# Textkodierung und Textanalyse mit TEI, UE

## XML Schemasprachen und RELAX NG

Christopher Pollin
*Zentrum für Informationsmodellierung - Austrian Centre for Digital Humanities*

# Überblick

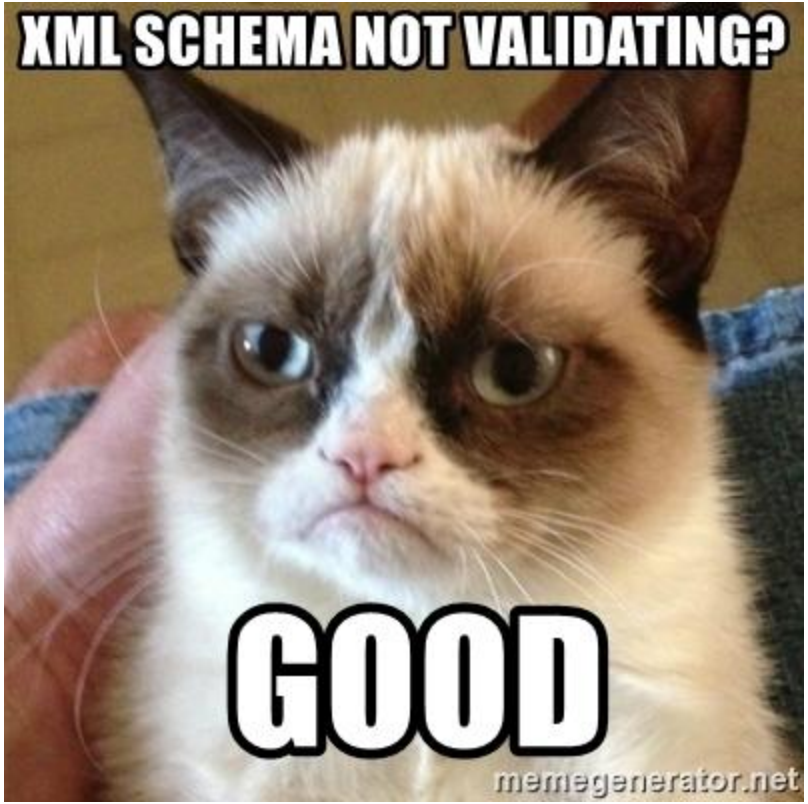
**Wohlgeformtes vs. valides XML**

**RELAX NG**

**Eine Grammatik/Schema für “Memos”**

**Übung**

**Assignment 1**


XML SCHEMA NOT VALIDATING?
GOOD
memegenerator.net

# Wohlgeformtes vs. valides XML

XML Dokumente müssen **wohlgeformt** und **valide** sein !!!

**Wohlgeformt**

Das XML entspricht den Regeln für XML Dokumente **(Syntax)**

- Ein Wurzelelement
- Element- und Attributnamen entsprechen den festgesetzten Regeln
- Elemente dürfen nicht überlappen
- Anfangs- und Endtag müssen übereinstimmen
- Groß- und Kleinschreibung

**Valide**

Dem XML ist ein DTD, XML Schema oder RELAX NG zugeordnet gegen. Es kann validiert werden **(Semantik)**.

- Ein Schema beschreibt die Struktur eines XML Dokuments
- Die Namen für Elemente und Attribute werden definiert
- Inhaltsmodelle für Elemente werden beschrieben: kein Inhalt (leeres Element), Text, gemischter Inhalt (Text und Elemente), nur Elemente
- Attribute und Attributwerte werden definiert

# **RELAX NG**
# (REgular LAnguage for XML Next Generation)

- ~ relaxing
- RELAX NG ist eine  Schemasprache für XML.
  - http://books.xmlschemata.org/relaxng/
- steht im Grad der Komplexität zwischen Document Type Definition (DTD) und XML Schema.
- (wahlweise) XML-Syntax und eine “Compact non-XML syntax”
  - https://www.tei-c.org/release/doc/tei-p5-doc/en/html/SG.html
- Unterstützt ungeordnete Inhalte, Mixed Content und  Datentypen und Namespaces.

