CSV, XML, JSON, REST and SOAP

Kai Dührkop

Lehrstuhl fuer Bioinformatik
Friedrich-Schiller-Universitaet Jena
kai.duehrkop@uni-jena.de

11.-15. August 2014

Section 1

Data formats

- binary
  - not human readable
  - memory efficient
  - fast to parse
  - but: platform dependend (Little Endian vs. Big Endian?)
  - difficult to evolve the format (e.g. Word Documents are incompatible between different versions)

- binary
  - not human readable
  - memory efficient
  - fast to parse
  - but: platform dependend (Little Endian vs. Big Endian?)
  - difficult to evolve the format (e.g. Word Documents are incompatible between different versions)
- text
  - human readable (okay, xml...)
  - waste more memory? (compressed text files are often smaller than binary files!)
  - slow to parse
  - platform independend (but: still encoding problems!)
  - format can evolve (e.g. additional fields in xml)

- CSV: simplest format possible: Just strings separated by commas and newlines
- XML: most common text data format. XHTML is the language of the web.
- JSON: uses javascript syntax. Much better readable and more sparse than XML
- YAML: Same as JSON, but uses indentation instead of brackets

```
name,formula,id
glycose,C6H12O6,79025
alanine,C3H7NO2,5950
guanine,C5H5N5O,764
"4-amino-1,2,4-triazin-5-one",C10H10N4O,58
```

- column separator: default is ",", but also "\t" very common
- row separator: usually "\n"
- strings can be enclosed in quotation marks to escape special characters
- quotation marks are escaped by another quotation marks
- csv can be read and written in excel!

poor man's csv

```
with open("table.csv", "rb") as csvf:
  table = [line.split(",") for line in csvf]
```

very easy for simple formatted files (without escapes and quotation marks)

read csv

```
import csv
with open("table.csv", "rb") as csvf:
  table = [row for row in csv.reader(csvf)]
```

write csv

```
import csv
table = [["name", "formula", "id"],
  ["glucose", "C6H12O6", 79025]]
with open("table.csv", "wb") as csvf:
  csv.writer(csvf).writerows(table)
```

changing delimiters

```
import csv
# row, col and string sep can be changed
csv.reader(csvfile, delimiter="\t",
  quotechar="'")

# predefined 'dialects'
csv.reader(csvfile, dialect="excel")
```

- whenever your data has no nested structure: USE CSV!
- can be easily read in every programing language
- can be opened in text editors and in excel
- comma is default, but tabs are often better as you rarely have to escape strings in tab separated files

```
[
  {
    "name": "Glucose", "formula": "C6H12O6",
    "id": 79025,
    "similarTo": ["Hexose", "Fructose"],
    biological: true
  }
]
```

- just javascript data. Also almost compatible to pythons syntax
- booleans, numbers, strings, arrays, objects (dictionaries)
- popular in web: client and webserver often use JSON to communicate with each other. Javascript can naturally work with json.
- usually human readable (if indented) but not easy to parse

read json

```
import json
with open("someFile.json", "r") as jsonFile
  compounds = json.load(jsonFile)
compounds[0]["name"] #=> Glucose
compounds[0]["biological"] #=> True
```

write json

```
import json
mycompounds = [{"name": "Fructose",
  "formula": "C6H12O6", "id": 79025,
  "similarTo": ["Hexose", "Glucose"]}]
with open("someFile.json", "w") as jsonFile
  json.dump(jsonFile, mycompounds)
```

- text files always have an **encoding**
- but: encoding is neither a file attribute nor written in the text file
- programs have to guess the encoding (very bad!)
- unicode (UTF-8) as international standard for almost all characters
- linux uses UTF-8 by default, windows uses a lot of different encodings (UTF-16, latin1, ...)
- UTF-8 is superset of ASCII! So all ASCII files can be savely encoded as UTF-8

read json with encoding

```
import json
with open("someFile.json", "w") as jsonFile
  json.load(jsonFile, encoding="utf-16")
```

```
<?xml version="1.0" encoding="UTF-8"?>
<someRoot>
  <someElem>
  <otherElem someAttr="value">Some text</otherElem>
  </someElem>
</someRoot>
```

- most common data format
- solves a lot of problems (namespaces, encoding, embedded data)
- but not very readable, very verbose

