

Digitale Edition – Vertiefung und Nutzung X-Technologien

Marcel Schaeben, Sebastian Zimmer



Programm

Mo **14:00 – 15:30 | XPath vertieft**
Einführung / Wiederholung XML-Dokumentstruktur / oXygen
XPath / Prädikate / Operatoren / Funktionen

16:00 – 17:30 | Reguläre Ausdrücke
Analyse und Manipulation von Zeichenketten

Di **09:00 – 10:30 | Basistechnologien für Auswertungsdaten I**
Grundlagen Transformation von XML-Dokumenten mit XSLT

11:00 – 12:30 | Basistechnologien für Auswertungsdaten II
XSLT vertieft / Erweiterte Sprachkonstrukte / Sortierung / Ausgabeformate

14:00 – 15:30 | Auswertungen
XSL-Transformation nach CSV / Datenaufbereitung zur Anzeige im DARIAH-Geobrowser

XPath vertieft – Übersicht

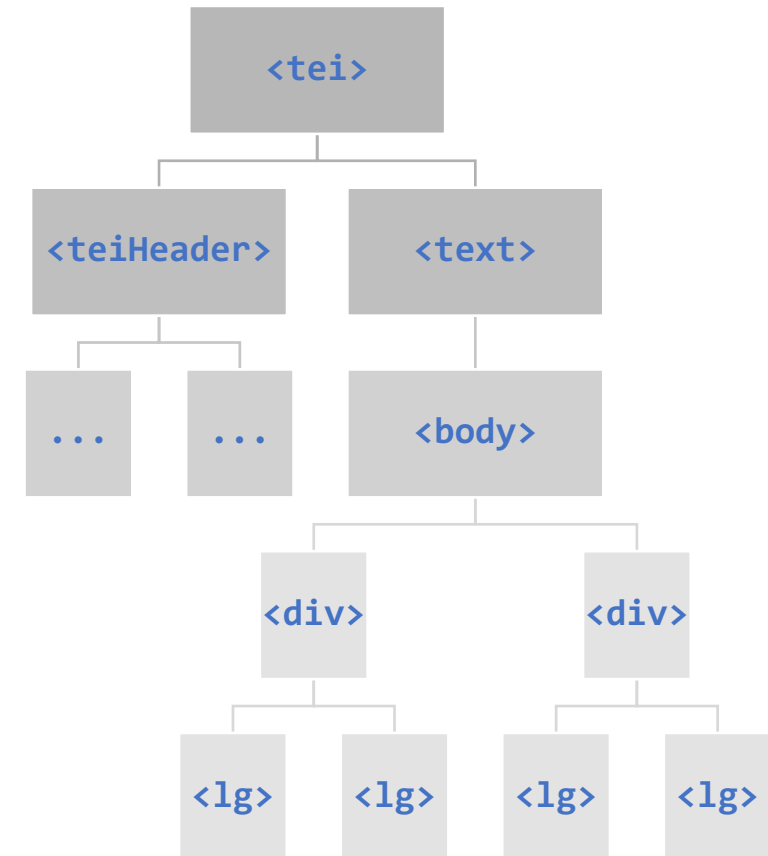
- Wiederholung: XML-Dokumentbaum
- XPath-Grundlagen
- XPath in oXygen
- Prädikate
- Operatoren
- Funktionen

XPath?

/TEI//div/lg[count(./l) > 2]

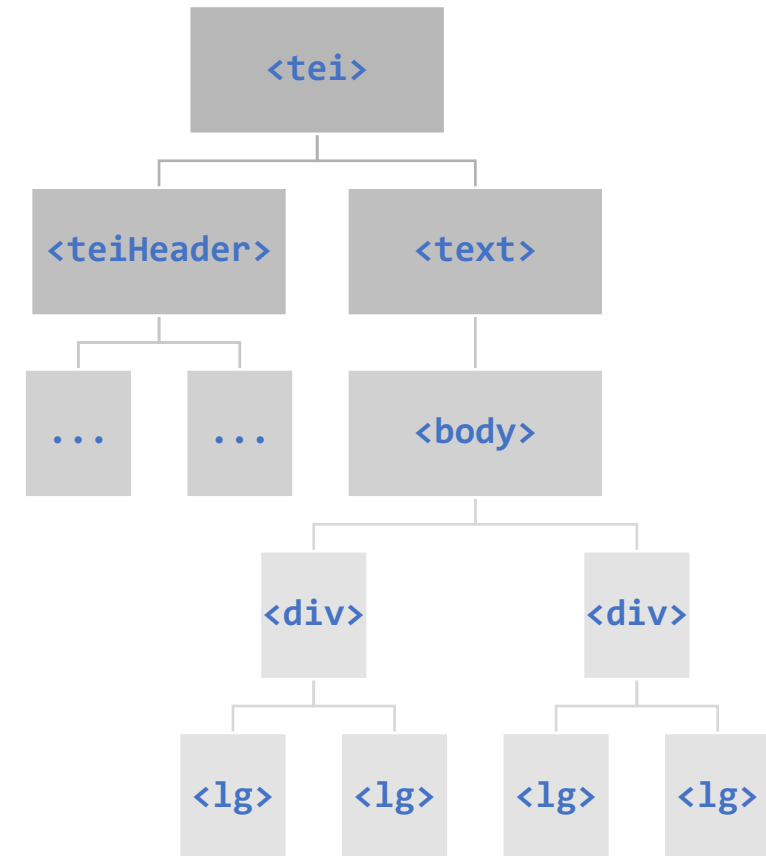
- „Xpath is a language for **addressing** parts of an XML document“ (W3C XPath Spezifikation) – <https://www.w3.org/TR/xpath-31/>
- Navigation in XML-Dokumenten
 - Vergleiche: Dateipfade – C:\Dokumente\Workshop\Folien.pptx
- Erzeugung von Rückgaben: Sequenzen (Mengen) von
 - Elementen
 - Zeichenketten (Strings)
 - Zahlen
 - Wahrheitswerten
- Fragen an das Dokument stellen