  - https://en.wikipedia.org/wiki/RELAX_NG

# Live-Demo

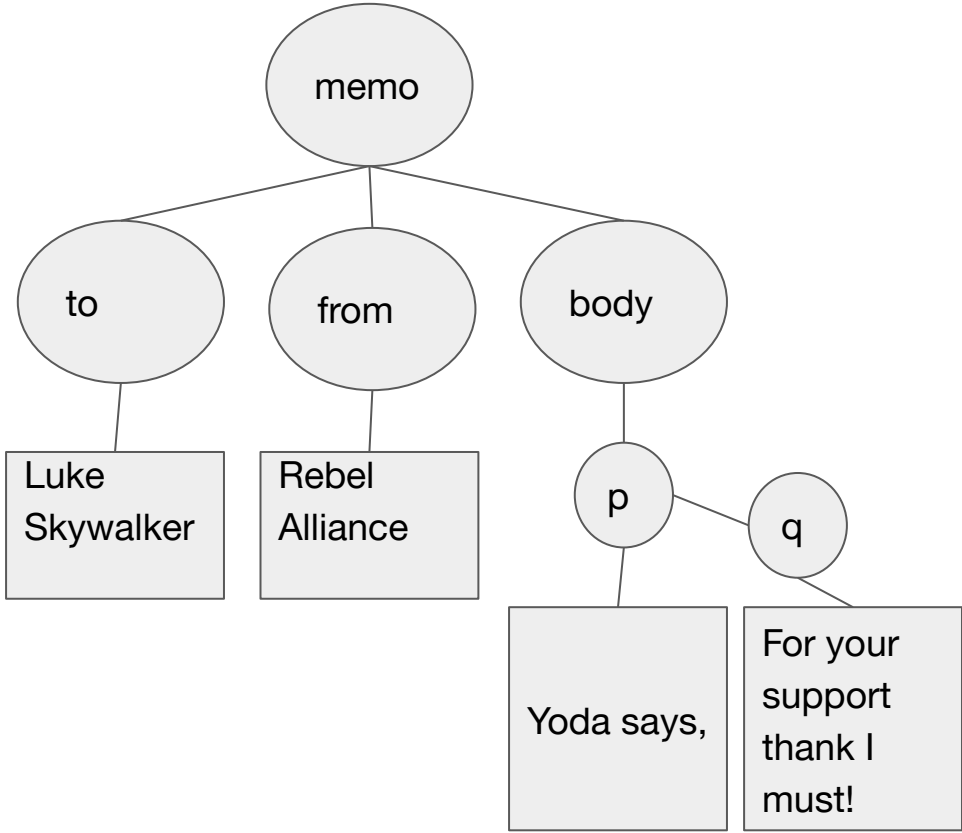
## Folgender Foliensatz als Dokumentation

Referenzen:

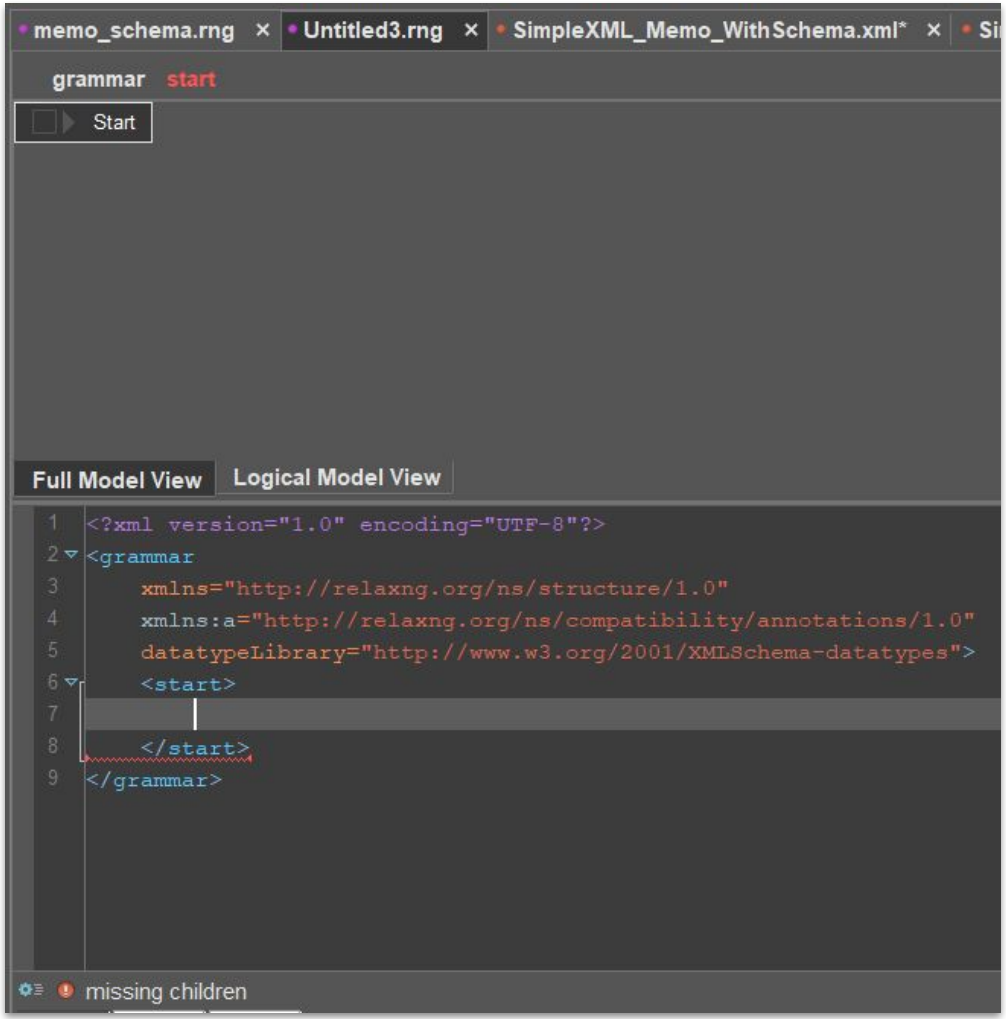
- http://books.xmlschemata.org/relaxng/page2.html
- https://speedata.github.io/relaxngtutorial-de/

# RELAX NG für “Memos”

```
<?xml version="1.0"?>
<memo>
  <to>Luke Skywalker</to>
  <from>Rebel Alliance</from>
  <body>
 <p>Yoda says,
         <q>
         For your support thank
         I must!
         </q>
       </p>
     </body>
</memo>
```



- Oxygen Editor: `File/New.../RELAX NG Schema - XML`
- Nun kann man sein .rng bearbeiten.
- Wir wollen eine Grammatik (Schema) für unsere Memos definieren.
    - Oxygen “hilft” uns beim erfassen von neuen Memos
    - Man kann überprüfen ob unsere Memos richtig sind



