DIS08 – Data Modeling

05 – Common data file formats: CSV, JSON, and XML

Philipp Schaer, Technische Hochschule Köln, Cologne, Germany

Version: WS 2021

Disclaimer

This lesson is partially based on the Library Carpentry

- https://librarycarpentry.org/lc-spreadsheets/

Chapter 14 of Automating the Boring Stuff

And some slides by Kai Dührkop

- https://bio.informatik.uni-jena.de/wp/wp-content/uploads/2015/03/web.pdf

Agenda

Last week

- Regular expressions
- Mining or searching in files

This week

- Common data formats
- CSV
- JSON
- XML
- Outlook: Navigating the XML tree with XPath

Next week

- Open Data principles
- Data cleaning in Excel in a nutshell
- Open Refine

Data formats

Binary

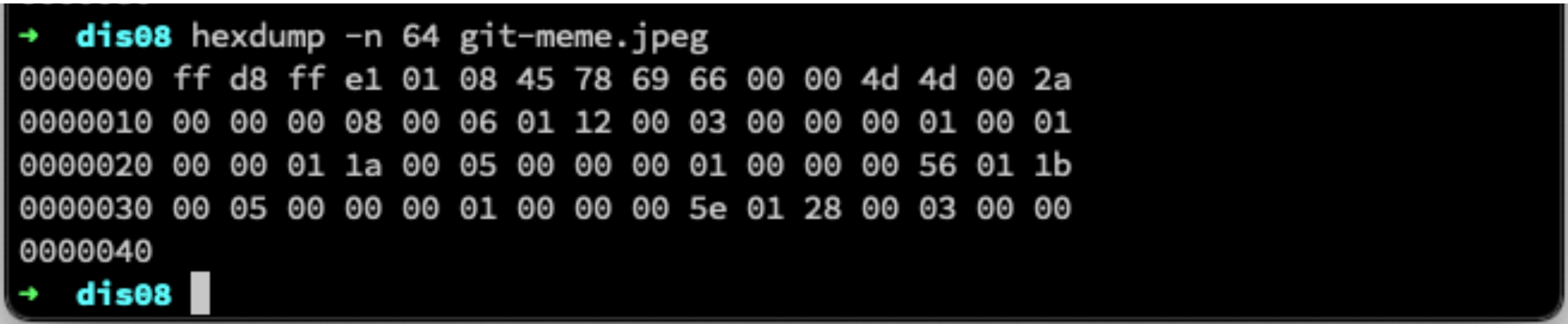
- not human readable
- memory efficient and fast to parse
- but: platform-dependent
- difficult to convert the format (e.g. open a Word Document in Open Office…)

Text

- (mostly) human readable
- waste more memory and relatively slow to parse
- platform-independent (but: still encoding problems!)
- format can be easily transformed

Binary data

- Binary files are usually thought of as being a **sequence of bytes**, which means the binary digits (bits) are grouped in eights.
- Binary files typically contain bytes that are intended to be interpreted as something other than text characters.
- Some binary files contain **headers** to interpret the data in the file. The header often contains a signature or **magic number** which can identify the format.
- JPEG magic numbers: **`ff d8 ff e0`** or **`ff d8 ff e1`**



```
→  dis08 hexdump -n 64 git-meme.jpeg
0000000 ff d8 ff e1 01 08 45 78 69 66 00 00 4d 4d 00 2a
0000010 00 00 00 08 00 06 01 12 00 03 00 00 00 01 00 01
0000020 00 00 01 1a 00 05 00 00 00 01 00 00 00 56 01 1b
0000030 00 05 00 00 00 01 00 00 00 5e 01 28 00 03 00 00
0000040
→  dis08
```

https://www.filesignatures.net/index.php?search=jpg&mode=EXT

Common (text) data formats

- **CSV**: simplest format possible: Just strings separated by commas and newlines.
- **JSON**: uses javascript syntax. Much better readable and more sparse than XML.
- **XML**: most common text data format. XHTML is the language of the web.

CSV - Comma seperated values #1

```
4/5/2015 13:34,Apples,73
4/5/2015 23:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 08:59,Oranges,52
```

- Column separator:
  - default is comma (“,” → CSV)
  - but also tabulator (“\t” → TSV)
  - very common row separator: usually newline (“\n”)
- Strings can be enclosed in quotation marks to escape special characters
- Quotation marks are escaped by another quotation marks

CSV - Comma seperated values #2

- The advantage of CSV files is **simplicity**.
- CSV files are **widely supported** by many types of programs, can be viewed in text editors, and are a straightforward way to represent data.
- Whenever your data has **no nested structure**: **USE CSV**!
- Can be easily read in every programming 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.

JSON - JavaScript Object Notation

```
[
  {
  "name": "Glucose",
  "formula": "C6H12O6",
  "similarTo": ["Hexose", "Fructose"]
   }
]
```

- Just Javascript data. Human readable but not easy to parse.
- Popular use: Client and web server often use JSON to communicate. Javascript can naturally work with JSON.
- Also almost compatible to Python’s syntax of booleans, numbers, strings, arrays, objects (dictionaries).

JSON Basics

JSON is built on two structures:

- A **collection of name/value pairs**. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An **ordered list of values**. In most languages, this is realized as an array, vector, list, or sequence.

