

Programmierkurs Python I

Stefan Thater & Michaela Regneri
Universität des Saarlandes
FR 4.7 Allgemeine Linguistik (Computerlinguistik)



Übersicht

- XML
- XML-Parser in Python
- HTML
- HTML-Parser in Python

Extensible Markup Language (XML)

- XML ist eine Sprache zum Annotieren und Strukturieren von Text
- Markierungen sind sog. *Tags* (in spitzen Klammern)
- Jeder öffnende Tag `<tag>` braucht einen schließenden `</tag>`

```
<kalender>
  <woche nr="50">
    <termin>
      <tag>Freitag</tag>
      <start>8.30</start>
      <ende>10.00</ende>
      <was>Python I</was>
    </termin>
  </woche>
</kalender>
```

3

XML - Syntax

- keine vordefinierten Elemente (Tags)
- Elementnamen: beliebige Standard-ASCII-Zeichen, keine Leerzeichen; am Anfang muss ein Buchstabe stehen, aber nicht xml / Xml / XML ...
- Die Elemente dürfen Attribute haben:
`<element attribut="wert">...</element>`
- zwischen öffnendem und schließendem Tag kann Text stehen (**CDATA**): `<element>c-data text</element>`
- Tags dürfen „leer“ sein, (keine CDATA) in dem Fall darf man Start- und End-Tag zusammenfassen:
`<element attribut="irgendwas" />`

4

XML - Syntax

- Elemente dürfen (und müssen) geschachtelt sein
 - es gibt immer genau ein Wurzel-Element (*Root*), das alle anderen Element enthält
 - ein Element darf beliebig viele *Kinder* (=eingeschachtelte Elemente) haben
- Die Elemente müssen *wohlgeschachtelt* sein:
wenn ein Tag <tag1> vor einem Tag <tag2> geöffnet wird, muss der schließende Tag </tag2> vor </tag1> kommen
(verboten: <wurzel><a></wurzel>)

5

XML - Kopfzeile

- XML-Dateien sollten eine Kopfzeile haben, die die benutzte XML-Variante decodiert; minimal so:
<?xml version="1.0"?>
- In dieser Zeile kann auch Encoding spezifiziert werden, damit die übrige Datei später korrekt verarbeitet werden kann:
<?xml version="1.0" encoding="ISO-8859-1"?>

6

XML - ein Beispiel

```
<?xml version="1.0"?>
<sentence>
  <phrase type="NP">
    <word id="1" pos="Det" flek="nom.pl.m">Die</word>
    <word id="2" pos="NN" lemma="Spatz" flek="nom.pl.m">
      Spatzen</word>
  </phrase>
  <phrase type="VP">
    <word id="3" pos="VV" lemma="frieren"
      flek="3ps.pl.indik.praes.akt">frieren</word>
  </phrase>
</sentence>
```

7

Document Type Definition (DTD)

<http://www.w3schools.com/dtd/default.asp>

- DTDs beschreiben die Struktur für bestimmte XML-Dokumente (Elemente, Attribute und deren Anordnung)
- DTDs können sehr nützlich sein, wenn man mehrere XML-Dokumente der gleichen Klasse hat (z.B. mehrere Sätze, annotiert nach dem gleichen Schema)
- DTDs können als eigene Datei oder in der XML-Datei definiert werden
- Ein einfaches Beispiel für die Satz-Annotation...

8

DTD - ein einfaches Beispiel

```
<!DOCTYPE sentence [  
<!ELEMENT sentence (phrase+)>  
<!ELEMENT phrase (word+)>  
<!ELEMENT word (#PCDATA)>  
  
<!ATTLIST phrase type CDATA #REQUIRED>  
<!ATTLIST word id ID #REQUIRED>  
<!ATTLIST word pos CDATA #REQUIRED>  
<!ATTLIST word lemma CDATA #IMPLIED>  
<!ATTLIST word flek CDATA #IMPLIED>  
>
```

Wurzel

Elemente:
Name, Kinder
(Typ/Tag, Menge)