- XML is a tree
- node types: root, elements, text, attribute, comments, ...
- **schema** defines structure of a xml document
  - dtd (document type definition)
  - xsd (type definition in xml)
- XML parser (python builtin **xml** or external package **lxml**)
  - SAX (streaming parser)
  - DOM (tree like structure)
  - XPATH (querying in xml)

DTD

```
<!ELEMENT people_list (person)*>
<!ELEMENT person (name, birthdate?, gender?,
          socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

- ELEMENT defines what can be contained in an element node
- * for many, ? for optional (like regexp)
- #PCDATA for arbitrary text

XSD

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified" xmlns:xs="http://
  <xs:element name="people_list">
    <xs:complexType>
       <xs:sequence>
        <xs:element name="person">
          <xs:complexTye>
          ...
          </xs:complexTye>
        </xs:element
       </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
</xs:schema>
```

- xml documents should define their schema
- schema is often available online

SAX

- define a handler with callbacks that are called for certain events
- events are e.g. opening a node, closing a node, reading some text, ...
- very efficient, can parse arbitrary huge files
- handler is usually a finite state machine

SAX Handler

```
import xml.sax as sax
class MySaxHandler(sax.ContentHandler):
  def __init__(self):
    self.listening = False
    self.numlist = []

  def startElem(self, name, attrs):
    if name == "interesting":
      self.listening = True

  def endElem(self, name):
    self.listening = false
```

SAX Handler

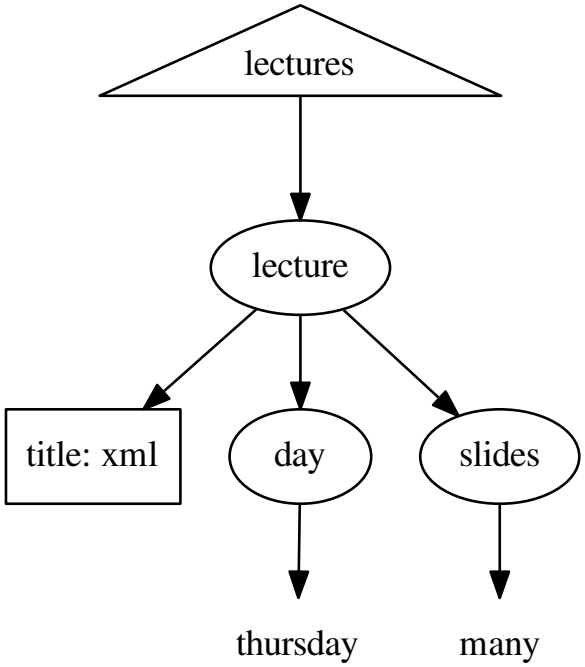
```
class MySaxHandler(sax.ContentHandler):
  def characters(self, chrs):
    if self.listening:
      self.numList.append(
        [int(x) for x in chrs.split]
      )
```

SAX Handler

```
handler = MySaxHandler()
sax.parse("myfile.xml", handler)
handler.numlist #=> [...]
```

DOM

```
<lectures>
  <lecture title="xml">
    <day>thursday</day>
    <slides>many</slides>
  </lecture>
</lectures>
```