- All details can be found online: https://www.json.org

JSON Objects

- An object is an unordered set of name/value pairs. An object begins with **{** (left brace) and ends with **}** (right brace). Each name is followed by **:** (colon) and the name/value pairs are separated by **,** (comma).

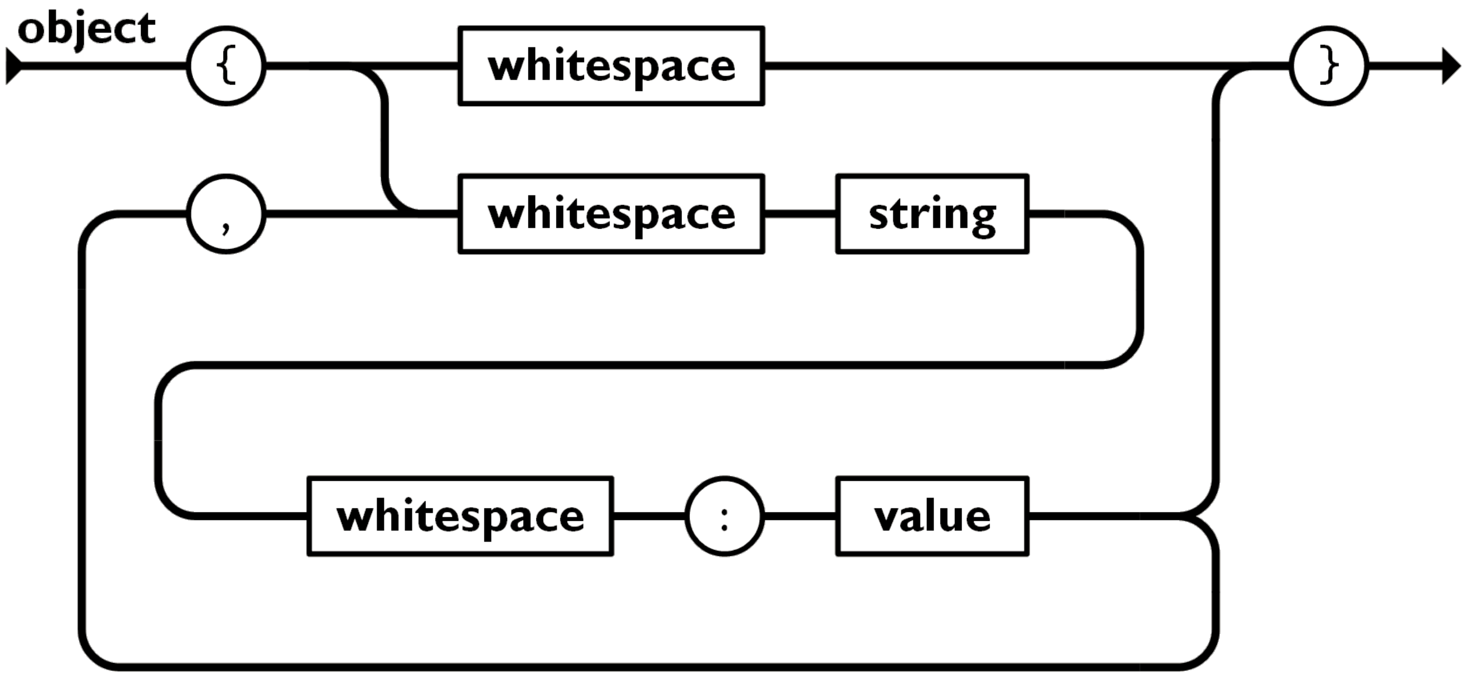


Image source: https://www.json.org

JSON Arrays

- An array is an ordered collection of values. An array begins with **[** (left bracket) and ends with **]** (right bracket). Values are separated by **,** (comma).