Attribute:
zugeh. Element,
Name, Typ,
Restriktion

9

DTD und XML-Dateien

- ein Sinn von DTDs kann sein, XML-Dateien zu validieren
- Um eine XML-Datei mit einer DTD zu verknüpfen, fügt man zwischen Kopfzeile und XML-Code
 - entweder die DTD selbst ein
 - oder folgendes:
<!DOCTYPE note SYSTEM "mydtd.dtd">
(mydtd.dtd enthält die DTD, nach der die XML-Datei validiert werden soll)

10

XML-Parser in Python (und anderen Sprachen)

- SAX (Simple API for XML)
 - SAX-Parser verarbeiten XML-Dokumente von vorne nach hinten
 - bestimmte Methoden werden aufgerufen, wenn Tags geöffnet oder geschlossen werden
 - kein Zugriff auf Dokumentstruktur (Kinder-Elemente, nachfolgende Elemente auf der gleichen Ebene / mit dem gleichen Namen..)
- DOM (Document Object Model)
 - Liest das Dokument einmal ganz ein
 - stellt dann die komplette Struktur zur Verfügung

11

SAX-Parser in Python

<http://docs.python.org/3.1/library/xml.sax.html>

- `xml.sax.make_parser()` gibt ein generisches Parser-Objekt zurück
- Das Parser-Objekt ist eine Instanz von `XMLReader`, es kann u.a. die XML-Datei lesen und strukturiert ausgeben (Zugriff auf Attribute, Tags...)
- damit der Parser etwas mit der Datei tut, muss mit `parser.setContentHandler(ContentHandler)` ein „Content Handler“ eingebaut werden
- `ContentHandler` ist die Klasse, von der wir unsere eigene XML-verarbeitende Klasse ableiten müssen

12

SAX-Parser - ContentHandler

<http://docs.python.org/3.1/library/xml.sax.handler.html>

- `startDocument()` / `endDocument()` wird direkt vor / nach Verarbeiten des XML-Codes aufgerufen
- `startElement(name, attrs)` wird bei jedem öffnenden Tag aufgerufen
 - `name` ist die Bezeichnung des Tags
 - `attrs` ist ein `Attributes`-Objekt:
 - * Anzahl der Attribut-Wert-Paare: `attrs.getLength()`
 - * Liste alle Attributs-Namen: `attrs.getNames()`
 - * der Wert von Attribut `at`: `attrs.getValue(at)`
 - * alle Einträge in `attrs` (\approx Dictionaries): `atts.items()`

13

SAX-Parser - ContentHandler

<http://docs.python.org/3.1/library/xml.sax.handler.html>

- `endElement(name)` wird aufgerufen, wenn ein Tag geschlossen wird
- `characters(str)` gibt aktuell gelesene CDATA zurück
 - keine Garantie, dass die CDATA für den aktuellen Tag als ganzes zurückgegeben wird!
 - um den ganzen Text zu bekommen, muss man die CDATA für den aktuellen Tag „aufsammeln“ -> ein Beispiel...

14

SAX-Parser - Beispiel

```
from xml.sax.handler import ContentHandler

class CDATAPrinter(ContentHandler):
    def startElement(self, name, attrs):
        self.cdata= ""

    def endElement(self, name):
        if self.cdata.strip():
            print(name, ':', self.cdata.strip())

    def characters(self, str):
        self.cdata += str
```

15

SAX-Parser - Beispiel

```
<kalender> <woche nr="50"> <termin>                                cal.xml
  <tag>Freitag</tag> <start>8.30</start>
  <ende>10.00</ende> <was>Python I</was>
</termin></woche></kalender>
```

```
import xml.sax
parser = xml.sax.make_parser()
parser.setContentHandler(CDATAPrinter())
parser.parse('cal.xml')
```

```
tag : Freitag                                Konsolenausgabe
start : 8.30
ende : 10.00
was : Python I
```