Wiederholung: XML-Dokumentstruktur



Wiederholung: XML-Dokumentstruktur

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          ...
        </lg>
        ...
      </div>
      <div type="couplet">
        <l n="13">So long as men can breathe or eyes can see,</l>
        <l n="14">So long lives this and this gives life to thee.</l>
      </div>
    </body>
  </text>
</TEI>
```



Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee.</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Dokumentknoten

„Unsichtbarer“ Knoten, der das gesamte Dokument umfasst

Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee.</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Wurzelknoten

Der erste (und einzige)
Elementknoten im Dokument

- Kind des Dokumentknotens
- Vorfahre aller weiteren
Elementknoten im Dokument

Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>..</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <ln="1">Shall I compare thee to a summer's day?</ln>
          <ln="2">Thou art more lovely and more temperate:</ln>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <ln="13">So long as men can breathe or eyes can see,</ln>
          <ln="14">So long lives this and this gives life to thee. </ln>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Elementknoten

<elementname>

...

</elementname>

Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee. </l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Attributknoten

attribut="wert"

Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee.</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Textknoten

Knotenarten

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
      <lg type="couplet">
        <l n="13">So long as men can breathe or eyes can see,</l>
        <l n="14">So long lives this and this gives life to thee.</l>
      </lg>
    </div>
  </body>
</text>
</TEI>
```