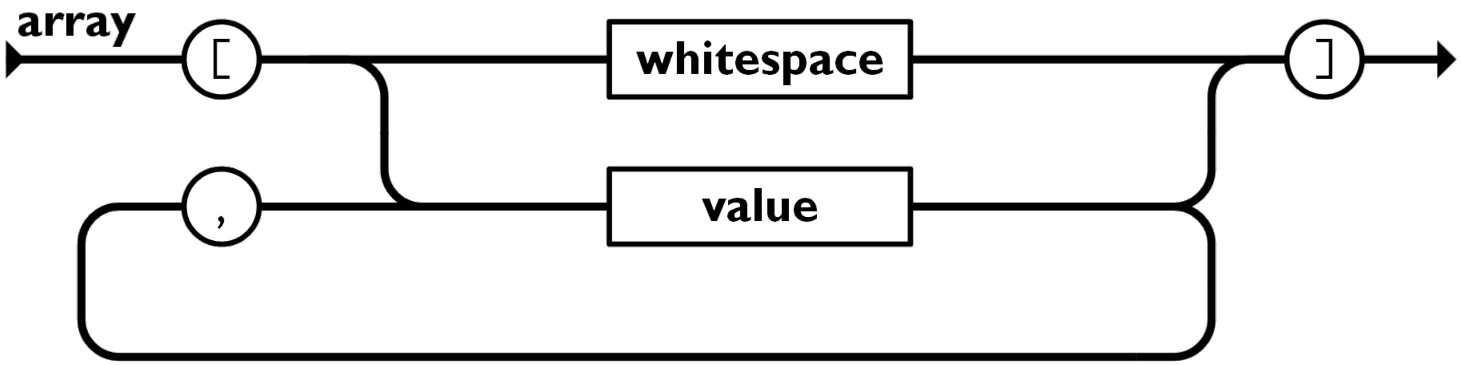
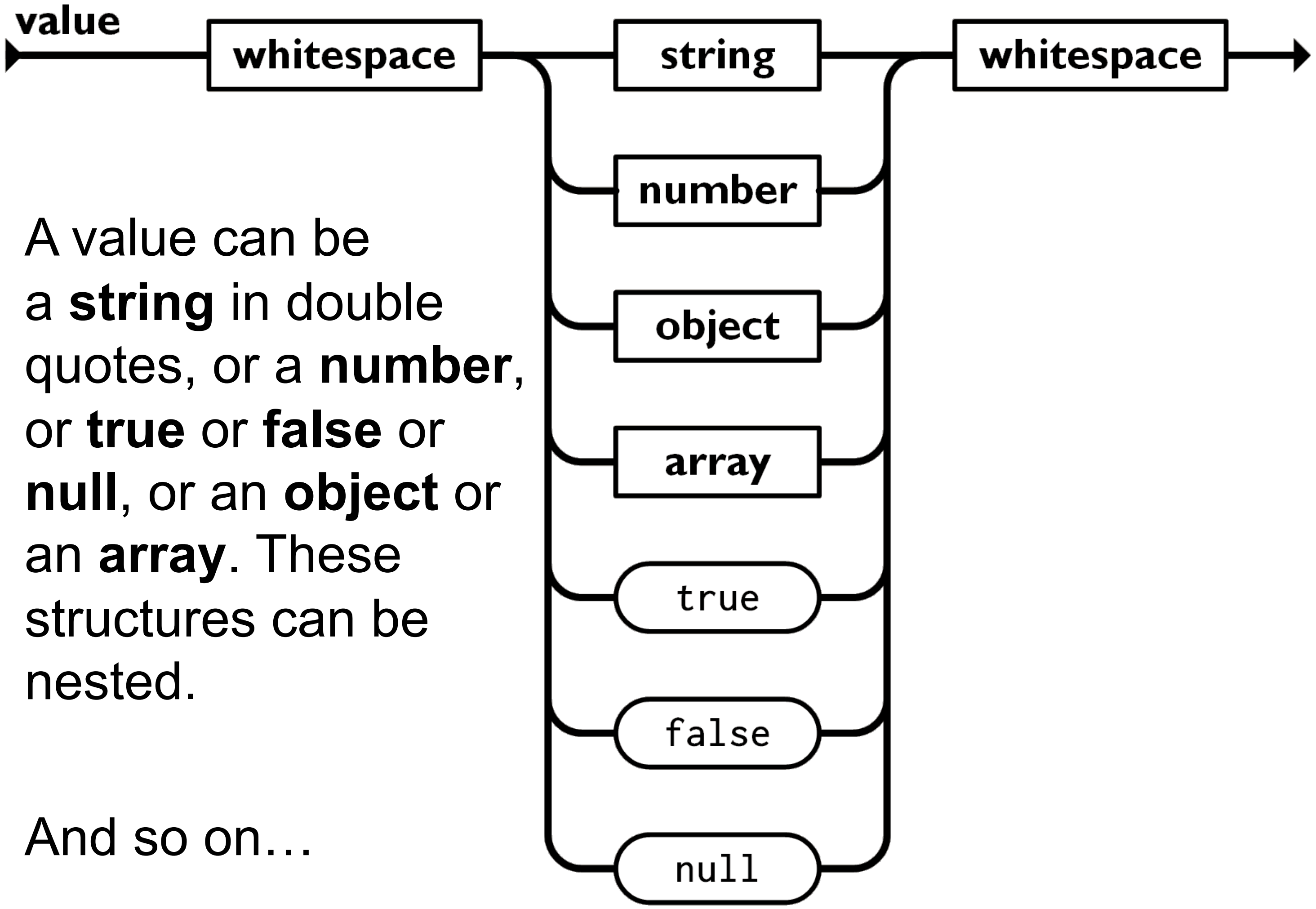


Image source: https://www.json.org

JSON Values



- A value can be a **string** in double quotes, or a **number**, or **true** or **false** or **null**, or an **object** or an **array**. These structures can be nested.
- And so on…

Image source: https://www.json.org

A little bit more complex example

```
{
  "firstName": "Philipp",
  "hobbies": ["Mac", "Python", "dank memes",
              "BBQ"],
  "age": 41,
  "children": [
    { "firstName": "Primus", "age": 5 },
    { "firstName": "Secundus", "age": 3 },
    { "firstName": ”Tertius", "age": 1 },
  ]
}
```

Still not mighty enough? XML for the win!

XML defines a **syntax** to provide **structured data** sets of **any kind** with simple, understandable **markups**, which can be evaluated by applications of various kinds.

What is only indirectly stated here:

- XML is the **eXtensible Markup Language**.
- XML is an international **W3C standard**.
- XML documents are human **and** machine readable!
- XML is a document **description language.**
- XML **separates structure** from **presentation**, or **content** from **presentation.**
- XML documents are developed according to a **document model**.

