)
- Query: SQL



SQL query

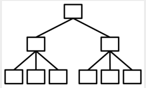
```
CREATE TABLE "places" (
 "name" TEXT,
 "population" INTEGER,
 "longitude" REAL,
 "latitude" REAL,
 PRIMARY KEY("name")
);

INSERT INTO places
VALUES
('Graz', 288806, 47.066667, 15.433333),
('Wien', 1897491, 48.208174, 16.373819);
```

(name, population, longitude, latitude)

Data structures: tree hierarchy 2 / XML

Applications: **eXtensible Markup Language (XML):**

- DBs: eXist, BaseX
- Query: XPath (w3s) and XQuery (w3s)

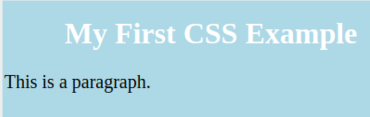


```
<!-- books.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>
</bookstore>

<!-- XQuery -->
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title

<!-- XPath -->
//title[@lang='en']
/bookstore/book[price>35.00]
```

Data structures: tree hierarchy 3 / web pages (HTML)

HTML (w3s) – structure

```
<!DOCTYPE html>
<html>
 <head>
   <title>Page Title</title>
 </head>
 <body>
  <h1>This is a Heading</h1>
  <p>This is a paragraph.</p>
 </body>
</html>
```



My First CSS Example

This is a paragraph.

CSS in HTML (w3s) – rendering

```
<!DOCTYPE html>
<html>
  <head>
    <style>
body {
  background-color: lightblue;
}
h1 {
  color: white;
  text-align: center;
}
p {
  font-family: verdana;
  font-size: 20px;
}
    </style>
  </head>
  <body>
    <h1>My First CSS Example</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

A primer on metadata

What are metadata?

- „data about data“
  1. data about containers of data = **structural metadata**
  2. data about the content represented by data = **descriptive metadata**
- functions:
  1. descriptive
  2. administrative
  3. technical
  4. use

There are standards for the description of metadata (and many are XML-based), e.g.

- Machine-Readable Cataloging (**MARC**)
- Metadata Object Description Schema (**MODS**)
- Encoded Archival Description (**EAD**)
- Lightweight Information Describing Objects (**LIDO**)
- Collective Description of Works of Art (**CDWA**) / Visual Research Association (**VRA**)
- Europeana Metadata Model (**EDM**)
- Resource Description Framework (**RDF**)
- Metadata Encoding & Transmission Standard (**METS**)
- Dublin Core (**DC**)
- Functional Requirements for Bibliographic Records (**FRBR**)
- the `<teiHeader>` has metadata…

Slides on metadata in DH with more information

1.3.2 Metadata standards

Dublin Core (DC)

What is the DC?

- founded in Dublin (Ohio) in 1995
- **two levels:** simple (15 elements) & qualified (additional `Audience`, `Provenance` and `RightsHolder`)
- **classes of terms:** elements (nouns) & qualifiers (adjectives).
- can be expressed in RDF/XML
- each element is optional & can be repeated
- also: dc:terms

> ❝ **The Dublin Core™ metadata standard** is a simple yet effective **element set for describing a wide range of networked resources.** [...] Another way to look at Dublin Core™ is as a "small language for making a particular class of statements about resources". In this language, there are two classes of terms – *elements* (nouns) and *qualifiers* (adjectives) – which can be arranged into a simple pattern of statements.
>
> (source) ❞

Core + Qualified

i.e. the elements:

1. title
2. subject
3. description
4. type
5. source
6. relation
7. coverage
8. creator
9. publisher
10. contributor
11. rights
12. date
13. format
14. identifier
15. language
16. (audience)
17. (provenance)
18. (rights holder)

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

   <rdf:Description rdf:about="http://media.example.com
                             /audio/guide.ra">
      <dc:creator>Rose Bush</dc:creator>
      <dc:title>A Guide to Growing Roses</dc:title>
      <dc:description>Describes process for
        planting and nurturing different kinds
        of rose bushes.</dc:description>
      <dc:date>2001-01-20</dc:date>
   </rdf:Description>
</rdf:RDF>
```

Note the two namespaces `rdf:` and `dc:`.