Kommentarknoten

<!-- Kommentar -->

Knotenarten

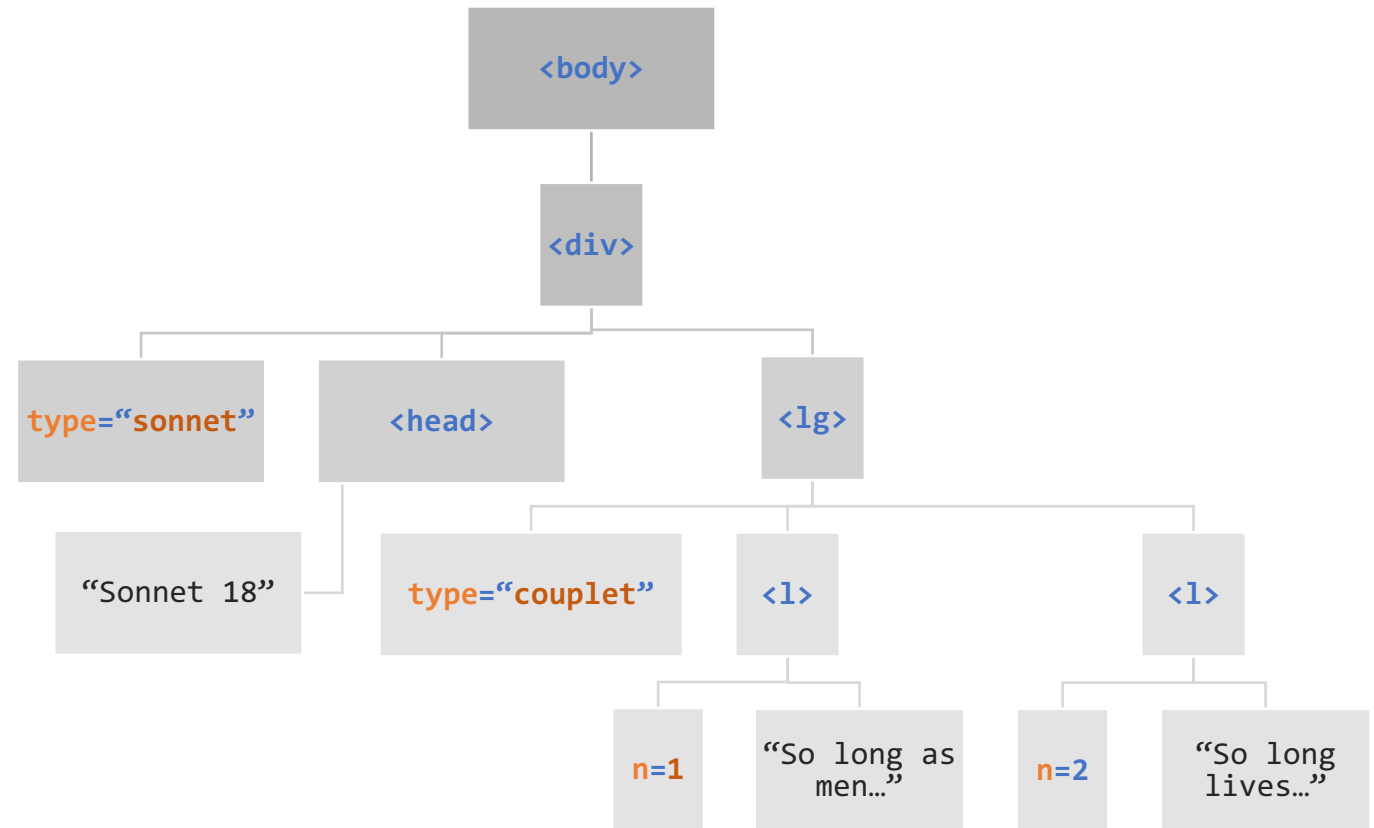
```
<?xml version="1.0" encoding="UTF-8"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <!-- TODO: hier fehlen Zeilen, bitte ergänzen -->
        </lg>
        ...
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee.</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

Verarbeitungsanweisungen

<? ... ?>

Element-, Attribut- und Textknoten

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="couplet">
          <l n="1">So long as men...</l>
          <l n="2">So long lives ...</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```



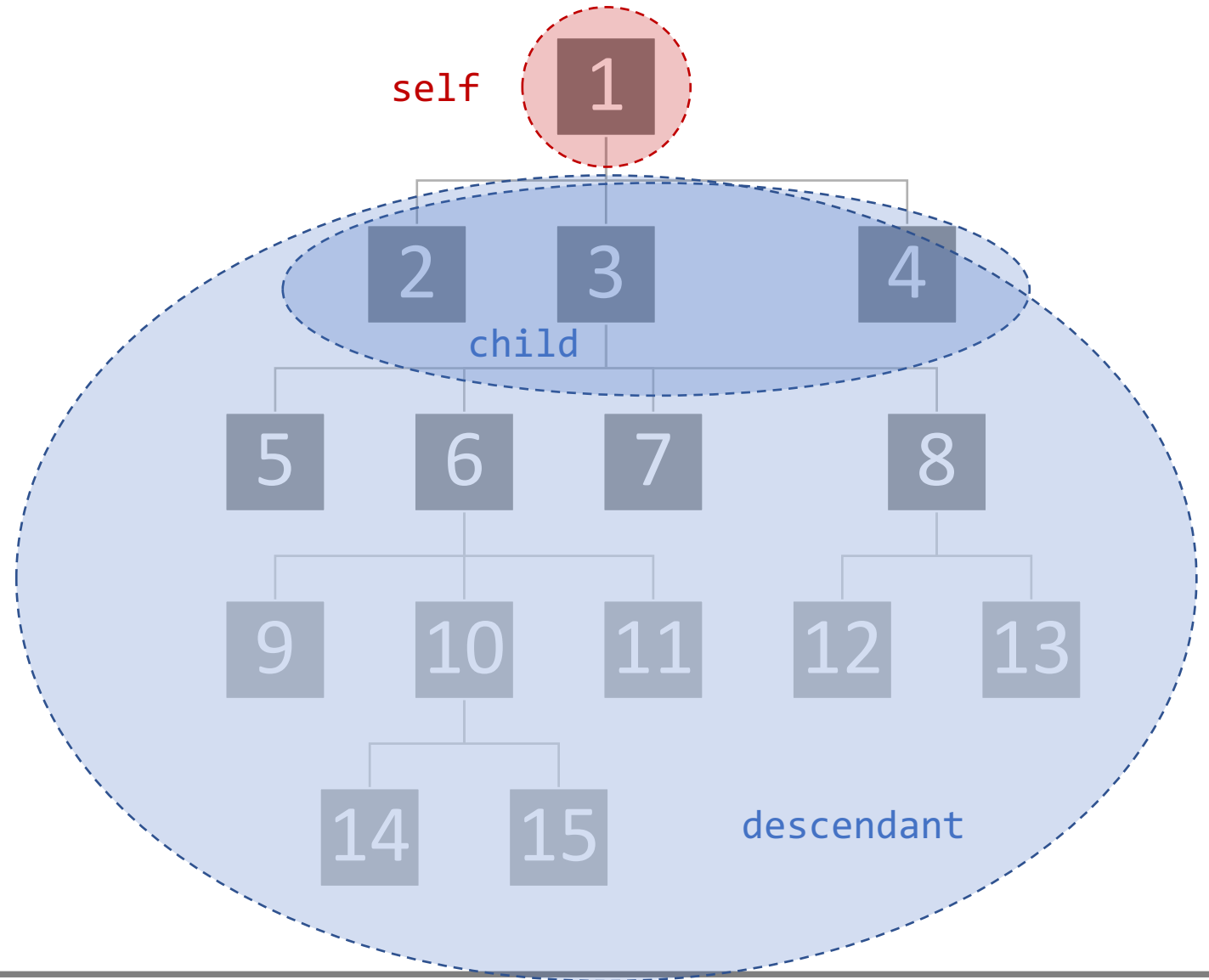
Bewegung im Baum

`/TEI/text//lg/l`

- Schrittweise Bewegung im Baum durch XPath-Ausdrücke
- Schritte im Pfad werden durch „/“ getrennt
- XML-Verarbeitung beginnt am Dokument-Knoten
- XPath-Schritte haben eine Richtung, die sog. „Achse“

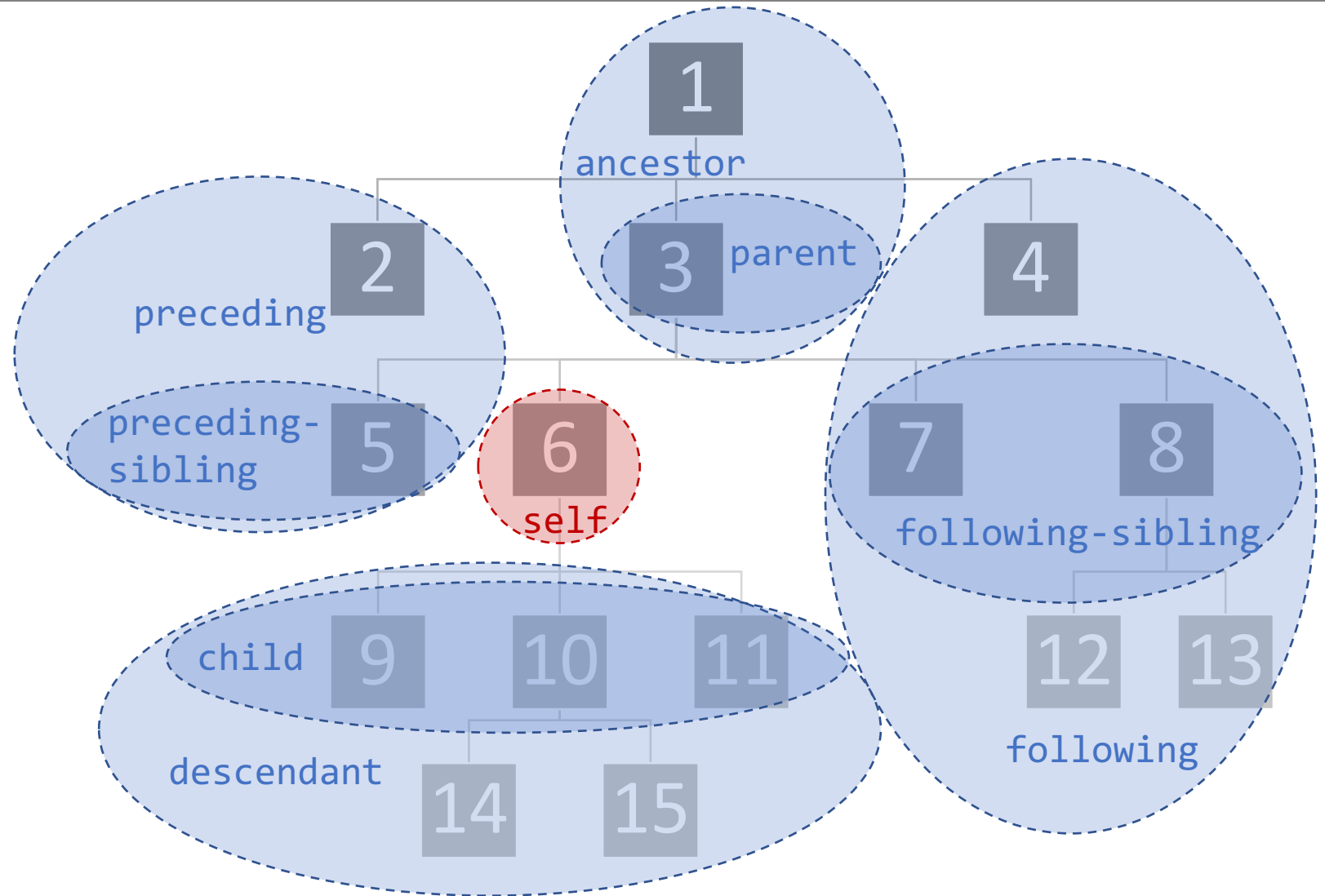
Achsen: „Familienverhältnisse“

- Selbst
self
- Eltern / Kind
parent / child
- Vorfahren / Nachfahren
ancestor / descendant



Achsen: „Familienverhältnisse“

- Selbst
self
- Eltern / Kind
parent / child
- Vorfahren /
Nachfahren
ancestor
descendant
- Geschwister
preceding-sibling
following-sibling



XPath-Syntax: Lokalisierungsschritte

- Aneinanderreihung von Schritten der Form

achse::knotentest[Prädikat]

getrennt durch „/“

- Beispiel

/child::TEI/child::text/descendant-or-self::l
/descendant-or-self::l/attribute::n

XPath-Achsen

self::

child::

descendant::

descendant-or-self::

preceding-sibling::

preceding::

following-sibling::

following::

parent::

ancestor::

attribute::

XPath-Syntax: Lokalisierungsschritte

„gehe von hier nach da“

/child::TEI/child::text/descendant-or-self::l

⇔ /TEI/text//l

/descendant-or-self::l/attribute::n

⇔ //l/@n

XPath-Achsen	Kurzform
self::	.
child::	/
descendant:: descendant-or-self::	//
preceding-sibling::	
preceding::	
following-sibling::	
following::	
parent::	..
ancestor::	
attribute::	@

XPath-Syntax: Knotentests

