Dienstag: Markup/Datenformate

4 Markup/Datenformate
- XML
- JSON

Markup/Datenformate

Motivation

- Viele Korpora enthalten Informationen über den reinen Text hinaus:
  - Metadaten
  - Annotationen
  - Dokumentgrenzen
- Annotationen oder Metadaten sind Eigenschaften von Texten
- Standard für Darstellung der Eigenschaften von Daten: XML
- Vorteil: Standard genau definiert, Bibliotheken für viele Programmiersprachen vorhanden

Markup/Datenformate

eXtensible Markup Language I

- XML ermöglicht die Speicherung und den Austausch strukturierter Daten über Textdateien
- Schreiben und Lesen mit normalem Text-Editor (emacs, vi, kate)
- XML-Parser erkennen die Struktur von XML-formatierten Daten
- Heutzutage sind viele Daten XML-formatiert (z.B. ODF-Formate)
- Ausführliches Tutorial unter `http://w3schools.com/xml/`

eXtensible Markup Language II

- XML-Dateien beginnen mit einem Header
- Daten sind zwischen XML-Tags eingeschlossen
- XML-Tags sind grundsätzlich nicht vordefiniert

Beispiel (`starwars.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
   <person>
      <name>Luke Skywalker</name>
      <homeworld>Tatooine</homeworld>
   </person>
   <person>
      <name>Han Solo</name>
      <homeworld>Corellia</homeworld>
   </person>
</people>
```

eXtensible Markup Language III

Beispiele (Attribute, Leere Elemente)

```
<person>
   <name>Han Solo</name>
   <homeworld>Corellia</homeworld>
   <height unit="meters">1.8</height>
</person>
```

```
<person>
   <name>Luke Skywalker</name>
   <jedi />
</person>
```

Zeichenvorrat I

An sich können sowohl in den Attributen als auch zwischen Tags beliebige Zeichen aus UTF-8 verwendet werden. Aber:

Entities

- & `&`
- < `<` (less than)
- > `>` (greater than)
- ' `&apos;` (apostrophe, single-quote)
- " `&quot;` (quotation mark, double-quote)

Beispiel

```
<text>7 &gt; 5</text>
```

Zeichenvorrat II

CDATA-Deklaration

Innerhalb von XML-Tags kann die CDATA-Deklaration verwendet werden, um anzuzeigen, dass Character-Daten verwendet werden. Dann müssen keine XML-Entities benutzt werden

Beispiel

```
<text><![CDATA[7 > 5]]></text>
```

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes:
  `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes: `http://www.unicode.org/charts/`

Numeric Character Reference (NCR)

- Unicode-Zeichen sind nummeriert
- Mit einer NCR kann man eine Nummer angeben, die als Zeichen interpretiert wird
- Hexadezimal `&#xHHH;` HHH ist der Hexadezimalwert des Zeichens

  Dezimal `&#DDD;` DDD ist der Dezimalwert des Zeichens

Beispiel

Linefeed `
`

Ē `Ē`

ů `ů`

- Weitere numerische Codes:
  `http://www.unicode.org/charts/`

Leere Tags

Beispiele (Verschiedene Arten, leere Elemente zu notieren)

```
<jedi></jedi>
<jedi />
<jedi/>
```

Beispiel (Auch leere Tags können Attribute enthalten)