All you need to know about XML…

An XML document may contain:

- **Elements**, possibly with attributes
- **Processing instructions**
- **Comments**
- **Entity references**

An XML document must be **well-formed** and can be **validated**.

- XML attribute values must be in **“** (double quotes).
- XML documents are encoded as linear strings.
- XML documents begin with a special processing instruction, the **prologue**.

All you need to know about XML…

root

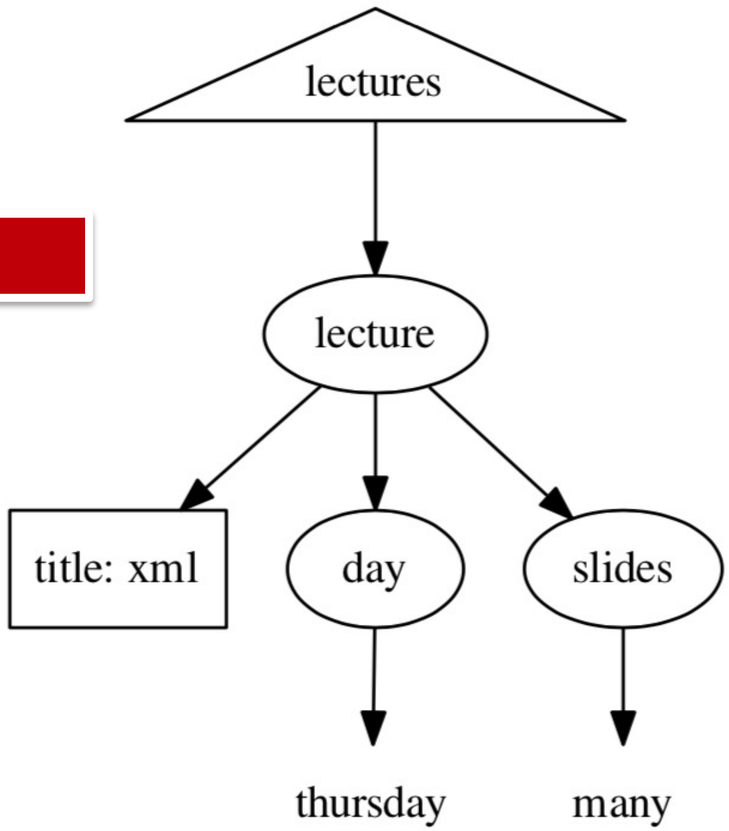
attribute

element

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

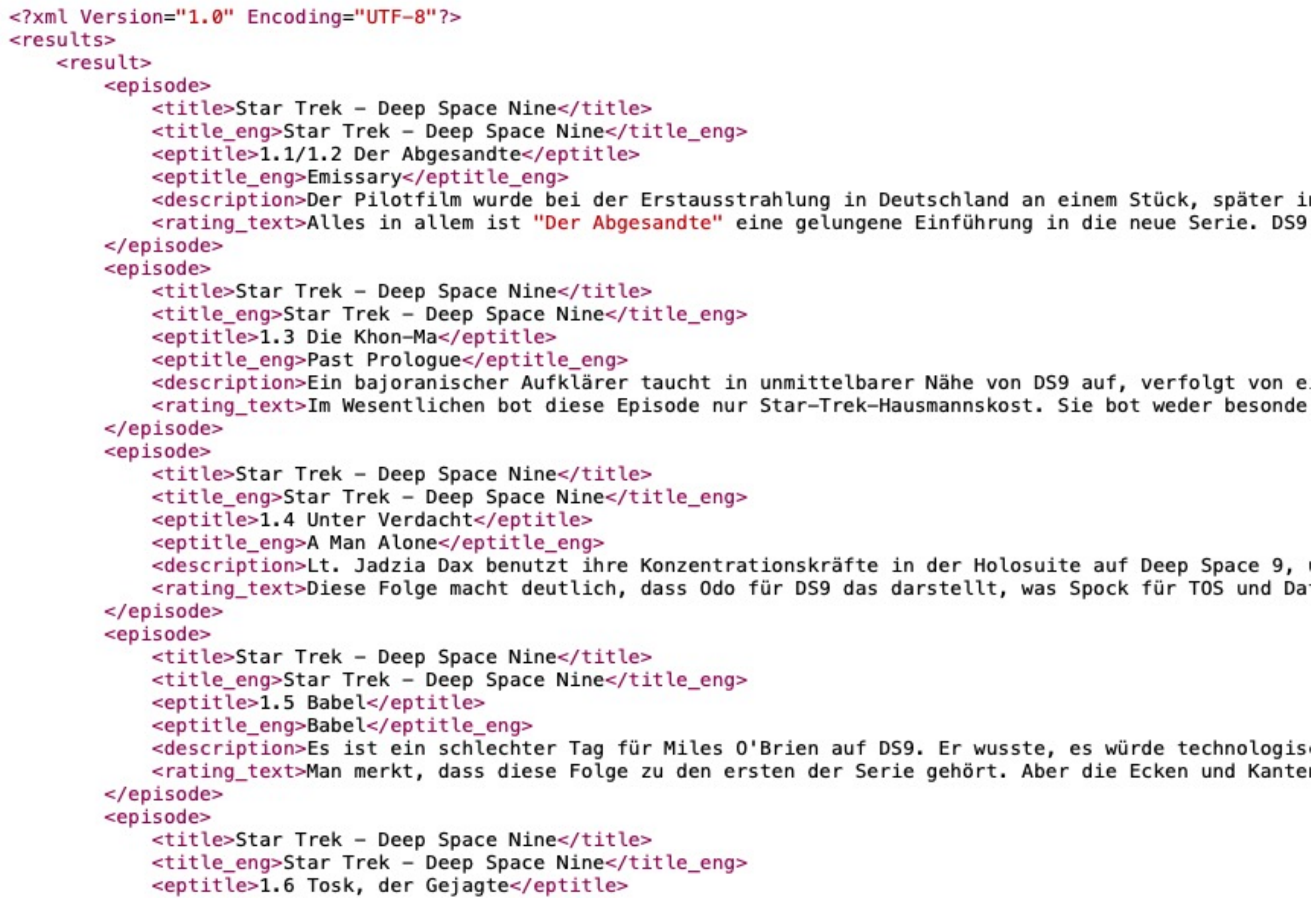
value

closing tag



- Very common data format
- Solves a lot of problems (namespaces, encoding, embedded data)
- Not very readable, very verbose

A real-world XML example



```
<?xml Version="1.0" Encoding="UTF-8"?>
<results>
    <result>
        <episode>
            <title>Star Trek - Deep Space Nine</title>
            <title_eng>Star Trek - Deep Space Nine</title_eng>
            <eptitle>1.1/1.2 Der Abgesandte</eptitle>
            <eptitle_eng>Emissary</eptitle_eng>
            <description>Der Pilotfilm wurde bei der Erstausstrahlung in Deutschland an einem Stück, später i
            <rating_text>Alles in allem ist "Der Abgesandte" eine gelungene Einführung in die neue Serie. DS9
        </episode>
        <episode>
            <title>Star Trek - Deep Space Nine</title>
            <title_eng>Star Trek - Deep Space Nine</title_eng>
            <eptitle>1.3 Die Khon-Ma</eptitle>
            <eptitle_eng>Past Prologue</eptitle_eng>
            <description>Ein bajoranischer Aufklärer taucht in unmittelbarer Nähe von DS9 auf, verfolgt von e
            <rating_text>Im Wesentlichen bot diese Episode nur Star-Trek-Hausmannskost. Sie bot weder besonde
        </episode>
        <episode>
            <title>Star Trek - Deep Space Nine</title>
            <title_eng>Star Trek - Deep Space Nine</title_eng>
            <eptitle>1.4 Unter Verdacht</eptitle>
            <eptitle_eng>A Man Alone</eptitle_eng>
            <description>Lt. Jadzia Dax benutzt ihre Konzentrationskräfte in der Holosuite auf Deep Space 9,
            <rating_text>Diese Folge macht deutlich, dass Odo für DS9 das darstellt, was Spock für TOS und Da
        </episode>
        <episode>
            <title>Star Trek - Deep Space Nine</title>