Metadata Encoding and Transmission Standard

METS

- tool for encoding digital library objects
- container format for documents in which contents of different formats can be integrated
- also describes relationships between objects
- describes logical and physical structure of an object
- also contains descriptive (bibliographical) and administrative metadata
- relatively simple and straightforward
- supports a wide range of materials
- <website> & more info here

Only structural map is required.

```
<mets>
  <metsHdr/>
  <dmdSec/>
  <amdSec/>
  <fileSec/>
  <structMap/>
  <structLink/>
  <behaviorSec/>
</mets>
```

Resource Description Framework (RDF)

RDF

- framework for describing resources in the World Wide Web
- can contain metadata
- language of the 'Semantic Web' (web 3.0) – makes things machine-processable
- RDF Schema (RDFS) offers Classes and Properties

RDF/ turtle notation (`.ttl`) example from before

```
@prefix ex: <http://example.com/#> .

ex:Graz a ex:city;
ex:name "Graz" ;
ex:inhabitants 288806 ;
ex:location [ ex:lat 47.4; ex:long 5.26 ] .
```

Metadata Object Description Schema (MODS)

MODS

- can represent the major elements from a MARC record
- → represents key bibliographic data in easily understandable names
- → easier to understand than MARC (for the uninitiated)
- bridges the gap between library application and bibliographic source that don't make use of cataloging metadata formats
- richer than the Dublin Core (DC) but less detailed than MARC
- partially backwards compatible with MARC

```
<mods:mods xmlns:mods="http://www.loc.gov/mods/v3">
  <mods:titleInfo>
    <mods:nonSort>The </mods:nonSort>
    <mods:title>
      1946 Library of Congress recital
    </mods:title>
  </mods:titleInfo>
  <mods:relatedItem type="constituent"
                    ID="DMD_disc01_tr001">
    <mods:titleInfo type="uniform">
      <mods:partName>Chaconne von Vitali
        </mods:partName>
    </mods:titleInfo>
  </mods:relatedItem>
  <mods:identifier type="lccn">99594334
    </mods:identifier>
</mods:mods>
```

Code example from here.

Encoded Archival Description (EAD)

EAD

is a standard for encoding descriptive information regarding archival records

example EAD & Wikipedia (source of the example)

```
<eadheader>
   <eadid countrycode="us" identifier="bachrach_lf">
     bachrach_lf</eadid>
   <filedesc>
      <titlestmt>
         <titleproper encodinganalog="Title">
          Louis Fabian Bachrach Papers</titleproper>
         <subtitle>An inventory of his papers at
           Blank University</subtitle>
         <author encodinganalog="Creator">Mary Smith</author>
      </titlestmt>
      <publicationstmt>
         <publisher encodinganalog="Publisher">
           Blank University</publisher>
         <date encodinganalog="Date" normal="1981">
           1981</date>
      </publicationstmt>
   </filedesc>
   <profiledesc>
      <creation>John Jones
         <date normal="2006-09-13">13 Sep 2006</date>
      </creation>
   </profiledesc>
</eadheader>
```

Charters Encoding Initiative (CEI)

CEI

considers the possibilities of a standard to encode medieval and early modern charters with XML.

- implements the *Vocabulaire Internationale de Diplomatique*
- founded in 2004
- MOM-CA, the collaborative charter archive of Monasterium.net works with CEI
- <website> & an example. Also: the example below.

```
<text type="charter">
  <idno>600202b</idno>
  <chDesc id="a">
    <head>Prag, 1360 Febr. 2.</head>
      <issued>
         <placeName>Prag</placeName>
         <date>1360-02-02</date>
      </issued>
      <abstract>
         <p> Karl verspricht Ludwig ...
         </p>
      </abstract>
      <witList>
         <witness sigil="B">
            Brandenburgisches LHA Potsdam
            ‘Rep. 37 Hohennauen Nr. 683,
            fol. 225’’ (18. Jh.)
         </witness>
      </witList>
      <diplomaticAnalysis>
         <bibl type="D">Fidicin, 42...</bibl>
      </diplomaticAnalysis>
   </chDesc> <tenor> </tenor>
</text>
```