```
<jedi dark="no" />
```

Kommentare

Beispiel

```
<person>
   <name>Han Solo</name>
   <homeworld>Corellia</homeworld>
   <!-- dies ist ein kommentar -->
</person>
```

Markup/Datenformate

XML-Parser: DOM

- DOM steht für „Document Object Model”
- Liest das gesamte XML-Dokument in den Speicher und baut eine Baumstruktur im Speicher auf
- Objekte im DOM-Baum können über Methoden angesprochen werden (z.B. `childOf()`, `attributeOf()`)
- Vorteil: einfach und übersichtlich
- Nachteil: hoher Speicherverbrauch
- Weitere Informationen unter `http://www.w3.org/DOM/`

XML-Parser: SAX

- SAX steht für „Simple API for XML”
- Ereignis-basierte API: Man definiert aufzurufende Funktion, wenn bestimmte Elemente gefunden werden
- Nachteile: man muss die XML-Struktur gut kennen, keine automatischen Kontext-Informationen, Validierung sehr kompliziert
- Vorteile: sehr speichereffizient, sehr schnell
- Weitere Informationen unter `http://www.saxproject.org`

XML-Parsing in Python

Dokumentation

`https://docs.python.org/library/xml.html`

DOM in Python

- `xml.dom`
- `xml.etree.ElementTree`

XML-Parsing in Python

Dokumentation

`https://docs.python.org/library/xml.html`

DOM in Python

- `xml.dom`
- `xml.etree.ElementTree`

ElementTree Datenstrukturen

ElementTree überträgt das komplette XML-Dokument in eine Baumstruktur.

Element repräsentiert einen Knoten im XML-Baum mit folgenden Eigenschaften

- Tag als string
- Attribute als dictionary
- Text als string
- Alle Kindknoten als Sequenz-Objekt

XML-Parsing in Python

Beispiel (ElementTree)

```
>>> import pprint, xml.etree.ElementTree as et
>>> tree = et.parse('starwars.xml')
>>> r = tree.getroot()
>>> r.tag
'people'
>>> r.attrib
{}
>>> people = list(r) # Kindknoten von r
>>> person1 = people[0]
>>> pprint.pprint(list(person1)) # Kindknoten von person1
[<Element 'name' at 0x7fd824cc5150>,
 <Element 'homeworld' at 0x7fd824cc5210>,
 <Element 'jedi' at 0x7fd824cc52d0>]
 >>> people[0].find("name").text # find() Funktion
'Luke Skywalker'
```

XML-Parsing in Python

SAX

- Modul `xml.sax`
- in DOM wird ein Baumobjekt aus dem Dokument konstruiert.
- in SAX wird ein Parser-Objekt konstruiert, das das Dokument sequentiell nach bestimmten *Inhalten* durchsucht.
- Was mit den Inhalten gemacht werden soll, definiert ein `ContentHandler`, der dem Parser übergeben wird.
- SAX-Modul enthält einen Standard-ContentHandler, von dem ein eigener ContentHandler erbt.

XML-Parsing in Python

Beispiel (benutzerdefinierter ContentHandler)

```
import xml.sax

class JediSpotter(xml.sax.ContentHandler):
    def __init__(self):
        xml.sax.ContentHandler.__init__(self)

    def startElement(self,name,attrs):
        """überschreibt Methode von ContentHandler"""
        if name  == "jedi":
            print "I saw a jedi!"
```

XML-Parsing in Python

Beispiel (SAX Parsing)

```
>>> handler = JediSpotter()
>>> xml.sax.parse(open('starwars.xml'),handler)
I saw a Jedi!
```

XML und HTML

Historisch gesehen kommt XML nach HTML, ist aber allgemeiner und hat eine andere Zielsetzung:

- HTML beschreibt, wie Daten *aussehen* sollen
- XML beschreibt, welche Informationen Daten beinhalten und welche Eigenschaften sie haben

HTML - Beispiel

Beispiel

```
<b>
   Fetter Text.
</b>
Das ist ein
<a href="http://www.google.de">
   Link
</a>
```

Ausführliche Informationen zu HTML: `http://de.selfhtml.org`

Markup/Datenformate

XPath I

- XML-Strukturen sind baumartig
- XPath beschreibt Pfade durch diesen XML-Baum
- Die XPath-Notation ist Pfaden im Unix-Dateisystem sehr ähnlich
- Für XPath ist jede Entität in XML ein Knoten:
  - Elemente
  - Attribute
  - Text
  - Kommentare
  - ...
- Weitere Informationen unter `http://www.w3schools.com/xml/xpath_intro.asp`

XPath II

Beispiel