            <title_eng>Star Trek - Deep Space Nine</title_eng>
            <eptitle>1.5 Babel</eptitle>
            <eptitle_eng>Babel</eptitle_eng>
            <description>Es ist ein schlechter Tag für Miles O'Brien auf DS9. Er wusste, es würde technologis
            <rating_text>Man merkt, dass diese Folge zu den ersten der Serie gehört. Aber die Ecken und Kante
        </episode>
        <episode>
            <title>Star Trek - Deep Space Nine</title>
            <title_eng>Star Trek - Deep Space Nine</title_eng>
            <eptitle>1.6 Tosk, der Gejagte</eptitle>
```

A well-formed XML document

An **element** always has a **start tag** and an **end tag**.

- This is a **<StartTag>.**
- This is the end of the **</StartTag>**.
- The tag names are **case-sensitive**.
- <tag></tag> and not <tag></Tag>.

Empty elements, called milestone elements, can be abbreviated

- <emptyTag />.
- Empty means that attributes may be included, but no content may be placed between the tags

A well-formed XML document

All elements must be **nested correctly**!

```
<up>
  <down>Text</down>
</up>
```

```
<up>
<down>Text</up>
</down>
```

Element names

- must begin with a letter, underscore, or colon.
- can contain letters, numbers, hyphens, periods, or underscores, as well as umlauts and accents.

An XML document has only **exactly one root** node!

Attributes

Elements can be defined in more detail by **attributes**.

- Attributes are always in the start tag!
- `<name AttrName="Attributwert"> </name>`

Properties of attributes

- When modeling documents, it must always be weighed which information is **element-worthy** and which is **attribute-worthy**.
- Multiple attributes are allowed per element.
- Same named attributes in different elements

```
<book>
   <person role="author">Stephen King</person>
   <person role="translator">Ralph Meier</person>
</book>
```

XML Prolog

An XML document always starts with a prolog that contains

- the XML declaration
  - `<?xml version="1.0"?>`
  - `<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>`
- processing information (optional)
  - `<?xsl-stylesheet type="text/xsl" href="myown.xsl"?>`
- Embedding of the document model (optional)
  - DTD, XML Schema, RelaxNG, Schematron
  - `<!DOCTYPE tei SYSTEM "tei.dtd">`

Entities

Entities stand for **something else**. They are used to **avoid conflicts** during XML processing!