Simple Knowledge Organization System (SKOS)

SKOS

RDF vocabulary for representing semi-formal *knowledge organization systems* (KOSs), such as thesauri, taxonomies, classification schemes and subject heading lists.
→ less rigorous than the logical formalism of ontology languages such as OWL

(SKOS primer)

```
@prefix skos:
  <http://www.w3.org/2004/02/skos/core#> .
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://www.example.com/> .
@prefix ex1: <http://www.example.com/1/> .
@prefix ex2: <http://www.example.com/2/> .

ex:animals rdf:type skos:Concept;
  skos:prefLabel "animals"@en;
  skos:narrower ex:mammals.

ex:mammals rdf:type skos:Concept;
  skos:prefLabel "mammals"@en;
  skos:broader ex:animals.
```

Europeana Data Model (EDM)

EDM

Model to integrate data sources from different providers and thus improve interoperability:

> “EDM transcends domain-specific metadata standards, yet accommodates the range and richness of community standards such as LIDO for museums, EAD for archives or METS for digital libraries.” (EDM Factsheet)

…integrates the following standards:

1. **OAI ORE** (Open Archives Initiative Object Reuse & Exchange) for organizing an object's metadata and digital representation(s)
2. **Dublin Core** for descriptive metadata
3. **SKOS** (Simple Knowledge Organization System) for conceptual vocabulary representation
4. **CIDOC-CRM** for event and relationships between objects

This is achieved in RDF (can be written as XML) → RDF uses the Semantic Web principles to integrate those data sources →

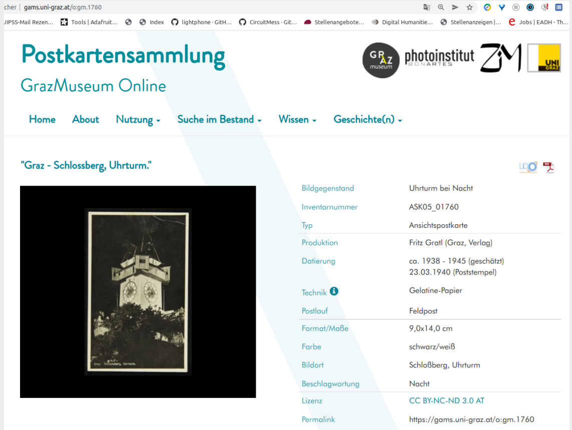
Metadata standards aren't exclusive, they can be combined!

Lightweight Information Describing Objects

LIDO

> Lightweight Information Describing Objects (LIDO) is an **XML schema for describing museum or collection objects.** Memory institutions use LIDO for “exposing, sharing and connecting data on the web”. It can be applied to all kind of disciplines in cultural heritage, e.g. art, natural history, technology, etc.
>
> LIDO is a specific application of CIDOC CRM. (Wikipedia)

A postcard (text-bearing object)


cher | gams.uni-graz.at/o:gm.1760
JIPSS-Mail Rezen... Tools | Adafruit... Index lightphone · GitH... CircuitMess · Git... Stellenangebote... Digital Humanitie... Stellenanzeigen |... Jobs | EADH - Th...
GRAZ museum
photoinstitut BONARTES
ZiM
UNI GRAZ
Home
About
Nutzung
Suche im Bestand
Wissen
Geschichte(n)

Postkartensammlung

GrazMuseum Online

"Graz - Schlossberg, Uhrturm."