**<element name="memos">**

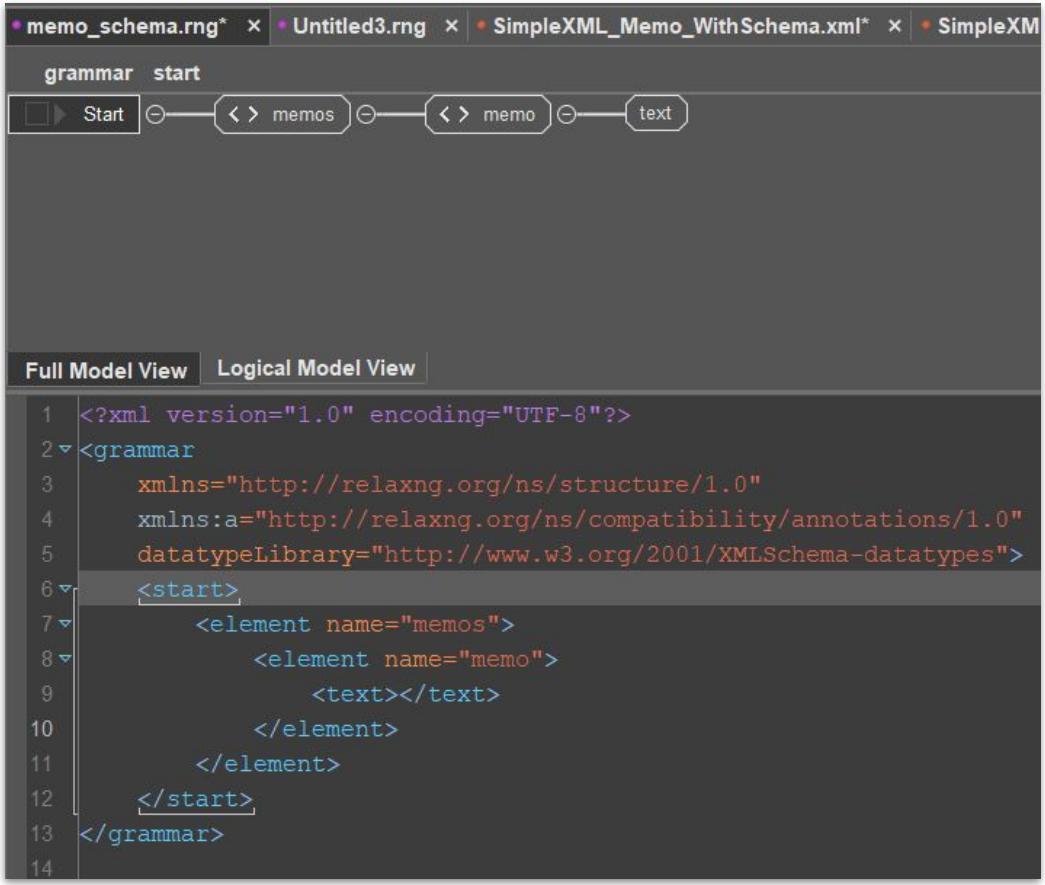
Definiert ein neues (verpflichtendes) Element `<memos>`

**<text/>**

Definiert, dass an dieser Stelle sich Text befinden darf.

```
<element name="memos">
    <element name="memo">
        <text/>
    </element>
</element>
```

Baumstruktur definieren. In einem `<memos>` muss sich ein `<memo>` befinden, in dem ein `text` steht.



Neues XML erzeugen

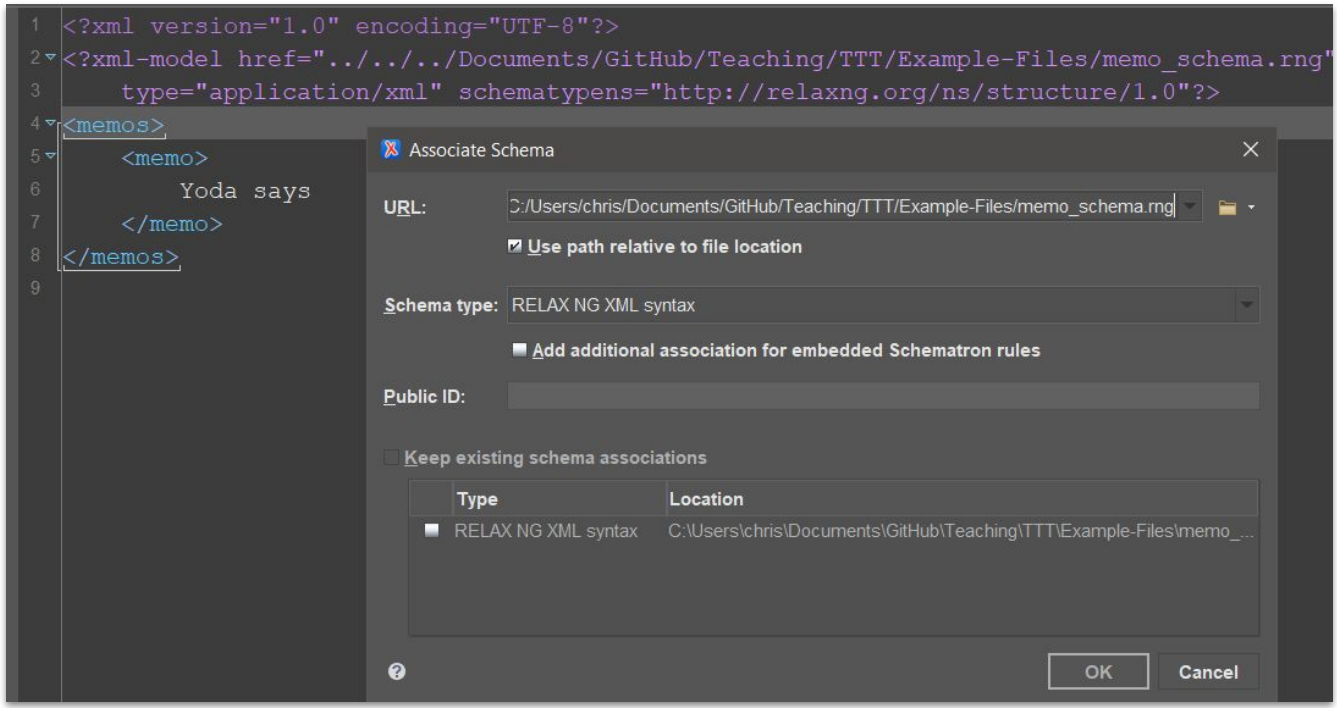
Wurzelelement anlegen.