DOM

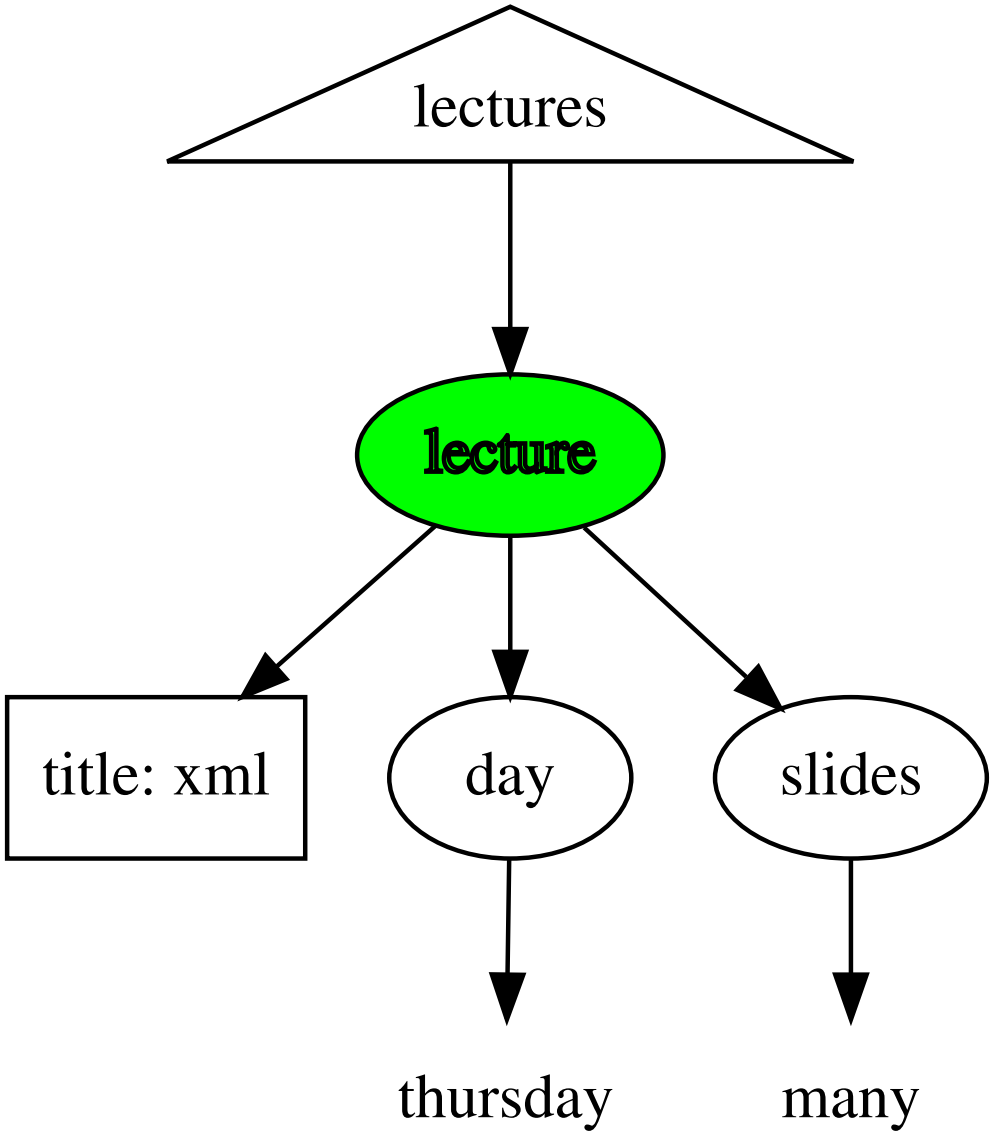
```
from xml.dom.minidom import parse
import xml.dom as dom
document = parse("myFile.xml")
# get tag name of root node
root = document.documentElement
print root.tagName
# get tag names of all children
[node.name for node in root.childNodes]
# get titles of all lecture nodes
[node.getAttribute("title") for node
  in document.getElementsByTagName("lecture")
]
```