„bist du der, den ich suche?“

`/child::TEI/child::text/descendant-or-self::l`

$\Leftrightarrow$ `/TEI/text//l`

`/descendant-or-self::l/attribute::n`

$\Leftrightarrow$ `//l/@n`

Knotentest	Beschreibung
<code>xyz</code>	Elementknoten mit dem Namen „xyz“
<code>@xyz</code>	Attributknoten mit dem Namen „xyz“
<code>*</code>	Jeder Elementknoten
<code>@*</code>	Jeder Attributknoten
<code>node()</code>	Jeder Knoten beliebigen Typs
<code>text()</code>	Nur Textknoten
<code>comment()</code>	Nur Kommentarknoten: <!-- xyz -->
<code>processing-instruction()</code>	Verarbeitungsanweisungen <?name wert?>

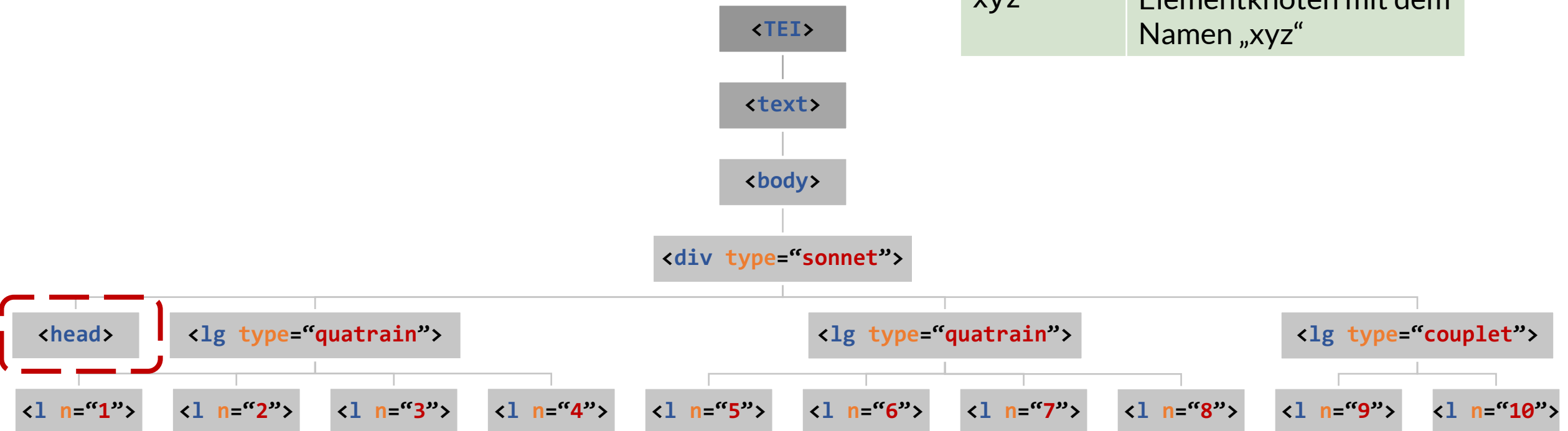
Beispiele: Einfache Ausdrücke

`/TEI/text/body/div/head`

`//body/div/head`

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“

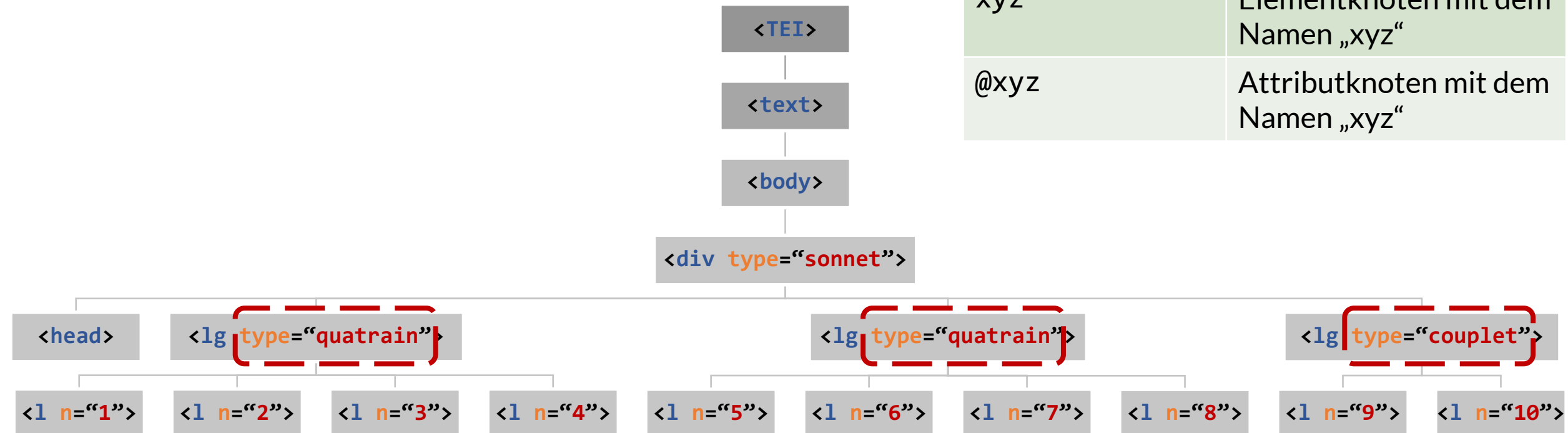


Beispiele: Einfache Ausdrücke

`//body/div/lg/@type`

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“



Übung: Einfache Ausdrücke

- Öffnen Sie die Datei *beispiel1.xml* in Oxygen
- Finden Sie alle *lg*-Elemente, die vom *div*-Element abstammen
- Finden Sie den *Parent* des *head*-Elements
- Finden Sie alle *l*-Elemente
- Probieren Sie weitere einfache XPath-Ausdrücke aus

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//
parent::	..

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“

XPath-Syntax: Prädikate

Bedingungen, welche die Kontextsequenz weiter einschränken

„finde alle vierzeiligen Verse“

→ „finde alle lg-Elemente vom typ ‚quatrain‘“

`/TEI/text//lg[@type="quatrain"]`

„finde alle Verse, die Zeilen enthalten“

→ „finde alle lg-Elemente, die l-Elemente enthalten“

`/TEI/text//lg[l]`

Operator	Beschreibung
=	Gleichheit
!=	Ungleichheit
gt oder >	Größer als
ge oder >=	Größer als oder gleich
lt oder <	Kleiner als
le oder <=	Kleiner als oder gleich
+ - * div	Addition, Subtraktion, Multiplikation, Division
and	Verknüpfung mit „UND“
or	Verknüpfung mit „ODER“

XPath-Syntax: Prädikate

Prädikate können kombiniert und verschachtelt werden

„finde alle lg-Elemente vom typ ‚quatrain‘, die l-Elemente enthalten“