- XML allows to set up a reference that points to an entity.
- The XML processor **replaces** the references.
- There are some predefined entity references e.g. special characters:

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

Special characters

character	notation in XML	comment
<	&lt;	
>	&gt;	only problematic in tags and in ']]>'
&	&amp;	
"	&quot;	only problematic in attribute values with "…"
'	&apos;	only problematic in attribute values with '…'

This element is encoded as
&lt;code&gt;&lt;Element&gt;...&lt;/Element&gt;&lt;/code&gt;

This element is encoded as
<code><Element>...</Element></code>

XML verification

The verification is done by an XML parser.

- An XML document **must be well-formed** and **can be valid**.

Check XML document for **well-formedness**

- An XML document is well-formed if it **complies with the rules of the XML standard**.

Check XML document for **validity**

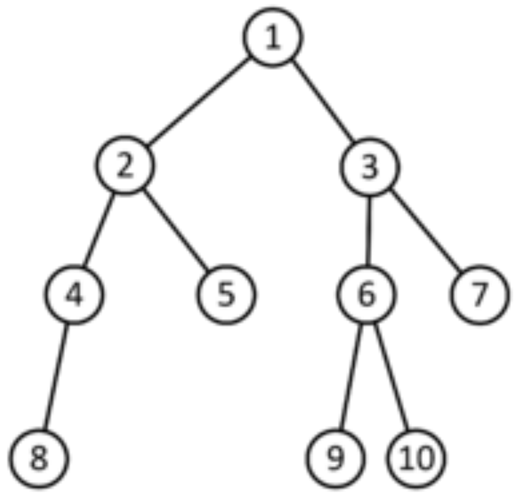
- An XML document is valid if it is **well-formed and conforms to the grammar** of the XML schema.

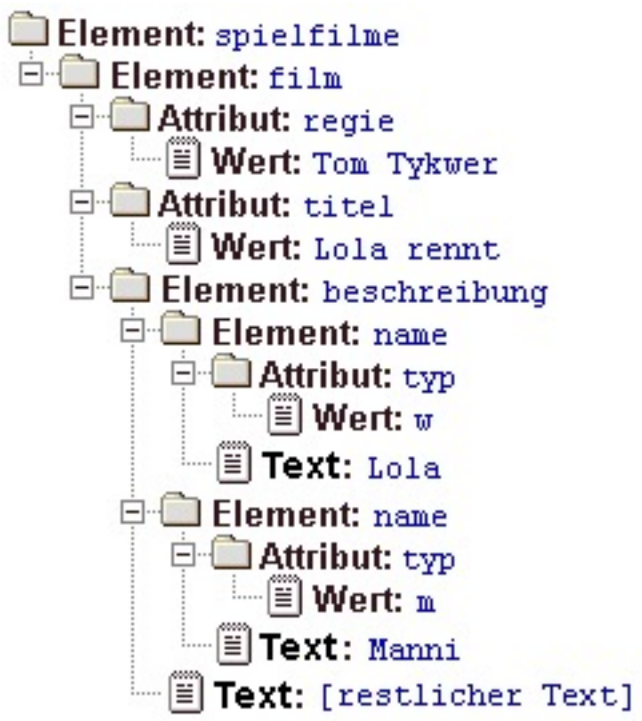
But let‘s face it: XML is often very, very complex…

XML as a tree structure

```
<spielfilme>
 <film regie="Tom Tykwer" titel="Lola rennt">
  <beschreibung>
    <name typ="w">Lola</name> rennt für <name typ="m">Manni</name>, der 100000 Mark
    liegengelassen hat und noch 20 Minuten Zeit hat, das Geld auszuliefern.
  </beschreibung>
 </film>
</spielfilme>
```

XML parser





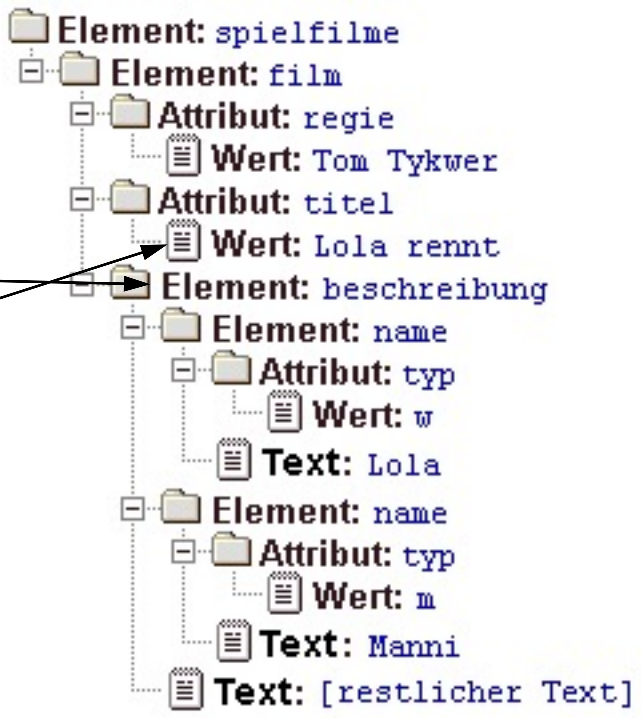
```
Element: spielfilme
  Element: film
    Attribut: regie
      Wert: Tom Tykwer
    Attribut: titel
      Wert: Lola rennt
    Element: beschreibung
      Element: name
        Attribut: typ
          Wert: w
        Text: Lola
      Element: name
        Attribut: typ
          Wert: m
        Text: Manni
      Text: [restlicher Text]
```

