Lecture 4: Handling XML

PCL II, CL, UZH
March 16, 2016

Universität Zürich[UZH]

Contents

- XML
  - History and Purpose
  - XML Fundamentals
  - Well-formedness and validity
- XHTML
- Auxiliary technologies
- XML parser
- XML libraries in Python

```
<?xml version="1.0" encoding="UTF-8" ?>
<cichlids>
    <cichlid ID="c1">
        <name>Zeus</name>
        <color>gold</color>
        <teeth>molariform</teeth>
        <breeding-type>lekking</breeding-type>
    </cichlid>
</cichlids>
```

XML: History

- 1970s: Standard Generalized Markup Language (SGML) developed by IBM
  http://www.w3.org/MarkUp/SGML/
- 1986: SGML becomes standard (ISO 8879)
- 1989: First ideas towards HTML at CERN (Tim Berners-Lee)
- 1994: Berners-Lee founds World Wide Web Consortium (W3C)
- 1996: First work on eXtensible Markup Language (XML)
- 1998: Specification of XML 1.0
- Since then: only slight revisions

What is XML?

- E**X**tensible **M**arkup **L**anguage (XML)
- Subset of SGML
- Purpose/Design:
  - Designed to store and transport data
  - Human- and machine-readable
  - Rigorous rules
- **Meta** Markup language
  - does not represent a defined set of tags and elements available, unlike HTML
  - Not concerned with layout or presentation
- platform independent
- Many XML-based specifications: XHTML, SVG, RSS, EPUB, XSLT, RDF ...

What is XML?

- XML vocabulary is not predefined
  - author can define the content and document structure
- XML describes data
  - no information on how data will be presented
- XML documents form a tree structure
  - root element (= parent of all other elements)
  - branches
  - leaves/child nodes

XML concepts

<?xml version="1.0" encoding="UTF-8" ?>

- XML declaration/ XML prolog
  - optional
  - **must** be first line of the document
  - Specify encoding used:
    - `<?xml version="1.0" encoding="UTF-8"?>`
    - UTF-8 is the default character encoding

XML concepts

<?xml version="1.0" encoding="UTF-8" ?>

<node

- XML declaration
- Start tag/Opening tag

XML concepts

```
<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >
```

- XML declaration
- Start tag/Opening tag
- Attribute(s)
  - Attribute values must be quoted (single quotes or double quotes)

XML concepts

```
<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >
    <child>...</child>
```

- XML declaration
- Start tag/Opening tag
- Attribute(s), attribute values
- Child nodes (hierarchy)

XML concepts

```
<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >
    <child>...</child>
</node>
```

- XML declaration
- Start tag/Opening Tag
- Attribute(s)
- Child nodes (hierarchy)
- End tag/Closing Tag
  - All nodes must have a closing tag
  - tags are case sensitive

XML concepts

```
<?xml version="1.0" encoding="UTF-8" ?>
<node attr1="value" attr2="value2" >
    <child>...</child>
</node>
```