```
/TEI/text//lg[@type="quatrain"]  
/TEI/text//lg[@type="quatrain" and l]
```

„finde alle div-Elemente, die mind. ein lg-Element vom typ ‚couplet‘ enthalten“

```
//div[lg[@type="couplet"]]
```

Operator	Beschreibung
=	Gleichheit
!=	Ungleichheit
gt oder > ge oder >=	Größer als Größer als oder gleich
lt oder < le oder <=	Kleiner als Kleiner als oder gleich
+ - * div	Addition, Subtraktion, Multiplikation, Division
and	Verknüpfung mit „UND“
or	Verknüpfung mit „ODER“

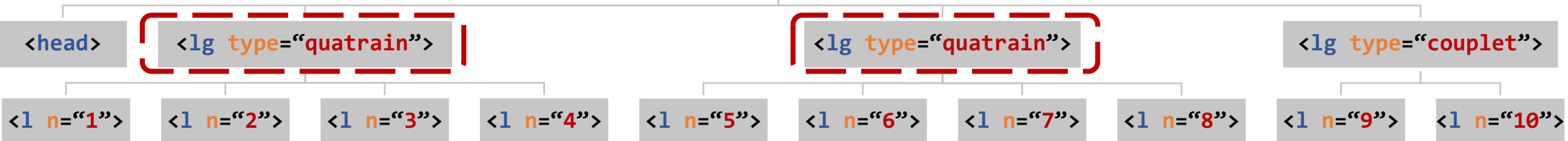
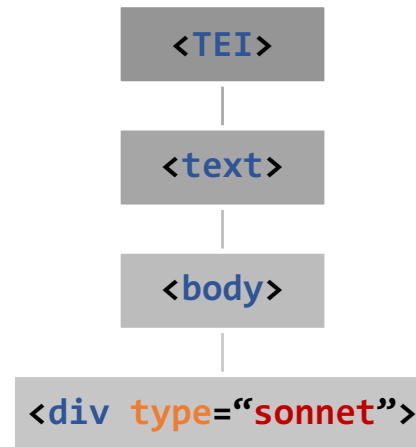
Beispiele: Ausdrücke mit Prädikaten

`//body/div/lg[@type="quatrain"]`

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

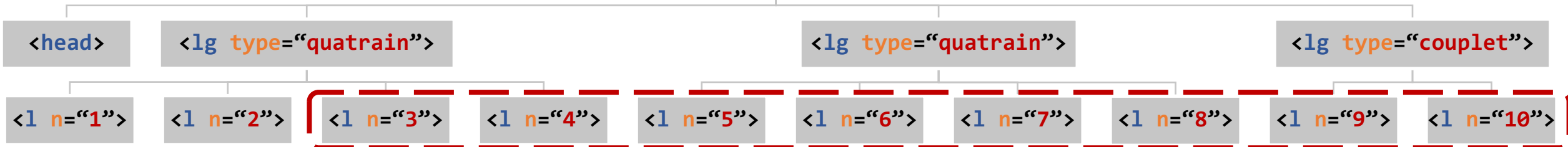
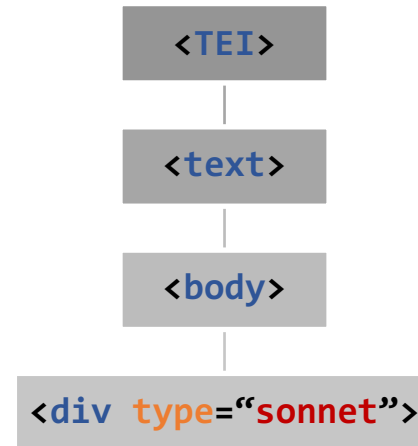
Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“

Operator	Beschreibung
=	Gleichheit



Beispiele: Ausdrücke mit Prädikaten

`//body/div/lg/l[@n > 2]`



XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“

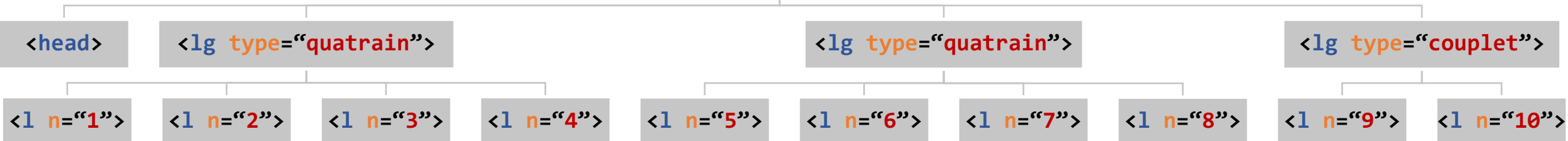
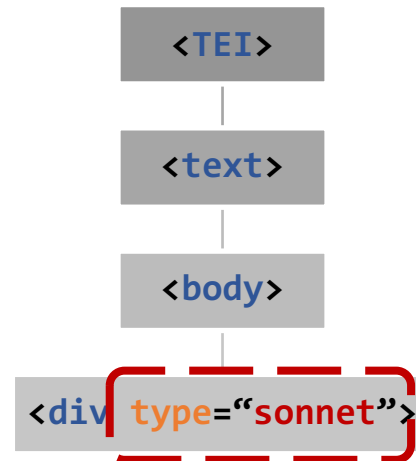
Operator	Beschreibung
>	Größer als

Beispiele: Ausdrücke mit Prädikaten