ETree

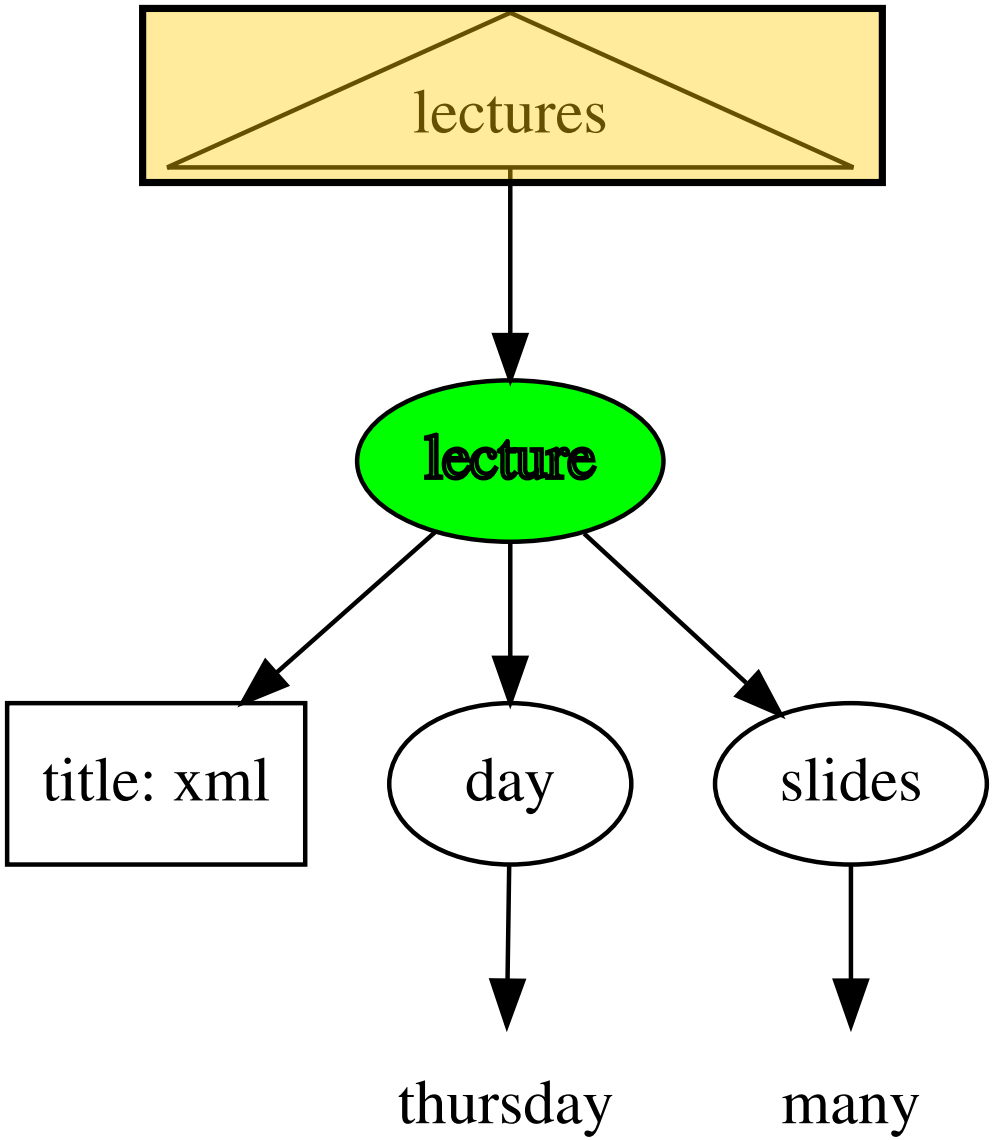
```
import xml.etree.ElementTree as ET
root = ET.parse("myFile.xml")
root.tag #=> lectures
root[0] #=> first child
root[0].attrib["title"] #=> "xml"
# all text in children
[node.text for node in root]
# search lecture nodes in subtree
[node.attrib["title"] for node
  in root.findIter("lecture")]
```