XPath to navigate the tree

```
<spielfilme>
 <film regie="Tom Tykwer" titel="Lola rennt">
  <beschreibung>
    <name typ="w">Lola</name> rennt für <name typ="m">Manni</name>, der 100000 Mark
    liegengelassen hat und noch 20 Minuten Zeit hat, das Geld auszuliefern.
  </beschreibung>
 </film>
</spielfilme>
```

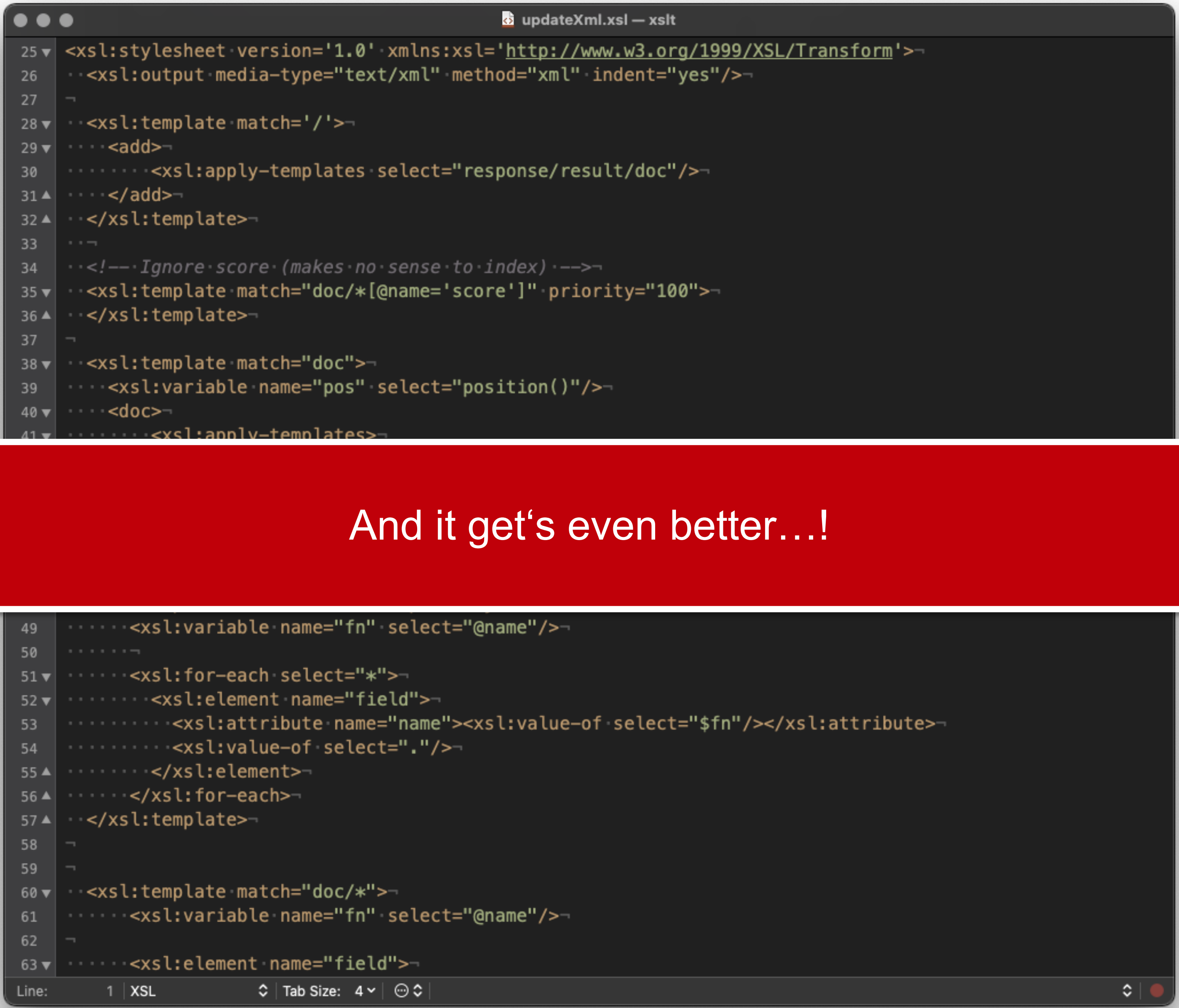
XPath example

/film/beschreibung
/film[@regie='Tom Tykwer']/@titel
...



JSON vs XML

JSON	XML
It is based on JavaScript language.	It is derived from SGML.
It is a way of **representing objects**.	It is a markup language and uses tag structure to represent data items.
It does not provides any support for **namespaces**.	It supports namespaces.
It supports **arrays**.	It doesn't supports arrays.
Its files are very **easy to read** as compared to XML.	Its documents are comparatively difficult to read and interpret.
It doesn't use end tag.	It has start and end tags.
It is less **secured**.	It is more secured than JSON.
It doesn't supports **comments**.	It supports comments.
It supports only UTF-8 **encoding**.	It supports various encoding.