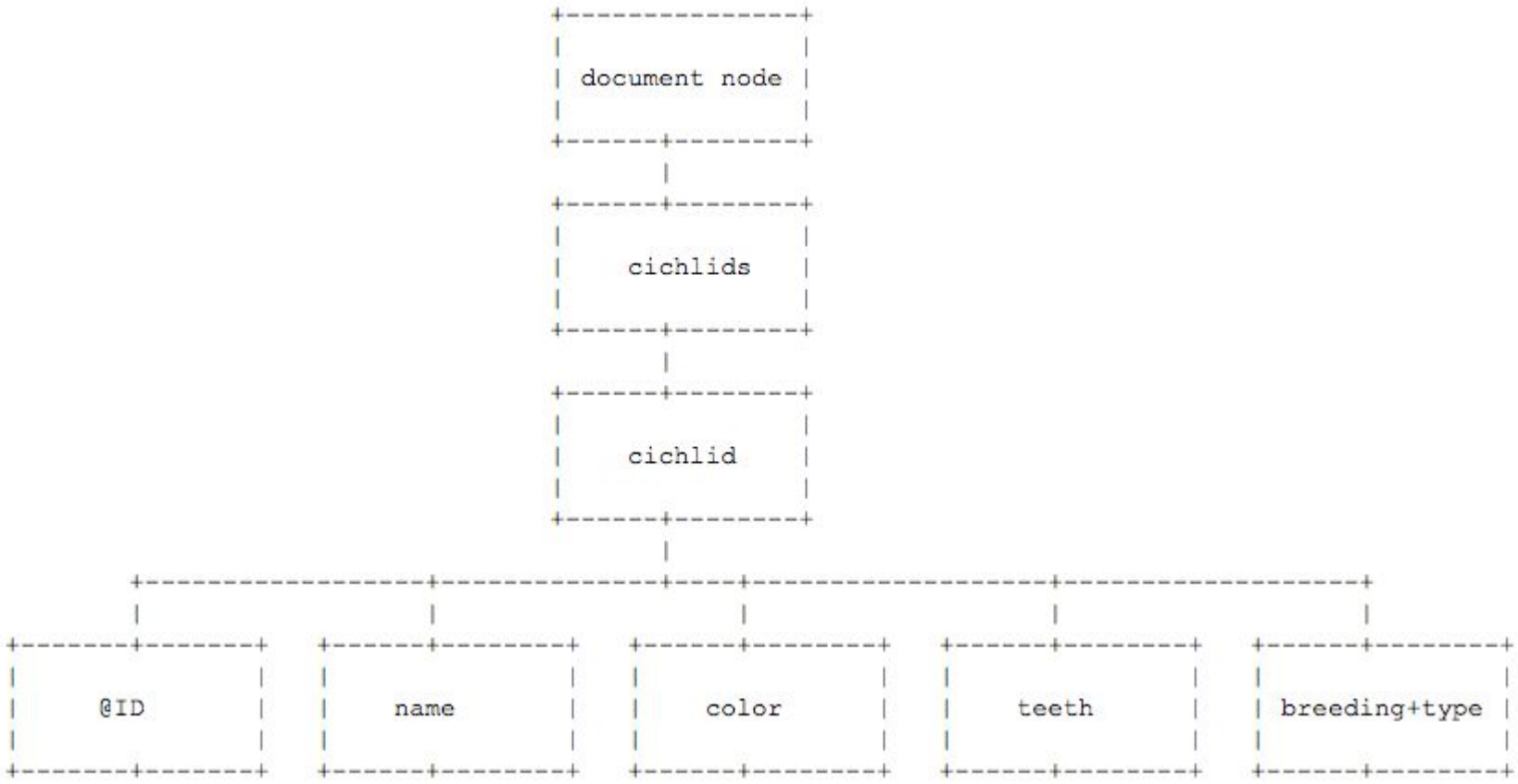
More concepts:

- element
- root element
- Entities (don’t worry about them - until you have to)
- XML tree structure

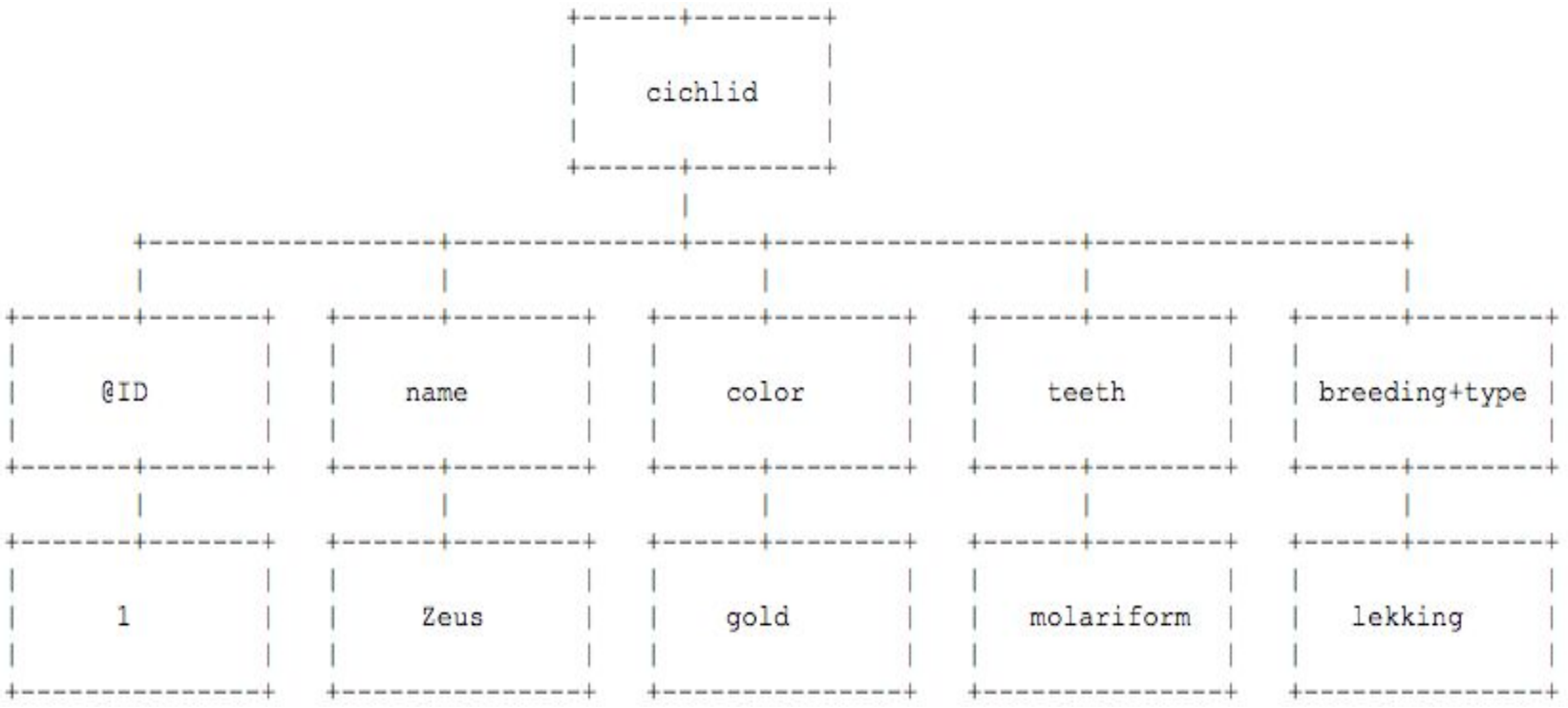
What do you mean by tree structure?

```
<?xml version="1.0" encoding="UTF-8" ?>
<cichlids>
    <cichlid ID="c1">
        <name>Zeus</name>
        <color>gold</color>
        <teeth>molariform</teeth>
        <breeding-type>lekking</breeding-type>
    </cichlid>
</cichlids>
```