XPath

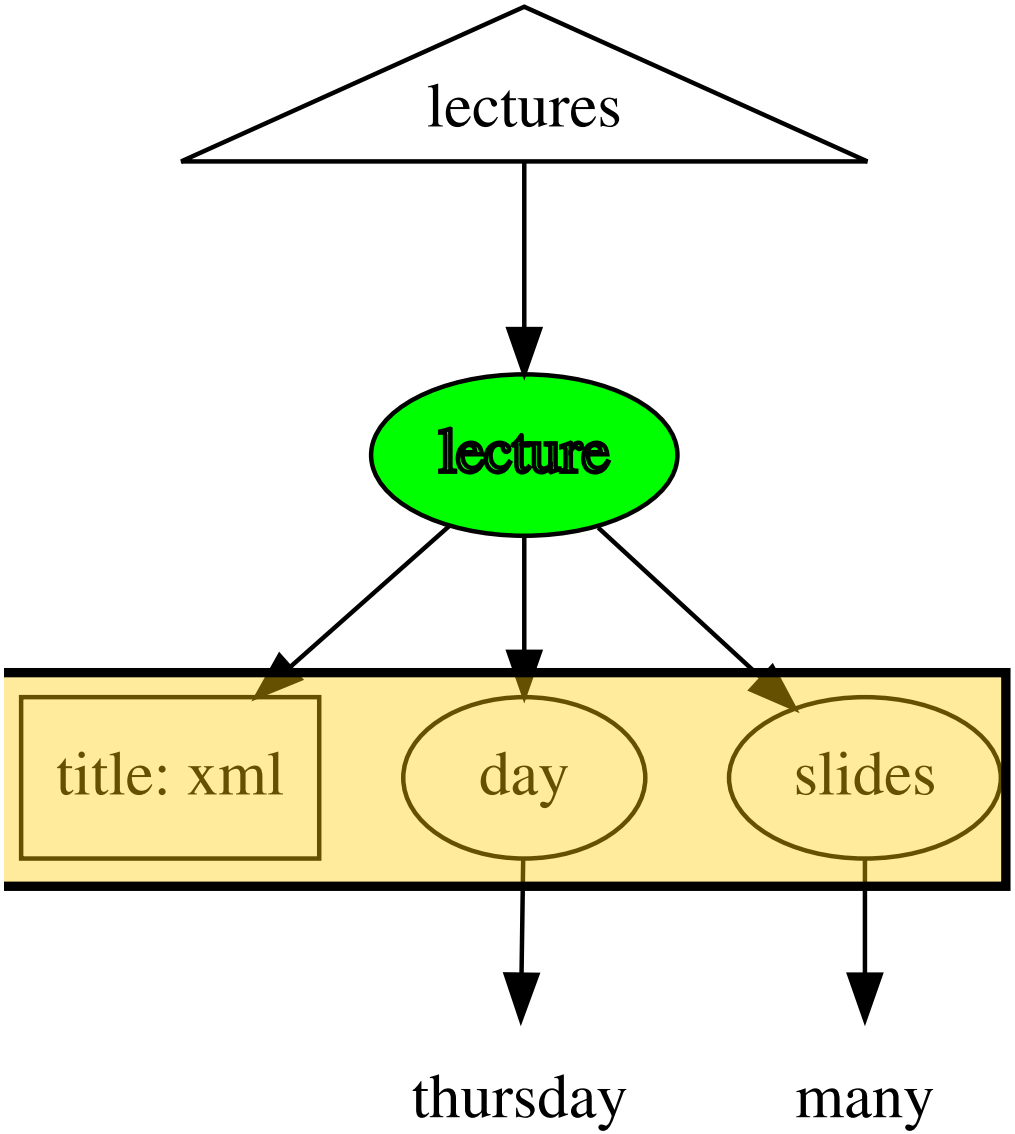
- query language for searching in XML
- search **node sets**
- an XPath expression consists of an **axis**, a **node test** and a list of **predicates**
- the expression describes a **path** or **subtree** in the XML document