Bildgegenstand	Uhrturm bei Nacht
Inventarnummer	ASK05_01760
Typ	Ansichtspostkarte
Produktion	Fritz Gratl (Graz, Verlag)
Datierung	ca. 1938 - 1945 (geschätzt) 23.03.1940 (Poststempel)
Technik	Gelatine-Papier
Postlauf	Feldpost
Format/Maße	9,0x14,0 cm
Farbe	schwarz/weiß
Bildort	Schloßberg, Uhrturm
Beschlagwortung	Nacht
Lizenz	CC BY-NC-ND 3.0 AT
Permalink	https://gams.uni-graz.at/o:gm.1760

```
<lido:titleWrap>
    <lido:titleSet>
        <lido:appellationValue>
            Chickens and Ducks
        </lido:appellationValue>
    </lido:titleSet>
</lido:titleWrap>
```

LIDO example

Notice the two namespaces (`lido:` and `t:`)!

```
<lido:lido
  xmlns:lido="http://www.lido-schema.org"
  xmlns:t="http://www.tei-c.org/ns/1.0">
  <lido:lidoRecID lido:type="PID">o:gm.1760</lido:lidoRecID>
  <lido:category>
    <lido:conceptID lido:source="CIDOC"
                    lido:type="ID">E22</lido:conceptID>
    <lido:term xml:lang="eng">Man-Made Object</lido:term>
    <lido:term lido:label="info:fedora/context:gm">
      Postkartensammlung Online</lido:term>
  </lido:category>
  <lido:descriptiveMetadata xml:lang="deu">
    <lido:objectClassificationWrap>
      <lido:objectWorkTypeWrap>
        <lido:objectWorkType>
          <lido:conceptID lido:source="http://vocab.getty.edu/aat"
            lido:type="ID">300026819</lido:conceptID>
          <lido:term lido:label="info:fedora/context:gm-ansicht">
            Ansichtspostkarte</lido:term>
        </lido:objectWorkType>
      </lido:objectWorkTypeWrap>
    </lido:objectClassificationWrap>
    <lido:objectIdentificationWrap>
      <lido:titleWrap>
        <lido:titleSet>
          <lido:appellationValue>Graz - Schlossberg, Uhrturm.
             </lido:appellationValue>
        </lido:titleSet>
      </lido:titleWrap>
      <lido:repositoryWrap>
        <lido:repositorySet lido:type="orgname">
          <lido:repositoryName>
            <lido:legalBodyID lido:source="http://d-nb.info/gnd"
              lido:type="ID">2022740-1</lido:legalBodyID>
            <lido:legalBodyName>
              <lido:appellationValue>GrazMuseum</lido:appellationValue>
            </lido:legalBodyName>
          </lido:repositoryName>

    <!-- to be continued... LIDO is very verbose --->
```

1.4 Web Dev Triad (HTML, CSS, JS)

Hyper Text Markup Language (HTML)

.HTML

Defines the structure of websites. Due to their common origin in SGML: lots of similarity with XML but, unlike XML's extensibility, HTML has a fixed tag set (much less!).

The most important HTML elements to remember html, head, body. div, p, h1-6. span, ul, ol, li. table, tr, td.

```
<!DOCTYPE html>
<html>
 <head>
  <title>Page Title</title>
  <style> /* better in extra file */
  h1 {
   color: blue;
   font-family: verdana;
   font-size: 300%;
   }
   p  {
    color: red;
    font-family: courier;
    font-size: 160%;
    }
   p.important {
    color: green;
    }
  </style>
  <script>
    alert("Hello World!");
   </script>
 </head>
 <body>
  <h1>This is a Heading</h1>
  <p>This is a paragraph. <br />
    <a href="https://www.w3schools.com">
    This is a link</a>
  </p>
  <img src="img.jpg"
    width="500" height="600" />
  <p style="color:red">I am a paragraph</p>
  <p title="I'm a tooltip">
    This is a paragraph.
  </p>
  <p>My mother has
     <span style="color:blue">blue</span>
     eyes.</p>
  <p class="important">
    Note that this is an important
    paragraph. :)</p>
 </body>
</html>
```