```
<people>
   <person>
      <name>Luke Skywalker</name>
   </person>
   <person>
      <name>Han Solo</name>
   </person>
</people>
```

`/people`	people
`/people/person`	person (Wählt mehr als ein Element aus!)
`/people/person[1]`	person (wählt das erste person unter people)

XPath III

Syntax

knotenname	Wählt alle (direkten) Kindknoten mit Tag *knotenname*
/	Pfad beginnt am Root-Knoten
//	Wählt Knoten im Dokument unabhängig davon, wie tief sie eingebettet sind.
.	Wählt den aktuellen Knoten
..	Wählt den Eltern-Knoten des aktuellen Knotens
@	Wählt Attribute

XPath IV

Beispiele

`people`	Wählt alle direkten Kindknoten mit Tag `people`
`/people`	Wählt das root-Element people
`//person`	Wählt alle person-Elemente, egal wo sie im Dokument stehen
`//people/person`	Wählt alle person-Elemente, die Kind von einem people-Element sind
`//@unit`	Wählt alle Attribute, die unit heißen
`/people/person[last()]`	Wählt das letzte person-Element unter people
`//height[@unit='meters']`	Wählt alle height-Elemente, die ein Attribut unit haben, das auf „meters" gesetzt ist

Beispiele Python

- `ElementTree`: teilweise Unterstützung von XPath Syntax
- Suchpfad muss immer relativ zum aktuellen Knoten angegeben werden (siehe Beispiel)!

```
>>> import xml.etree.ElementTree as et
>>> tree = et.parse('starwars.xml')
>>> r = tree.getroot()
>>> # alle name-Elemente
>>> r.findall(".//name")
```

https://docs.python.org/2/library/xml.etree.elementtree.html#xpath-support

Markup/Datenformate

JSON

- JSON = `JavaScript Object Notation`
- Textbasiertes Format zum Datenaustausch.
- Entwickelt in den frühen 2000ern für Kommunikation zwischen Server und Webanwendung.
- Verwendet JavaScript(-ähnliche) Syntax.
- In vielen Programmiersprachen sind Bibliotheken für Dekodierung und Generierung von JSON vorhanden

`http://www.w3schools.com/js/js_json_intro.asp`

Vergleich mit XML

Gemeinsamkeiten

- Textbasiert, für Menschen (und Maschinen) lesbar
- Hierarchisch
- Sprachunabhängig

Unterschiede

- XML ist eine Auszeichnungssprache
- JSON ist ein Datenformat
- JSON kann Arrays verwenden

Beispiel

Beispiel (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<people>
   <person>
      <name>Luke Skywalker</name>
      <homeworld>Tatooine</homeworld>
   </person>
   <person>
      <name>Han Solo</name>
      <homeworld>Corellia</homeworld>
   </person>
</people>
```

Syntax I

Schlüssel-Wert-Paare

- Daten werden als Schlüssel-Wert-Paare dargestellt (`"name":"Han Solo"`)
- Schlüssel ist immer eine Zeichenkette
- Mögliche Werte:
  - null
  - Zahl (`int`,`float`)
  - Zeichenkette
  - Wahrheitswert
  - Array (in `[]`)
  - Objekt

Syntax II

JSON-Objekte

- Stehen in `{ }`
- Enthalten durch Kommata getrennte Schlüssel-Wert-Paare
- In einem Objekt darf jeder Schlüssel nur einmal verwendet werden!

Beispiel (JSON)

```
{"people":[
    {"name":"Luke Skywalker", "homeworld":"Tatooine"},
    {"name":"Han Solo", "homeworld":"Corellia"}
]}
```

Warum JSON?

- Kürzer als XML (keine End-Tags), daher weniger Speicherverbrauch
- Oft weniger Overhead beim Einlesen von JSON-Daten
- Häufig verwendet: z.B. Twitter API, Serialisierung von Datenstrukturen
- Nützlich für weniger komplexe Daten (z.B. Metadaten)

JSON verwenden

Beispiel (JSON in Python)

```
>>> import json, pprint
>>> obj = json.load(open('starwars.json'))
>>> pprint.pprint(obj)
{u'people': [{u'homeworld': u'Tatooine',
              u'name': u'Luke Skywalker'},
             {u'homeworld': u'Corellia',
              u'name': u'Han Solo'}]}
>>> json.dump(obj,open('starwars_dump.json','w'),indent=3)
```

https://docs.python.org/2/library/json.html

Übung 5