What do you mean by tree structure?



What do you mean by tree structure?



XML elements

- Every XML element can contain:
  - Text
  - Attributes
  - Other elements
  - A mix of the above
- Empty XML elements:
  - Elements without content:

    <element></element>

    <element />
  - Empty elements can still have attributes

XML elements: naming rules and conventions

Naming rules:

- any name can be used (except xml)
  - cannot even start with the letters xml
- names are case-sensitive
- names must start with a letter or underscore
- names cannot contain spaces
  - can contain letters, digits, hyphens, underscores, and periods

Naming conventions:

- descriptive names
- short and simple names

Comments in XML

<!-- This is a comment -->

- Allowed in a comment: everything except --
- Do not put data inside comments

Characters with special meaning in XML

- An XML document can contain any character, except & and <
- Why?

Wrong:

<formula> x < y & x != 2 </formula>

Right:

<formula> x < y & x != 2 </formula>

Checking XML documents

- Well-formedness (“syntactic correctness”)
- Validity (“semantic correctness”)

Well-formedness of XML

Most aspects of well-formedness:

- XML declaration must be on the first line of the document
- Every start tag must have a corresponding end tag
- Tags names must begin with a letter or underscore
- Elements must be properly nested
- There must be a single root element
- Attribute values must be quoted
- An element must not contain two attributes with the same name
- Comments and processing instructions within tags are not allowed

Validity of XML

- Optional, many XML documents are never validated
- Schema languages:
  - DTD (old!)
  - XML Schema
  - Relax NG
  - Schematron
  - (...)
- Content of schema: rules
- (demo)

Contents

- XML
- XHTML
- Auxiliary technologies
- XML parser
- XML libraries in Python

XHTML (skip perhaps)

XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage

- Variant of HTML 4.0 (and HTML 5)
- **Restricted syntax** to well-formed XML
- Conditions:
  - Closing tag (</p> etc.) must be present
  - Nesting place overlaps
  - Quotation marks to attribute values
  - Only one root element (usually <html>)
  - Empty elements such as <hr> must be <hr />
  - elements and attribute names must be in lowercase