16

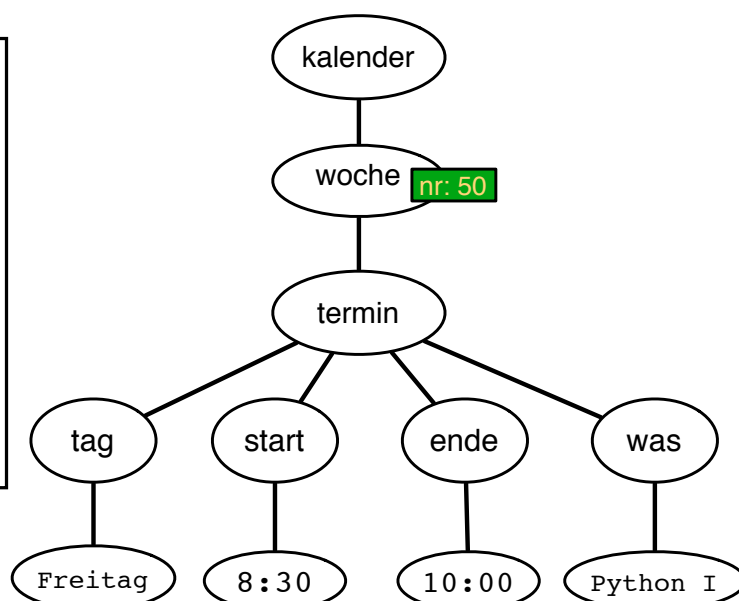
DOM

- DOM ist kein reiner Parser, sondern eine Datenstruktur für Zugriff *und* Bearbeitung / Erstellung von XML-Strukturen
- anderes Konzept als SAX: das Dokument wird eingelesen, dann kann strukturierter Zugriff erfolgen
- die benutzerdefinierten Aktionen finden mit dem DOM-Objekt statt, nicht beim Einlesen der XML-Datei
- das ist speicherintensiver, aber kann auch deutlich bequemer sein
- Ein DOM-Parser verwaltet Dokumente als Bäume:

17

Baumstruktur in XML

```
<kalender>  
  <woche nr="50">  
    <termin>  
      <tag>Freitag</tag>  
      <start>8.30</start>  
      <ende>10.00</ende>  
      <was>Python I</was>  
    </termin>  
  </woche>  
</kalender>
```



18

DOM-Parser in Python

<http://docs.python.org/3.1/library/xml.dom.html>

<http://docs.python.org/3.1/library/xml.dom.minidom.html>

- Zuerst erzeugt man sich ein DOM-Objekt:

```
from xml.dom.minidom import parse
domobject = parse('calendar.xml')
```

- Man kann auch XML-String direkt parsen:

```
from xml.dom.minidom import parseString
domobject = parseString("<root><c>text</c></root>")
```

- auf die Knoten im XML-Baum kann dann mit verschiedenen Methoden zugegriffen werden

19

DOM-Parser in Python

Alle Elemente sind von Node abgeleitet und geben Zugriff auf diverse Felder und Methoden:

- `Node.nodeType` Die Art des XML-Elements, u.a.: `TEXT_NODE`, `ENTITY_NODE`, `DOCUMENT_NODE`, `ATTRIBUTE_NODE`
- `Node.childNodes` Geordnete Liste der direkten Kinder
- `Node.firstChild` / `Node.lastChild`
- `Node.hasChildNodes()`
- `Node.attributes` das Attribut-Objekt
- `Node.nextSibling` / `Node.previousSibling`

20

DOM-Parser in Python

- für „normale“ Tags (Element):
 - `Element.tagName`
 - `Element.hasAttribute(name)`
 - `Element.getAttribute(name)`
 - `Element.getElementsByTagName(name)`
- Attribut-Objekte funktionieren im Prinzip wie in SAX (sind aber abgeleitet von `Node`)
- CDATA sind `Text`-Objekte, auch von `Node` abgeleitet. Für Knoten der Klasse `Text` gibt es das Feld `Text.data` (beinhaltet den CDATA-Text)

21

DOM-Parser - Beispiel

```
from xml.dom.minidom import parse

domobject = parse('sentence.xml')
for e in domobject.getElementsByTagName('word'):
    for a,v in e.attributes.items():
        print(a, ':',v)
    for node in e.childNodes:
        if node.nodeType == node.TEXT_NODE:
            print(node.data)
```