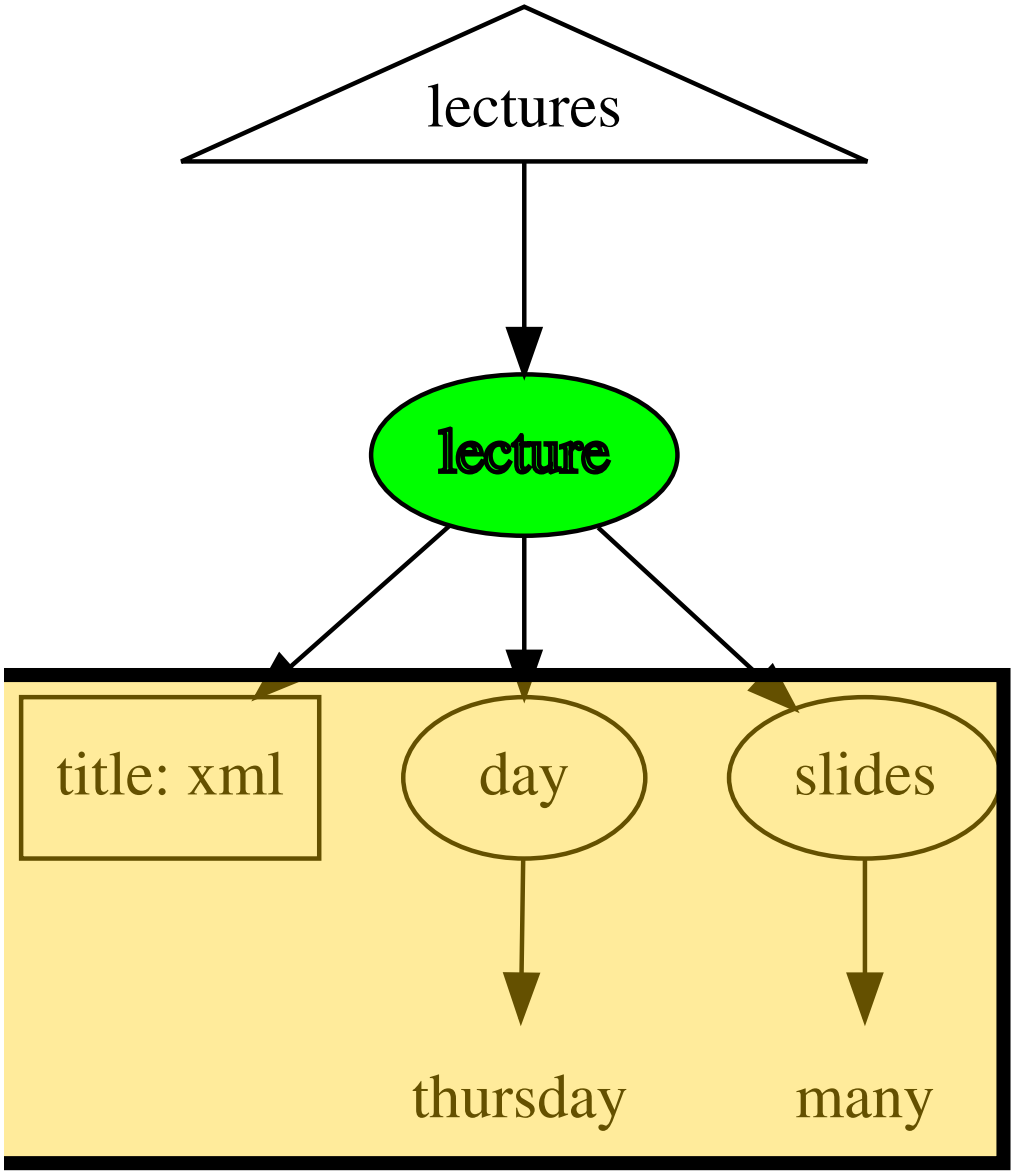
- axis is *direction* from start node



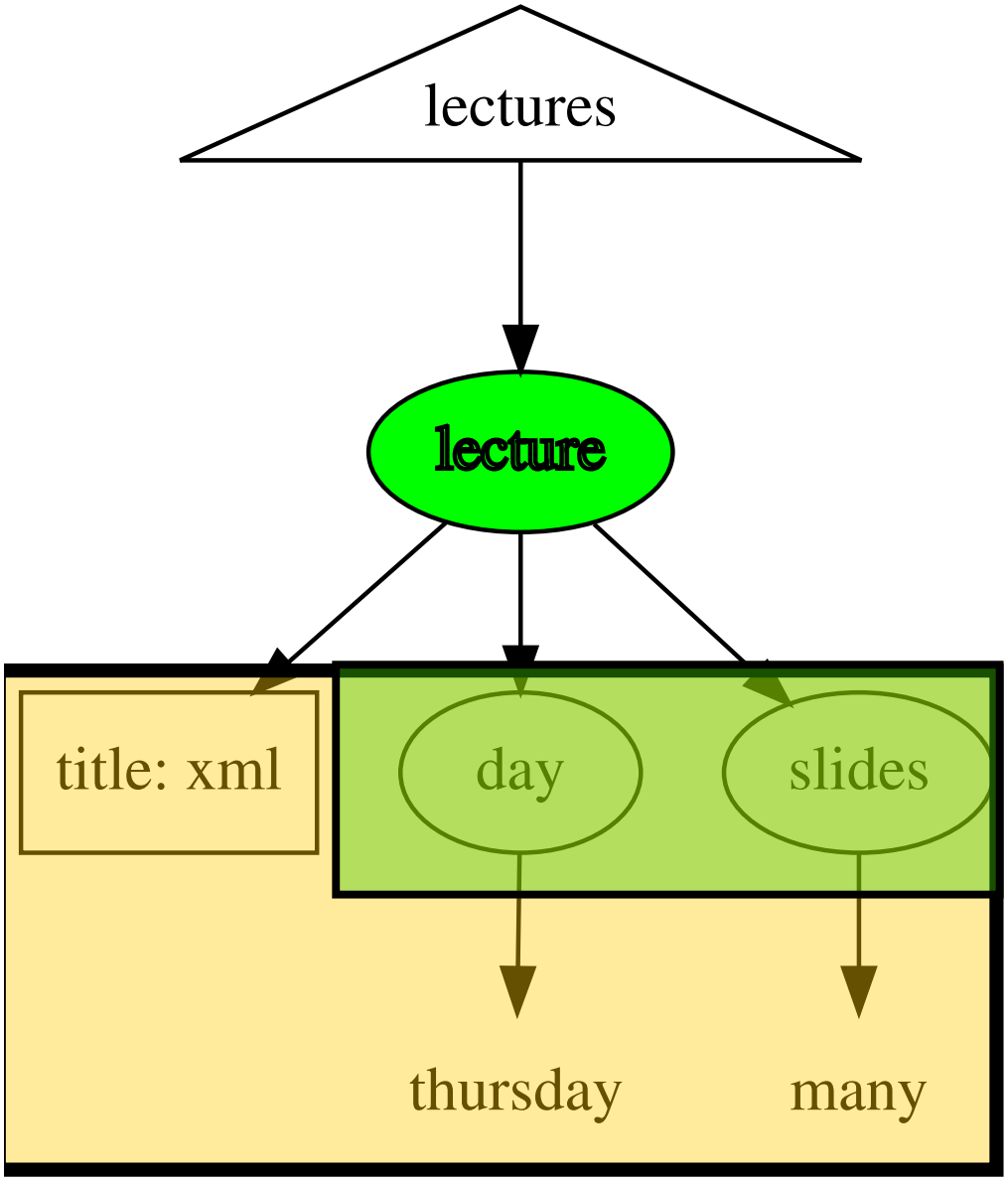
- **parent**



- **child**



- **descendant**



- **node test restrict to certain node types (e.g. element nodes or text nodes)**

```
child::A/descendant::node()/child::text()
```

- verbose syntax
- each single expression is a selection step
- expressions combined with / are applied on all nodes in the resulting node set

```
../A/B//C/*/E
```

- abbreviations for common axis
- . is current node, .. is parent, // is descendant, / is child
- wildcard * for tag name
- similar to linux file paths

```
//slides[@title="xml"][1]/child::text()
```

- predicates are written in square brackets
- most predicates restrict the node set to nodes with special attributes or attribute values
- other xpath operations like union are also possible
- but usually it's easier to just search nodes with XPath and then further process them by a DOM library

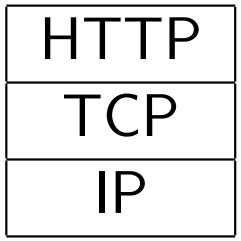
```
import xml.etree.ElementTree as ET
root = ET.parse("myFile.xml")

# find by xpath
for node in root.findall(
  "./lecture[@title='xml']/day"):
  print node.text
```

- etree supports only a subset of XPath
- full xpath support with library lxml

Section 2

Web Services



- HTTP protocol is on top of TCP/IP stack
- defines the communication between **client** and **server** in web applications
- client sends a **request** to the server, consisting of a **header** and a **body**
- server sends a **response** to the client, consisting of a **header**, **body** and **status code**

Request

- method (GET, POST, HEAD, PUT, DELETE, PATCH, ...)
- URI (scheme, host, path, query/parameters)
- (optional) cookies
- (optional) key,value pairs (POST)