XHTML

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
      <head>
            <title>Beispiel</title>
      </head>
      <body>
            <h1>Beispielseite</h1>
            <p>Ein Absatz</p>
            <p>Noch ein<br/>Absatz</p>
            <ol>
                  <li>Listelement</li>
                  <li>Listelement</li>
            </ol>
            <p>
                  <img src="bild.jpeg" alt="Bildmotiv" />
            </p>
      </body>
</html>
```

XHTML

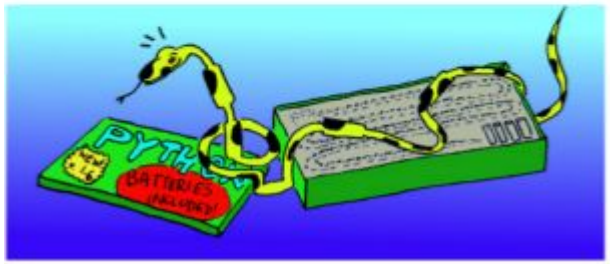
```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
      <head>
            <title>Beispiel</title>
      </head>
      <body>
            <h1>Beispielseite</h1>
            <p>Ein Absatz</p>
            <p>Noch ein<br/>Absatz</p>
            <ol>
                  <li>Listelement</li>
                  <li>Listelement</li>
            </ol>
            <p>
                  <img src="bild.jpeg" alt="Bildmotiv" />
            </p>
      </body>
</html>
```

Beispielseite

Ein Absatz

Noch ein
Absatz

1. Listelement
2. Listelement



Contents

- XML
- XML technologies
  - XPath
  - XSLT
  - XSL-FO
- XML parser
- XML libraries in Python
  - parse XML document
  - create/modify XML document

Even more auxiliary technologies

- **XPath**
- **XSLT**
- XQuery
- XForms
- XLink
- XPointer
- XInclude
- XML Schema
- ...

What is XSL-FO?

- E**X**tensible **S**tylesheet **L**anguage (XSL) - Formatting Objects
- Describes how an XML document should be rendered, e.g. as PDF
- XSL-FO documents for XML are comparable with CSS style sheets for HTML

What is XPath?

- To **navigate** XML documents and select content from them
- Embedded into many XML technologies, as a sublanguage
  - XSLT
  - XQuery
  - XLink / XPointer
  - any popular XML library, in many programming languages

- XPath navigates the XML tree structure
- Important concepts: expression, step, axis, predicate

XPath fundamentals

```
<?xml version="1.0" encoding="UTF-8" ?>
<cichlids>
    <cichlid ID="c1">
        <name>Zeus</name>
        <color>gold</color>
        <teeth>molariform</teeth>
        <breeding-type>lekking</breeding-type>
    </cichlid>
</cichlids>
```

XPath fundamentals

```
<?xml version="1.0" encoding="UTF-8" ?>
<cichlids>
        <cichlid ID="c1">
            <name>Zeus</name>
            <color>gold</color>
            <teeth>molariform</teeth>
            <breeding-type>lekking</breeding-type>
        </cichlid>
</cichlids>
```

Example 1: /cichlids/cichlid/name

Example 2: //cichlid[@ID=’c1’]

What is XSLT?

- A language that **transforms** XML documents
- (external slides)
- (Demo)

XSLT processors:

- Saxon (best one, free version)
- MSXSL
- Exselt
- (online tools)
- Libraries, e.g. libxsl

Quiz (only if we have time)

http://www.w3schools.com/quiztest/quiztest.asp?qtest=XML

Contents

- XML
- XHTML
- XML technologies
- XML parser
- XML libraries in Python

XML parser

- Checks for well-formedness, possibly for validity
- XML parser in most browsers, e.g. to check for well-formedness
- Unlike a well-formedness error, a validation error is not necessarily fatal

Types of XML parsers

- Event based (read as you go → streaming)
  - Advantage: speed, efficiency → does not store everything at once
  - Disadvantage: less flexible
  - e.g. Simple API for XML (SAX)

- Tree based (entire document as a tree at a time)
  - Advantage: entire document at any time, portions of a Document can easily be moved back and forth
  - Disadvantages: larger memory footprint, slow
  - e.g. Document Object Model (DOM) API

Contents

- XML
- XHTML
- Auxiliary technologies
- XML parser
- XML libraries in Python
  - parse XML document
  - create/modify XML document

XML parsing in Python

- `xml.etree.ElementTree`
- `lxml.etree`

```
# etree
from lxml.etree import Element

# ElementTree
from elementtree.ElementTree import Element

# ElementTree in the Python 2.5 standard library
from xml.etree.ElementTree import Element
```

Parsing library: nice-to-have

- Fast processing
- Low Memory consumption
- Complete document tree (tree-based)
- Optional partial parsing (event-based)
- XPath support

Parsing library: nice-to-haves

ElementTree

- Fast processing OK
- Low Memory consumption OK
- Complete document tree OK
- Optional partial parsing OK
- XPath support Only very limited

Parsing library: nice-to-haves

lxml.etree

- Fast processing OK
- Low Memory consumption OK
- Complete document tree OK
- Optional partial parsing OK
- XPath support OK

lxml.etree vs. ElementTree

- **lxml.etree**:
  - More functionality than ElementTree
    - e.g. (full) support for XPath, XSLT, Relax NG, XML Schema
  - Navigation to the parent of a node via `getparent()`
  - Navigation to the siblings of a node via `getnext()` and `getprevious()`

- **ElementTree:**
  - Elements have no reference to parent and sibling nodes
  - cElementTree: a fast C implementation of the ElementTree API

- **General:**
  - Same interface
  - Different behavior with respect to encodings

Source: http://lxml.de/1.3/compatibility.html

Parsing elements

XML node

<node

Node object

- n.tag <type 'dict'>
-

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
```