```
//body/div//lg[@type="quatrain"]/../@type
```

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//
parent::	..

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“

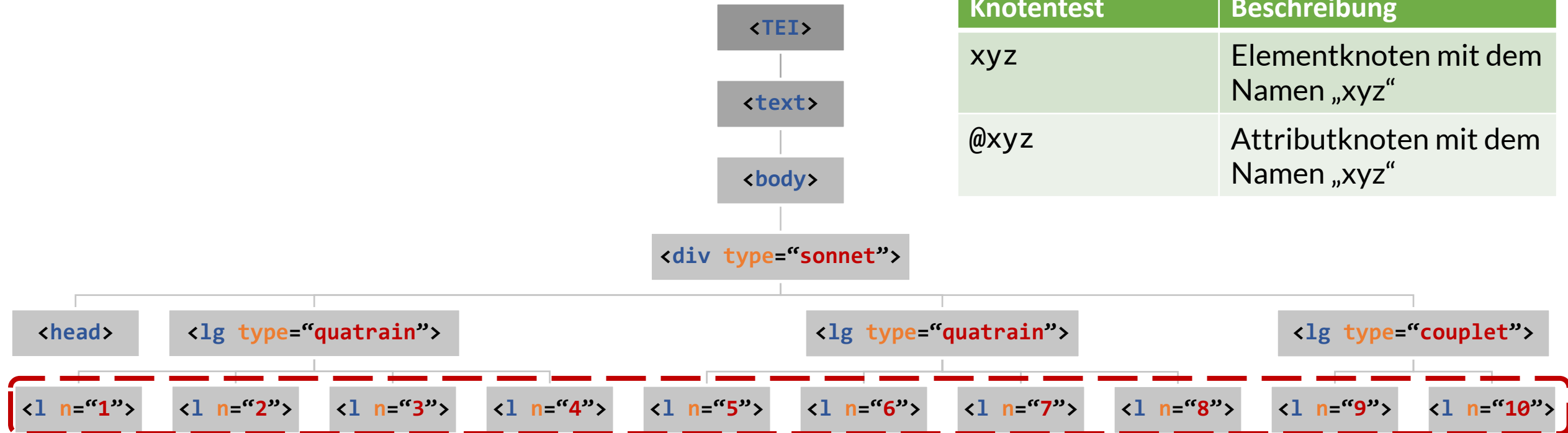


Beispiele: Ausdrücke mit Prädikaten