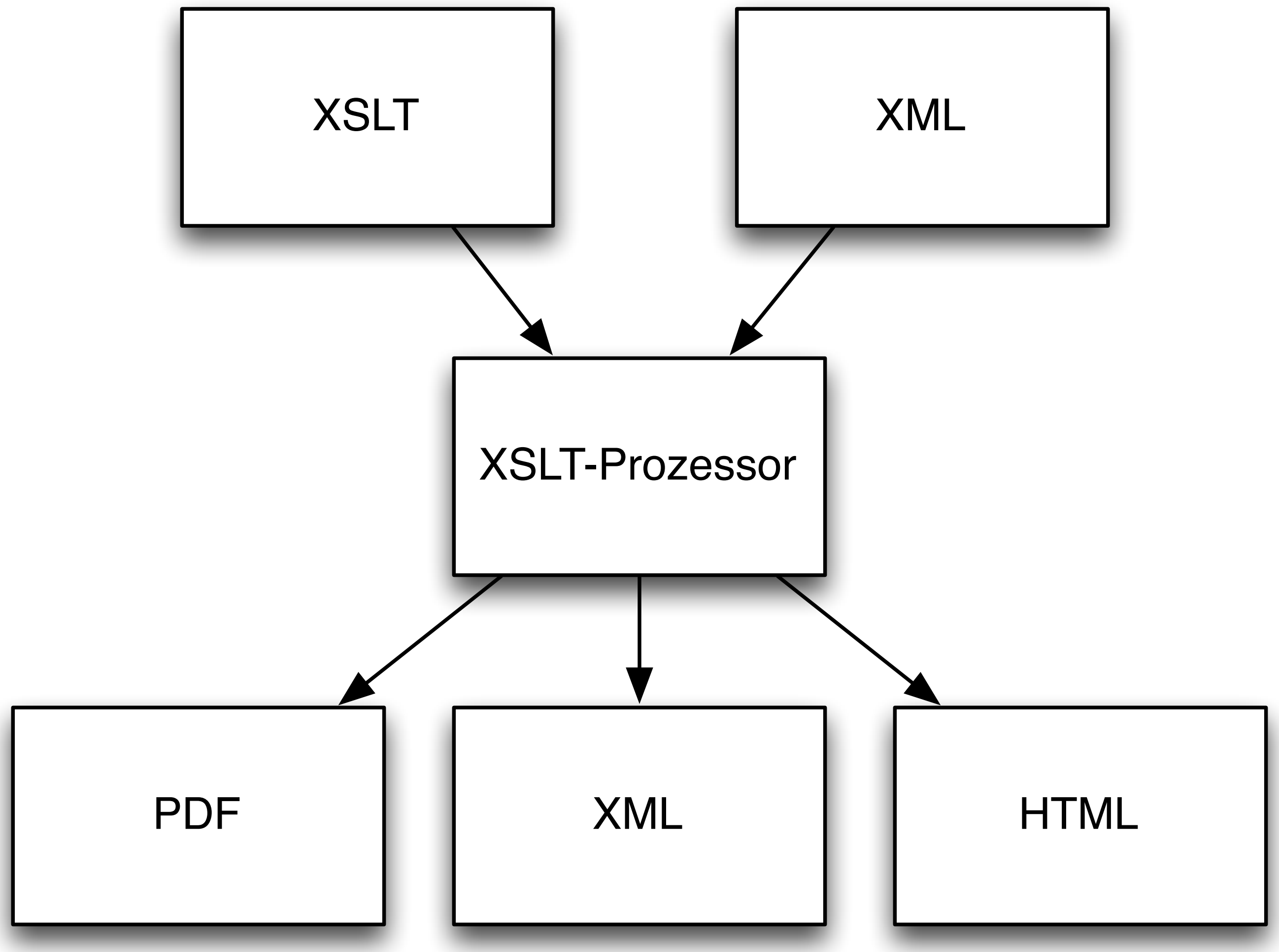
```
<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:output media-type="text/xml" method="xml" indent="yes"/>

  <xsl:template match='/'>
    <add>
        <xsl:apply-templates select="response/result/doc"/>
    </add>
  </xsl:template>

  <!-- Ignore score (makes no sense to index) -->
  <xsl:template match="doc/*[@name='score']" priority="100">
  </xsl:template>

  <xsl:template match="doc">
    <xsl:variable name="pos" select="position()"/>
    <doc>
        <xsl:apply-templates>
```

```
      <xsl:variable name="fn" select="@name"/>

      <xsl:for-each select="*">
        <xsl:element name="field">
          <xsl:attribute name="name"><xsl:value-of select="$fn"/></xsl:attribute>
          <xsl:value-of select="."/>
        </xsl:element>
      </xsl:for-each>
  </xsl:template>


  <xsl:template match="doc/*">
      <xsl:variable name="fn" select="@name"/>

      <xsl:element name="field">
```

Line: 1 XSL Tab Size: 4

And it get‘s even better…!

XSLT – One source, many targets



Take away message

- CSV, JSON, and XML are typical text-based file formats that **you WILL encounter in the wild**.
- All have their **use cases and applications** and it always depends on what you would like to accomplish!
- It’s good practice to **know all three** - one of these is usually available and applicable.