`Document/Schema/Associate Schema`

Über URL den Pfad angeben wo das .rng liegt.

Wir haben nun unser XML Dokument mit dem Schema verknüpft.

So ist auch ein XML-Schema/RELAX NG mit TEI verknüpft



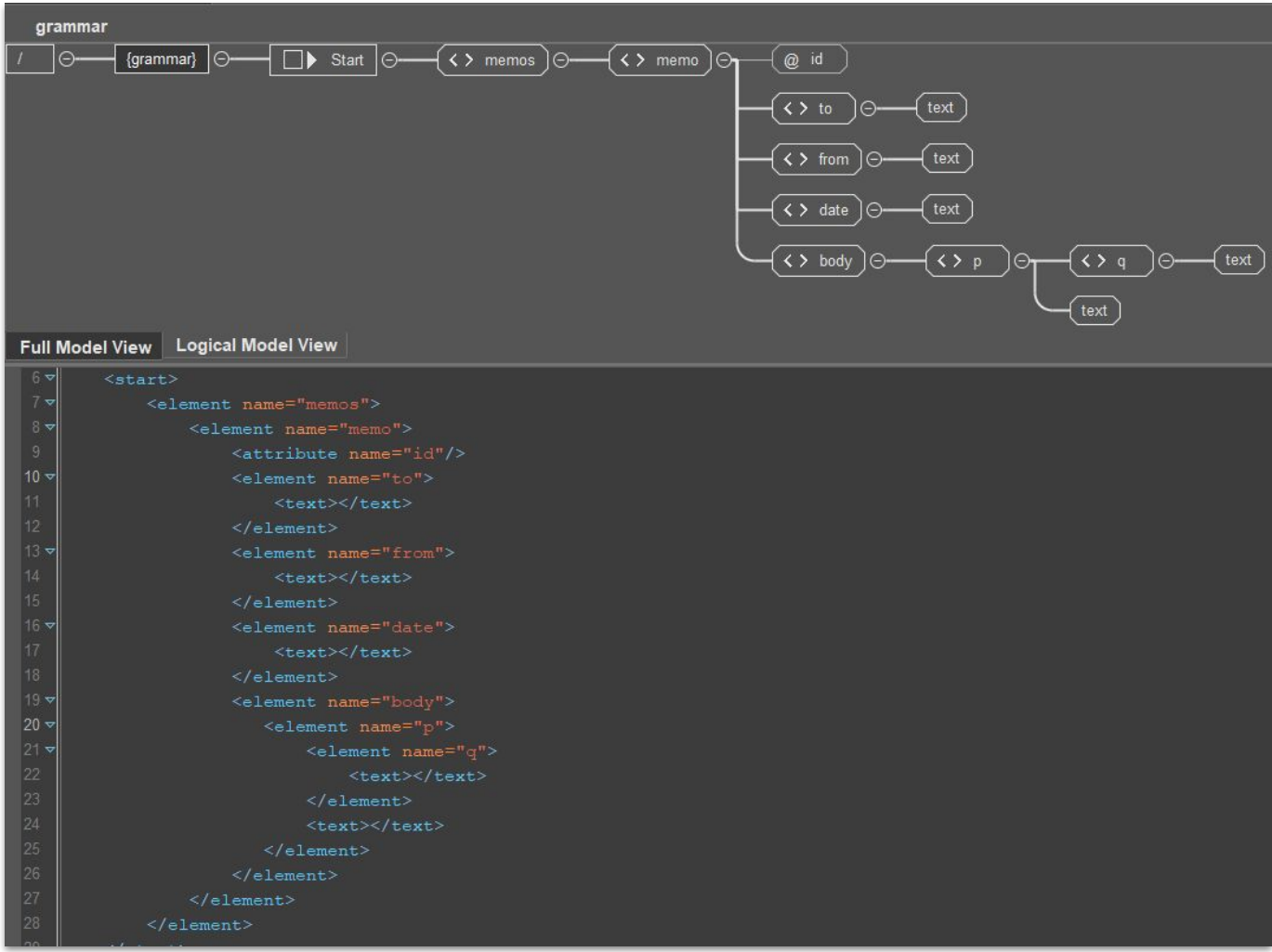

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="http://www.tei-c.org/release/xml/tei/custom/schema/relaxng/tei_jtei.rng"
   type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<?xml-model href="http://www.tei-c.org/release/xml/tei/custom/schema/relaxng/tei_jtei.rng"
   type="application/xml" schematypens="http://purl.oclc.org/dsdl/schematron"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0" rend="jTEI">
  <teiHeader>
     <fileDesc>
        <titleStmt>
```

Hinzufügen der Elemente `<to>`, `<from>`, `<date>` und <body> als Kindelement von <memos>.

Alle beinhalten text, `<body>` beinhaltet ein `<p>`, das wiederum ein `<q>` und text beinhaltet.

**`<attribute name="id"/>`**
Fügt ein Attribut id zu `<memo>` hinzu.

Oxygen zeigt uns sie Baumstruktur an.



**<oneOrMore>**
**<element name="memo">**
**[...]**
**</oneOrMore>**

Definiert, dass was sich innerhalb von <oneOrMore> befindet ein oder mehrmals auftreten kann.
→ wir also beliebig viele <memo> in <memos> geben können.

**<optional>**
**<element name="q">**
**[...]**
**</optional>**

Definiert, dass was sich innerhalb von <optional> befindet vorkommen kann, aber nicht muss.

```
<oneOrMore>
    <element name="memo">
        <attribute name="id"/>
        <element name="to">
            <text></text>
        </element>
        <element name="from">
            <text></text>
        </element>
        <element name="date">
            <text></text>
        </element>
        <element name="body">