```
//body/div//l[ancestor::div/@type="sonnet"]
```

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//
ancestor::	

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“



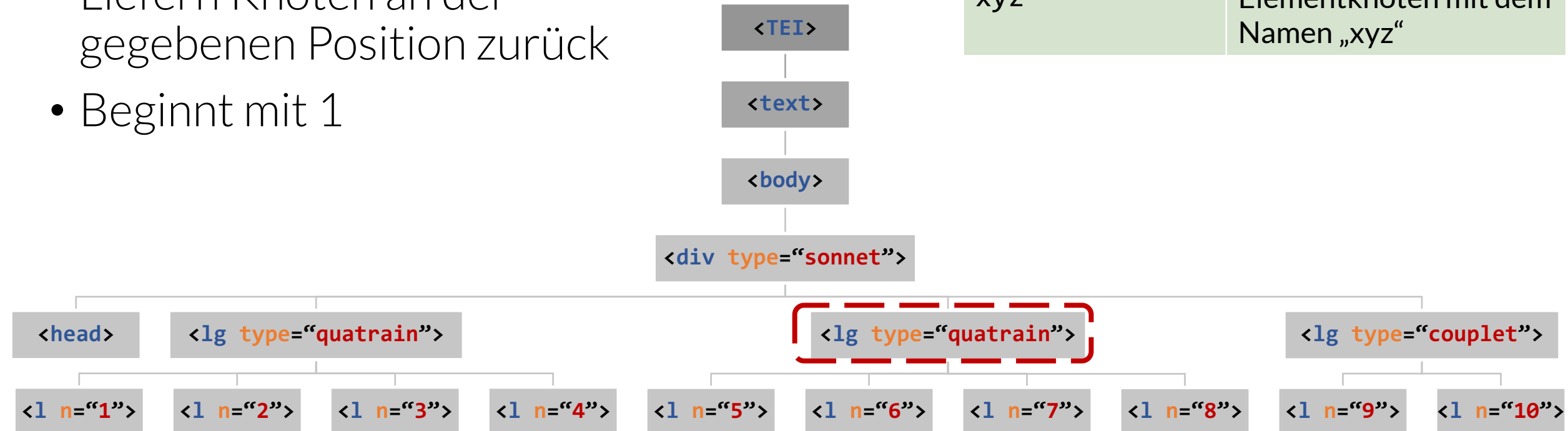
Beispiele: Ausdrücke mit Prädikaten

`//body/div/lg[2]`

- Sonderfall: Indizes
- Liefern Knoten an der gegebenen Position zurück
- Beginnt mit 1

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“

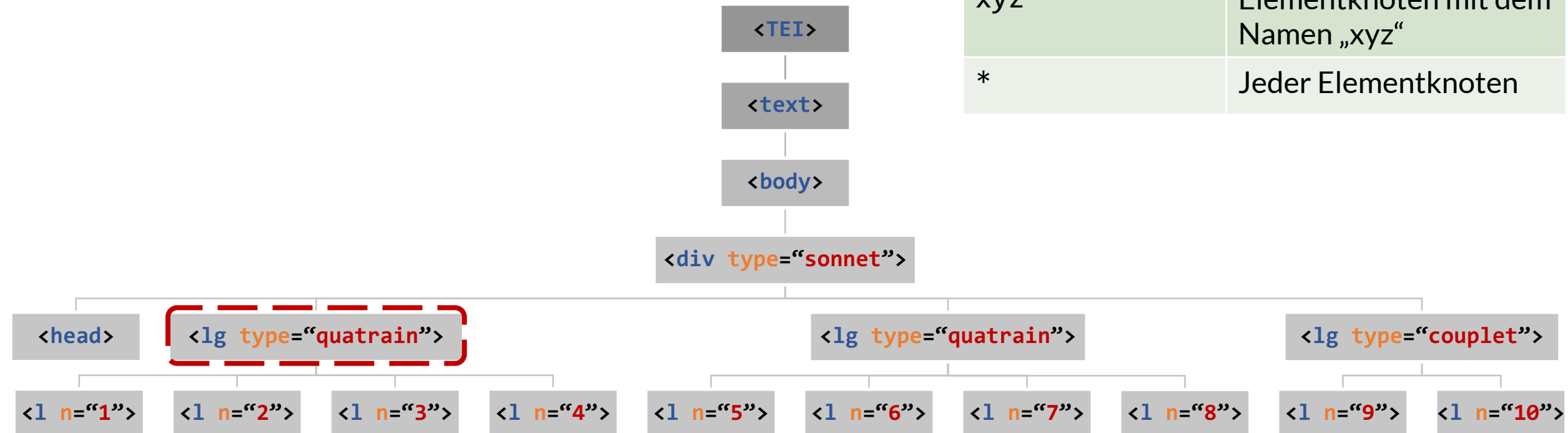


Beispiele: Ausdrücke mit Prädikaten

`//body/div/*[2]`

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
*	Jeder Elementknoten



Übung: Ausdrücke mit Prädikaten

- Öffnen Sie die Datei *beispiel1.xml* in Oxygen

- Finden Sie alle Zeilen aus Vierzeiler-Versen

`//lg[@type="quatrain"]/l`

- Finden Sie alle Zeilen mit den Nummern 1-6

`/l[@n <= 6]`

- Finden Sie die dritte Zeile des dritten Verses

`//lg[3]/l[3]`

XPath-Achsen	Kurzform
child::	/
descendant-or-self::	//

Knotentest	Beschreibung
xyz	Elementknoten mit dem Namen „xyz“
@xyz	Attributknoten mit dem Namen „xyz“

Operator	Beschreibung
=	Gleichheit
gt oder > ge oder >=	Größer als Größer als oder gleich

XPath-Syntax: Funktionen

`funktionsname(argument1, argument2)`

- Anstelle des Knotentests oder als Teil des Prädikats
- Argumente können Knoten(sequenzen) oder Werte sein
- Verlangen unterschiedliche Arten und Anzahlen von Argumenten
- Funktionen „tun“ verschiedenes und liefern unterschiedliche Ergebnisse zurück

```
//p/count()
```

```
contains(//titleStmt/title, "Brief")
```

```
contains("Briefpapier", "Brief")
```

XPath: Rückgabearten

Sequenzen aus ein oder mehreren:

- Knoten (Elemente, Attribute, Textknoten...)
`//lg` → (`<lg>`, `<lg>`, `<lg>`, `<lg>`)
- Zeichenketten (Strings)
`persName/concat(forename, " ", surname)` → („Horst Müller“)
- Zahlen
`count(//lg)` → (4)
- Wahrheitswerte (boolean: wahr / falsch)
`contains(“Schnecke“, “ecke“)` → (wahr)

```
<persName>  
  <forename>Horst</forename>  
  <surname>Müller</surname>  
</persName>
```

Sequenzen sind **geordnet** (haben eine bestimmte Reihenfolge) und können **Duplikate** enthalten.

Sequenzen können **leer** sein (Leere Sequenz: `()`)

Einige Funktionen I

- **count(sequence)**
zählt etwas, erwartet eine Sequenz von Knoten oder Werten, liefert eine Zahl zurück
`count(//p) → 5`
- **string-length(string)**
erwartet einen String, liefert seine Länge als Zahl zurück
`string-length("Rindfleischetiketierungsüberwachungs
aufgabenübertragungsgesetz") → 63`
- **starts-with(string, string)**
prüft, ob ein String mit einem anderen String beginnt; liefert einen Wahrheitswert zurück
`starts-with("Beispiel", "Bei") → wahr`
`starts-with(„Beispiel“, „Neben“) → falsch`

Einige Funktionen II

- **normalize-space(string)**

Entfernt überflüssigen Whitespace aus einem String (Leerzeichen, Zeilenumbrüche, Tabulatoren...), liefert einen String zurück