22

Hyper Text Markup Language (HTML)

<http://de.selfhtml.org/>

<http://www.w3schools.com/html/default.asp>

- HTML dient zur Formatierung und „Funktionalisierung“ von Text
- anders als bei XML geht es weniger um zusätzliche Semantik als um die Visualisierung und Formatierung der Strings (und anderer Sachen)
- Wie bei XML: Tags in spitzen Klammern
- Anders als bei XML: mögliche Tags sind vordefiniert (und werden von Browsern etc. visualisiert)

23

HTML - Beispiel

```
<html>
  <body>
    <b>This</b> <i>text</i> is <code>formatted</code>.
  </body>
</html>
```

This *text* is formatted.

24

HTML - XML

- auch in HTML haben manche Tags Attribute, wie der Tag für Links: `YAHOO!`
- nicht alle HTML-Seiten sind auch korrektes XML:
 - nicht alle Tags in HTML müssen geschlossen werden (z.B. `<p>` für Abschnitte, `` für Elemente in Aufzählungen)
 - manche Attribute in HTML dürfen ohne Anführungszeichen benutzt werden, z.B. `<table border=0>`
 - HTML unterscheidet Groß- und Kleinschreibung nicht (`fett` ist ok in HTML)

25

HTML-Parser in Python

<http://docs.python.org/3.1/library/html.parser.html>

- HTML-Parsing funktioniert ähnlich wie XML-Parsing
- Um eigene Parser-Funktionen zu implementieren, leitet man eine Klasse von `a` ab
- Die zur Verfügung stehenden Methoden sehen ähnlich aus wie die im XML-Parser:
 - `handle_starttag(tag, attrs)`, `handle_endtag(tag)` und `handle_startendtag(tag, attrs)` (leere Elemente)
 - `handle_data(data)` ist (so ungefähr) äquivalent zu `characters(str)`

26

HTML-Parser in Python

<http://docs.python.org/3.1/library/html.parser.html>

- der HTML-Parser ist sein eigener „ContentHandler“, eine Instanz gibt's einfach mit `HTMLParser()` bzw. einer Instanz der abgeleiteten Klasse
- um zu Parsen, „füttert“ man den Parser mit HTML-Text:
`parser.feed(htmlstring)`
- wenn der Parser inkorrektem XML begegnet, wird er es so lange ignorieren, bis der Fehler unkorrigierbar oder das Dokument zu Ende ist
- wenn man den Text eine URL parsen will, muss man die URL zuerst einlesen:

```
text = urllib.request.urlopen(url).read()
parser.feed(str(text))
```

27

HTML-Parser - Beispiel

```
from html.parser import HTMLParser

class MyHTMLParser(HTMLParser):

    def handle_starttag(self, tag, attrs):
        print("Oeffne " , tag)

    def handle_endtag(self, tag):
        print("Schliesse", tag)

    def handle_startendtag(self, tag, attrs):
        print("Leerer Tag", tag)
```

28

HTML-Parser - Beispiel

```
from urllib.request import urlopen
[...]  
parser = MyHTMLParser()  
parser.feed(str(urlopen("http://www.flickr.com/").read()))
```

Konsole

```
Oeffne html  
Oeffne head  
Oeffne meta  
Oeffne title  
Schliesse title  
Oeffne meta  
[...]  
Leerer Tag meta  
Schliesse head
```

```
<html> flickr.com  
<head>  
  <meta http-equiv="[...]>  
  <title>Welcome to Flickr [...]</title>  
  <meta name="keywords" content="[...]">  
  [...]  
  <meta name="viewport" content="width=950"/>  
</head>
```

29

Zusammenfassung

- XML zur Annotation von Text
- DTDs zum Validieren von XML-Dokumenten
- XML-Parsing
 - SAX-Parser
 - DOM
- HTML zur Formatierung von Text
- HTML vs. XML
- HTML-Parsing

30