           <element name="p">
               <optional>
                    <element name="q">
                        <text></text>
                    </element>
               </optional>
               <text></text>
           </element>
        </element>
    </element>
</oneOrMore>
```

Das können wir mit unserem .rng erzeugen.

Nur beim `<p>` haben wir noch einen Fehler, der dazu führt, dass unser XML nicht valide ist.

Unser .rng definiert, dass zuerst ein `<q>` kommen kann (optional) und dann ein `text`; aber nicht zuerst `text` und dann ein `<q>` [→ **Mixed Content**]

Auch könnten wir in `<date>` “hallo” reinschreiben und es wäre valide, wobei es sinnvoll wäre restriktiv nur Datumsangaben zuzulassen. Auch soll es optional sein.

`<to>`, `<from>` und `@id` sollen nicht leer sein dürfen.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="../../../Documents/GitHub/Teaching/T
    type="application/xml" schematypens="http://relaxn
<memos>
    <memo id="">
        <to></to>
        <from></from>
        <date></date>
        <body>
            <p></p>
        </body>
    </memo>
    <memo id="">
        <to></to>
        <from></from>
        <date></date>
        <body>
            <p>
                Hallo <q>C3PO</q>!
            </p>
        </body>
    </memo>
</memos>
```

**`<data type="string"/>`**
Definiert, dass hier ein bestimmter Datentyp angegeben werden muss (*Build-in primitive type)*.
Dieser Datentyp ist string, also Text

**`<data type="date"/>`**
Der Datentyp `date` ist so definiert:
YYYY-MM-DD, z.B. 2020-10-18

**`<param name="pattern">memo.+</param>`**
Hier definieren wir ein Pattern, das weiter bestimmt wie der Inhalt des Attributs oder Elements aussehen muss.