- (optional) uploaded files
- header contains the length, encoding and type of the data and much more

Response

- status code (e.g. 404 Page not Found)
- header with content length, type, encoding
- body containing xml, json, image or other stuff

REST

- a web service manages resources
- a resource has one or many URIs
- HTTP method describes the action:
  - GET: read the resource
  - POST: update the resource
  - PUT: create a new resource
  - DELETE: remove a resource
- usually only GET and POST are used. POST may be also used for read-only complex queries

```
import httplib as http
import json
# get molecular formula of compound with ID 1000
url = "pubchem.ncbi.nlm.nih.gov"
path="/rest/pug/compound/cid/"+
 "1000/property/MolecularFormula/json"
conn = httplib.HTTPConnection(url)
conn.request("GET",path)
res = conn.getresponse()
doc = json.load(res)
```

SOAP

- simple object access protocol
- client invokes functions on server, send or receive objects
- function and object definitions are described in **XML**
- extremely verbose, complicated and not human readable
- but: machine readable! protocol is automatically generated on server and client side
- **wsdl** file describes protocol (is xml itself)
- python library **suds**

```
from suds.client import Client
# url to wsdl file
url="http://www.chemspider.com/" +
  "MassSpecAPI.asmx?WSDL"
# autogenerate client
client = Client(url)

# print all functions and datatypes
print client

# invoke function
ids = client.service.SearchByFormula("C6H12O6")
```

Take home messages

- csv, json and xml as text-based data formats
- use DOM like api for small xml and SAX api for large xml files
- REST: simple GET requests over HTTP
- SOAP: extremely complicated. Use a SOAP library instead of writing requests yourself