CSS

CSS = Cascading Style Sheets

.CSS

- included via a stylesheet link in HTML or written inline
- describes the styling of websites
- separation of form & content:
  - **HTML:** content in structured form
  - **CSS:** the layout
  - (**JavaScript:** dynamic parts)
- the **Bootstrap framework** offers a ready-made, responsive design to reuse: box and grid model
- uses selectors to define layout

```
h1 {
font-family: Arial
}

p {
font-family: Arial ;
color: red
}

.person {
    font-weight:bold;
    font-size:smaller;
    font-style:italic;
}
```

Resources

- 15min to Bootstrap
- CSS3-Cheatsheet
- **W3 Schools:** W3schools CSS
- W3S Tables of properties
- CSS ZenGarden

JavaScript

modifying websites dynamically

.JS

- JS can change websites without having to reload them ('dynamically'). remember when you had to manually refresh pages?
- When we want to highlight something using a check box, we use js.
- Unlike HTMl (which is a markup language, think of it like a file format), js is a programming language (like XSLT).
- **→ we won't learn this today!**

  But there is a mini example and you will load js components when using Bootstrap and the GAMS wippets later.

A 'Hello World!' example

```
document.getElementById("demo").
   innerHTML = "Hello JavaScript";
```

1.5 XPath

XPath

axes

`/`	root node / one step further down the tree
`::*`	::* = all types of elements irrespective of name
`@`	attribute
`[number]`	rank in result set
`text()`	text content of element
`::elementtype`	condition for element type (node test)
`//`	at any depth
`[text() = 'text content']`	condition (predicate)
`.`	self
`(/../`	jump one hierarchical level regardless of name
`path//subpath`	arbitrary depth

Conditions which define subsets = predicates

```
//person[@gender="female"]
//person[firstname = "Stefan"]
//person[firstname != "Tanja"]
//person[1]
//person[last()]/firstname
//person[position() = 2]
/participants/person[position()=5]/firstname
/participants/person[last()]/firstname
count(child::*)
```

XPath functions

bgalertwhitefunctions – set of nodes

`position()`	position of current node, return: number
`count()`	number of nodes in given node set
`last()`	last node in selected node set

string functions

`substring-before(value, substring)`	substring of text node (*value*) coming before the indicated *substring*
`substring-before(., 'mina')`	e.g.
`substring-after(value, substring)`	like before
`substring(value, start, length)`	with start pos & length
`string-length(.)`	length of current text node
`concat(value1, value2, ...)`	concatenate
`concat(surname, ', ', forename)`	e.g.
`translate(target, string, replacement)`	replace
`normalize-space(target)`	removes trailing whitespace

Boolean functions

`starts-with(value, substring)`	return: true/false
`starts-with(., 'D')`	true/false
`contains(value, substring)`	if the current text node contains the string, returns yes
`not(comparison)`	returns yes if not XY

Functions with regex support

`matches(target, 'RegEx')`	like `contains()` but only exact matches
`replace()`	like `translate()` but supports regex
`tokenize()`	tokenize text into single words

Troubleshooting

1. **[Nothing works]** In case of doubt it's always the namespace! Or you're somewhere other than you expected (using relative paths) → try an absolute path!
2. **[Nothing found]** XML is **hierarchical**! Am I in the right spot? Can I reach the degree of **depth** that I wanted? In doubt, include `/../` to be more independent of hierarchy.
3. **[Not found enough?] Jumps in path depth:** A path (without `path//subpath`) in between will not jump any levels, only go one step deeper! Are there in-between layers you forgot? Like another nested `<div>`? Try putting `//` but then, depth is completely arbitrary (maybe also not the wanted outcome) – if you want a particular depth, like 3, try `/../../../`.
4. **[Path works but doesn't do what I wanted]** Formulate your query as precise as possible. Do I find all I want? Does my query match more than I wanted?
5. XSL's standard processing behaviour causes text content to be printed as is unless it's explicitly changed or suppressed.

Preparing XSL transformations

First query all elements used in your file (once each using the `distinct-values()` function) to be sure that you have everything covered or learn what you need to write template rules for.