**`<param name="pattern">.+</param>`**
. … beliebiges Zeichen
\+ … mindestens 1 (Quantor, wie bei RegEx)
Sorgt dafür, dass es nicht leer sein darf.

```
<element name="memo">
    <attribute name="id">
        <data type="string">
            <param name="pattern">memo.+</param>
        </data>
    </attribute>
    <!-- to -->
    <element name="to">
        <data type="string">
            <param name="pattern">.+</param>
        </data>
    </element>
    <!-- from -->
    <element name="from">
        <data type="string">
            <param name="pattern">.+</param>
        </data>
    </element>
    <!-- date -->
    <optional>
     <element name="date">
         <data type="date"/>
     </element>
    </optional>
    <!-- body -->
    <element name="body">
       <element name="p">
           <mixed>
               <zeroOrMore>
                   <element name="q">
                       <text/>
                   </element>
               </zeroOrMore>
           </mixed>
```

```
<param name="pattern">memo.+</param>
```

Zuerst muss die Zeichenfolge “memo” kommen, dann beliebige Zeichen.

```
<element name="p">
  <mixed>
    <zeroOrMore>
 <element name="q">
   <text/>
  </element>
</zeroOrMore>
   </mixed>
</element>
```

Sorgt dafür, dass innerhalb von <p> in beliebiger Reihenfolge text neben <q> stehen muss, wobei <q> nicht vorkommen müssen, aber beliebig oft vorkommen können.

```
<element name="memo">
    <attribute name="id">
        <data type="string">
            <param name="pattern">memo.+</param>
        </data>
    </attribute>
    <!-- to -->
    <element name="to">
        <data type="string">
            <param name="pattern">.+</param>
        </data>
    </element>
    <!-- from -->
    <element name="from">
        <data type="string">
            <param name="pattern">.+</param>
        </data>
    </element>
    <!-- date -->
    <optional>
     <element name="date">
         <data type="date"/>
     </element>
    </optional>
    <!-- body -->
    <element name="body">
       <element name="p">
           <mixed>
               <zeroOrMore>
                   <element name="q">
                       <text/>
                   </element>
               </zeroOrMore>
           </mixed>
```

Wir können nun diese Memos erzeugen und validieren!


```
<memos>
    <memo id="memo.1">
        <to>R2D2</to>
        <from>C3PO</from>
        <date>1990-01-01</date>
        <body>
            <p>
                <q>Bew" Pip Pip!</q>
            </p>
        </body>
    </memo>
    <memo id="memo.2">
        <to>Luke Skywalker</to>
        <from>Rebel Alliance</from>
        <body>
            <p>Yoda says,
                <q>
                    "For your support thank
                    I must!"
                </q>
            </p>
        </body>
    </memo>
    <memo id="memo.3">
        <to>Obi-Wan Kenobi</to>
        <from>General Grievous</from>
        <body>
            <p>I've been trained in your Jedi arts by Count Dooku!</p>
        </body>
    </memo>
</memos>
```

# Zusammenfassung

- <oneOrMore> + 1… n
- <zeroOrMore> * 0… n
- <optional> ? 0… 1

- `<text/>`
- `<attribute name="id"/>`
- `<data type="string">, <data type="date"> …`
- `<param name="pattern">.+</param>`
- `<mixed>` Kurzschreibweise für `Element|Text`
- `<interleave>` `Element|Text`
- `<element name="memo">`

Alles zu RELAX NG: http://books.xmlschemata.org/relaxng/page2.html

# Übung

Erweitere `memo_schema.rng`. Lade dazu von Moodle das .zip Memo RELAX NG Datei herunter.