Node object

- `n.tag`
- `n.attrib` <type 'dict'>

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
    some text
```

Node object

- `n.tag`
- `n.attrib`
- `n.text` <type 'str'>

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
    some text
    <child /><child />
```

Node object

- `n.tag`
- `n.attrib`
- `n.text`
- `n[0], … n[len(n)-1]`

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
    some text
    <child /><child />
</node>
```

Node object

- `n.tag`
- `n.attrib`
- `n.text`
- `n[0], … n[len(n)-1]`

Parsing elements

XML node

```
<node attr1="value1" attr2="value2">
    some text
    <child /><child />
</node>some more text
```

Node object

- `n.tag`
- `n.attrib`
- `n.text`
- `n[0], … n[len(n)-1]`
- `n.tail` <type 'str'>

Methods on elements

Element attribute

Like dictionaries:

- `n.get(`*`key`*`)`: *attribute access*
- `n.set(`*`key`*`,`*`value`*`)`: *attribute set*
- `n.items()`: *attributes as a list of feature-value pairs*
- `n.keys()`: *list of attribute key*

Methods on elements

Child nodes

Like lists:

- `n.append(),n.extend(),n.insert(),n.remove()`
- `list(n)`: *list all child nodes*
- `n.find(), n.findall()`:

  search child nodes (Supports XPath paths)

Text+Berg: important elements

```
<?xml version="1.0" encoding="UTF-8"?>
<book id="1901_mul">
     <article n="55">
          <tocEntry title="Alpine Journal" author="Redaktion" lang="de" category="Kleinere
     Mitteilungen"/>
          <div>
               <s n="55-1" lang="de">
                    <w n="55-1-1" pos="ADJA" lemma="alpin">Alpine</w>
                    <w n="55-1-2" pos="NN" lemma="Journal">Journal</w>
                    <w n="55-1-3" pos="." lemma=".">.</w>
               </s>
          </div>
     </article>
</book>
```

- Book: book
- Article: article
- Sections: div
- Sentences: s
- Words: w
- Entry from Table of Contents: tocEntry

lxml.etree and ElementTree

Example 1

- ElementTree:

```
from xml.etree import cElementTree as ET
```

- lxml.etree:

```
import lxml.etree as ET
```

```
doc = ET.parse("SAC-Jahrbuch_2008_de.xml")
doc.getroot().tag
words = doc.findall("//w")
len(words)
```

All `w` elements, no matter where they occur

(c)ElementTree: `.//w`

lxml.etree and ElementTree

Example 2

Extract all French articles:

```
for article in doc.findall("//article"):
    tocEntry = article.find("tocEntry")
    if tocEntry != None and tocEntry.get("lang") == "fr":
        for sentence in article.findall(".//s"):
            words = [word.text for word in sentence.findall
            ("w") if word.text]
            print(" ".join(words))
```

Why if tocEntry != None?

lxml.etree and ElementTree

Example 2

Extract all French articles:

```
for article in doc.findall("//article"):
    tocEntry = article.find("tocEntry")
    if tocEntry != None and tocEntry.get("lang") == "fr":
        for sentence in article.findall(".//s"):
            words = [word.text for word in sentence.findall
            ("w") if word.text]
            print(" ".join(words))
```

Why if tocEntry != None?
Omission leads to errors when no `tocEntry` is found

Event-based parsing

```
for event, elem in ET.iterparse("SAC-Jahrbuch_1901_mul.xml"):
    if elem.tag == "w":
        print elem.text
    elem.clear()
```

Create/modify XML documents

Creation

- Elements can be created and modified
- Save analogous to parsing

Create/modify XML documents

Creation

- Elements can be created and modified
- Save analogous to parsing

```
# Create a new root node (Name is arbitrary)
e = ET.Element("root")
# Alternative to working with existing XML files
e = ET.parse("some_file.xml").getroot()

e.attrib["key"] = "value"
e2 = ET.Element("body")
e.append(e2)
e2.text = "hello"
ET.tostring(e) # <root key="value"><body>hello</body></root>

# Alternative write (overwrite the file specified)
ET.ElementTree(e).write("some_xml_file.xml")
```

Questions?