`distinct-values(//@*/name())`	What types of attributes are there?
`distinct-values(//@*)`	What attribute values are in my document?
`distinct-values(//@rend)`	What values does `@rend` take?
`distinct-values(//name())`	What elements are there?

1.6 XSLT

XSLT

XSL (*eXtensible Stylesheet Language*) is a programming language for transforming XML data. This allows for the storage of presentation-independent base data from which different representations can be generated (i.e. structure information for HTML).

This is called the ***single source principle.***

XSLT (=XSL-Transformations) is often used as a synonym but actually is only one part like **XSL-FO** (*Formatting Objects* for describing print layouts in PDF; discontinued since 2012).

XSLT is made up of a set of processing rules (*templates*) according to which an XML document is traversed and processed. The rules are being applied whenever one matches the current node/-content.

variables

```
<xsl:variable name="substitute" select="content">
```

XSLT handles variable with a so-called *pass by value*, not *by reference*. Ergo only the value is passed that the element has at the moment the variable is created. If I define it too early, I might get the wrong value!

attributes

`<xsl:attribute>` is used to set attributes in the output document. It's not always necessary: Often you can use the shorthand `type={@rend}`.

plain text

`<xsl:text>` puts the text as it is, i.e. won't put in unwanted spaces (like XSL tends to do which can be a problem in LaTeX where space is significant.

XSL will be very liberal with spacing, unless you explicitly suppress this using `preserve-space`.

sorting

There is `<xsl:sort>` for sorting values.

conditions

`<xsl:choose>`

lets you choose a different way of processing according to a

test using `<xsl:when>`

. `<xsl:if>`

only does something when a condition is met (i.e. if an element doesn't exist or has content XY). You should use this to check if optional elements exist to avoid errors!

Person list in HTML using `for-each`

```
<ul> <!-- unordered in HTML -->
<xsl:for-each select="t:persName">
    <li> <!-- list element -->
    <xsl:value-of select="." />
    </li>
</xsl:for-each>
</ul>
```

Setting attributes

```
<xsl:for-each select="t:persName">
    <span class={@rend}><xsl:value-of select="."></span>
</xsl:for-each>
<!-- OR -->

<span>
    <xsl:attribute name="id">
        <xsl:value-of select="@rend">
        <xsl:text>-person</xsl:text>
    <xsl:attribute>
    <xsl:attribute name="interpretation">
        <xsl:value-of select="concat(@rend,@ana)">
    <xsl:attribute>
    <xsl:apply-templates /> <!-- for the content -->
<span>
<!-- result e.g.
    <span id="monster-person" interpretation="monster-evil">
    Weird Sisters </span> -->
```

Merging multiple documents with (`document()`)

```
<xsl:apply-templates
  select="document('Letter1_TEI.xml')/tei:TEI//tei:body/tei:div"/>
```

Mode

```
<xsl:apply templates select="//t:body" />
<xsl:apply-templates select="head" mode="toc"/>
```

Loops: `for-each`

Example: Get each person (`//persName`) and generate a listing (`<ul>`) of the last names (`lastname`):

```
<ul>
<xsl:for-each select="//persName">
    <xsl:sort select="lastname" order="ascending" />
    <li> <xsl:value-of select="lastname"/> </li>
</xsl:for-each>
</ul>
```

Conditions I: if

`xsl:if` only runs if condition in `@test` evaluates as 'true':

```
<xsl:if test=" xpath-ausdruck "> ... </xsl:if>

<xsl:for-each select="//book">
    <xsl:if test=" author = 'Cicero' ">
        <li><xsl:value-of select="title"/></li>
    </xsl:if>
</xsl:for-each>
```

Conditions II: choose

You can also differentiate a number of cases:

```
<xsl:choose>
    <xsl:when test="some xpath"> ... </xsl:when>
    <xsl:otherwise> ... </xsl:otherwise>
</xsl:choose>
```

You can also sort (`xsl:sort`), copy (`xsl:copy - xsl:copy-of`) and use variables.