Folgende Spezifikationen sollen umgesetzt sein:

- Wir wollen, dass wir `<to>` und `<from>` noch weiter spezifizieren können und zwar über ein Attribut type, das ausschließlich mit dem String “Droid”, “Jedi” oder “Sith” befüllt werden kann. `@type` soll aber optional sein.
- Jedes `<memo>` soll ein `<head>` als erstes Kindelement verfügen, in dem sich ein leeres Element `<ref/>` befindet, das ein `@target` Attribut hat, das auf eine beliebige URL verweist.
- Dazu brauchst du neben den bekannten Strukturen noch:
  **`<empty/>, <data type="anyURI"/>, <choice>, <value>`**
- Du wirst merken, dass du für `<to>` und `<from>` zweimal dasselbe definierst. Man kann Patterns auch auslagern und dann referenzieren. Definiere die optionale Auswahl von @type als “Benanntes Muster”. **`<define name="Typing">, <ref name="Typing"/>`**
  Hint: https://speedata.github.io/relaxngtutorial-de/
- Füge ein weiteres valides <memo> hinzu. Lade dein Ergebnis als .zip auf Moodle hoch (dein NACHNAME.rng + NACHNAME.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-model href="memo_schema_extended.rng" type="application/xml" schematypens="http://relaxng.org/ns/structure/1.0"?>
<memos>
  <memo id="memo.1">
    <head><ref target="https://en.wikipedia.org/wiki/R2-D2"/></head>
    <to type="Droid">R2D2</to>
    <from type="Droid">C3PO</from>
    <date>1990-01-01</date>
    <body><p><q>Bew" Pip Pip!</q></p></body>
  </memo>

  <memo id="memo.2">
    <head><ref target="https://en.wikipedia.org/wiki/Luke_Skywalker"/></head>
    <to type="Jedi">Luke Skywalker</to>
    <from>Rebel Alliance</from>
    <body><p>Yoda says, <q>"For your support thank I must!"</q></p></body>
   </memo>

   <memo id="memo.3">
   <head><ref target="https://en.wikipedia.org/wiki/Obi-Wan_Kenobi"/></head>
   <to type="Jedi">Obi-Wan Kenobi</to>
   <from type="Sith">General Grievous</from>
   <body><p>I've been trained in your Jedi arts by Count Dooku!</p></body>
  </memo>
</memos>
```

# Assignment 1 - https://de.wikisource.org/wiki/Der_Struwwelpeter

Du bist moderne Pädagogin/moderner Pädagoge und findest was im Stuwwelpeter vermittelt wird so gar nicht zeitgemäß. Du hast dir überlegt eine digitale Edition des Stuwwelpeter zu erstellen, in der du bestimmte Textpassagen versuchst zu kommentieren. Auf https://de.wikipedia.org/wiki/Struwwelpeter findet man recht viel Info dazu. Versuche unter diesem Gesichtspunkt und dem möglichen Forschungsinteresse einer Pädagogin/eines Pädagogen folgende Aufgaben durchzuführen:

1. Erstelle ein **XML** Dokument in dem du “Zur hundertsten Auflage”, “Vorspruch”, “Struwwelpeter”, sowie 3 Geschichten deiner Wahl mit XML auszeichnest. Entwickle deine eigenen Kriterien dafür. Richte es aber so aus, dass du auch Kommentare an bestimmte Textpassagen knüpfen kannst. Die 6 Komponenten sollen in einem XML sein und überlege dir wie du die Metadaten zum Buch Stuwwelpeter modellierst.

2. Entwerfe ein **RELAX NG**, das genau deinem Markup aus erster Aufgabe entspricht.

3. Erstelle ein **XML/TEI** zu Zur hundertsten Auflage, Vorspruch, Struwwelpeter, sowie 3 Geschichten deiner Wahl (https://tei-c.org/guidelines/). Die 6 Komponenten sollen in einem XML/TEI sein und überlege dir wie du die Metadaten zum Buch Stuwwelpeter sinnvoll im teiHeader modellierst.

4. Lade auf Moodle ein ASS1_NACHNAME.zip hoch, das insgesamt 4 Files beinhaltet: zwei .xml (deine Auszeichnung, TEI Auszeichnung), ein .rng und ein Text/Word-File beinhaltet.

5. Schreibe eine **Reflexion** deiner Arbeit (max. 1 Seite)