```
normalize-space("  Octavie      du Rey  
                  de Meynières  ")
```

→ "Octavie du Rey de Meynières"

- **lower-case(string)**

Wandelt einen String in Kleinbuchstaben um

```
lower-case("BeIsPiEl") → „beispiel“
```

- **not(boolean)**

Dreht den Wahrheitswert um

```
not(contains("beispiel", "ö")) → wahr
```

Einige Funktionen III

- **distinct-values(sequence of values)**

Filtert Duplikate aus einer Sequenz, liefert eine Sequenz zurück

`distinct-values(("a", "b", "a", "c")) → ("a", "b", "c")`

- **string-join(sequence of strings, string)**

Verkettet eine Sequenz von Strings, fügt einen String dazwischen ein, liefert einen String zurück

`string-join(("a", "b", "c"), ";") -> „a;b;c“`

- **concat(string, string, string...)**

Verkettet eine beliebige Anzahl an Strings

`concat("a", "b", "c") -> „abc“`

- **max(sequence of numbers)**

ermittelt die höchste Zahl aus einer Sequenz von Zahlen, liefert eine Zahl zurück

`max((1923, 1823, 1723, 2009, 1345)) => 2009`

Beispiele: Ausdrücke mit Funktionen

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>...</teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <l n="3">Rough winds do shake the darling buds of May,</l>
          <l n="4">And summer's lease hath all too short a date:</l>
        </lg>
        <lg type="couplet">
          <l n="13">So long as men can breathe or eyes can see,</l>
          <l n="14">So long lives this and this gives life to thee.</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

```
count(//1)
  → eine Zahl
  → 6

//lg[count(1) = 4]
  → eine Sequenz von lg-Elementen
  → <lg type="quatrain">

//1[contains(lower-case(.), "and")]
  → eine Sequenz von l-Elementen
  → (<l n=2>,<l n=4>,<l n=14>)

//lg[@type="couplet"]/string-join(1, " ")
  → ein String
  → „So long as men can breathe or eyes can
    see, So long lives this and this gives life to
    thee.“

//lg[1]/1[last()]
  → ein Element
  → (<l n=14>)
```

Übung: Ausdrücke mit Funktionen

In beispiel1.xml:

- Zählen Sie alle Verszeilen im Dokument
`count(//1)`
- Finden Sie alle Zeilen, die das Wort „and“ bzw „And“ enthalten
`//1[contains(., "and") or contains(., "And")]`
`//1[contains(lower-case(.), "and")]`
- Verketteten Sie den Text jeder Zeile im Dokument mit einem Schrägstrich (/)
`string-join(//1, " / ")`
- Finden Sie die letzte Zeile jedes Verses
`//1[last()]`

Funktion	Rückgabe
<code>count(sequence)</code>	Zahl: Anzahl der Elemente in der Sequenz
<code>contains(a, b)</code>	wahr/falsch: ist a in b enthalten
<code>string-join(sequence, string)</code>	String: verkette alle Elemente mit einem gegebenen String dazwischen
<code>last()</code>	Zahl: Position des letzten Elements der Sequenz

Weitere XPath-Sprachelemente

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>Sonnets</title>
        <author>William Shakespeare</author>
      </titleStmt>
    </fileDesc>
  </teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare thee to a summer's day?</l>
          <l n="2">Thou art more lovely and more temperate:</l>
          <l n="3">Rough winds do shake the darling buds of May,</l>
          <l n="4">And summer's lease hath all too short a date:</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

- Vereinigungsoperator: |

*Vereinigt zwei Knotensequenzen in
Dokumentreihenfolge*

`//titleStmt/title | //body//head`

`//body//head | //titleStmt/title`

→ (`<title>Sonnets</title>`, `<head>Sonnet 18</head>`)

Weitere XPath-Sprachelemente

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>Sonnets</title>
        <author>William Shakespeare</author>
      </titleStmt>
    </fileDesc>
  </teiHeader>
  <text>
    <body>
      <div type="sonnet">
        <head>Sonnet 18</head>
        <lg type="quatrain">
          <l n="1">Shall I compare...</l>
          <l n="2">Thou art more lovely...</l>
          <l n="3">Rough winds do shake...</l>
          <l n="4">And summer's lease...</l>
        </lg>
      </div>
    </body>
  </text>
</TEI>
```

- Sequenzkonstruktor: (... , ... , ...)

*Konstruiert Knoten-/Wertsequenzen in
angegebener Reihenfolge*

(//titleStmt/title, //body//head)

(//body//head, //titleStmt/title)

(//body//head, //titleStmt/title)[1]

//div[@type="sonnet"]/(head, lg/l[4])

→ (

 <head>Sonnet 18</head> ,

 <l n="4">...</l>

)

string-join(("a", "b", "c"), ",")

→ „a, b, c“

Übungen

- Öffnen Sie den Korpus der Briefedition der Berliner Intellektuellen (*teiCorpus.xml*)
- Suchen Sie alle deutschsprachigen Brieffitel
Tipp: `[lang("de")]` als Kurzform von `[@xml:lang="de"]`
- (1) Suchen Sie alle an der Korrespondenz beteiligten Personen(-namen),
(2) entfernen Sie Duplikate
Tipp: `correspDesc`
- **Experimentieren Sie!** (Ideen: alle erwähnten Orte, alle vorkommenden Gedichte [lg-Elemente], alle unterstrichenen Wörter)

Reguläre Ausdrücke in XPath

...sind nützlich zur

- **Analyse** von Text, z. B. *Enthält ein Textknoten eine Datumsangabe?*
`matches(text(), "\d{1,2}\.\d{1,2}\d{4,4}")`
- **Manipulation** z.B. *normalisiere Datumsangabe 30.09.2018 zu „2018-09-30“*
`matches(text(), "\d{1,2}\.\d{1,2}\d{4,4}")`
`replace("30.09.2018",
 "(\d{1,2})\.(\\d{1,2})\\.(\\d{4,4})",
 "$3-$2-$1")`

Reguläre Ausdrücke in XPath

- **Tokenisierung** – Aufspaltung eines Strings anhand eines Trenners

```
tokenize("Horst Müller", "\s+")  
→("Horst", "Müller")
```

```
reverse(tokenize("Müller, Horst ", ", \s"))  
→("Horst", "Müller")
```

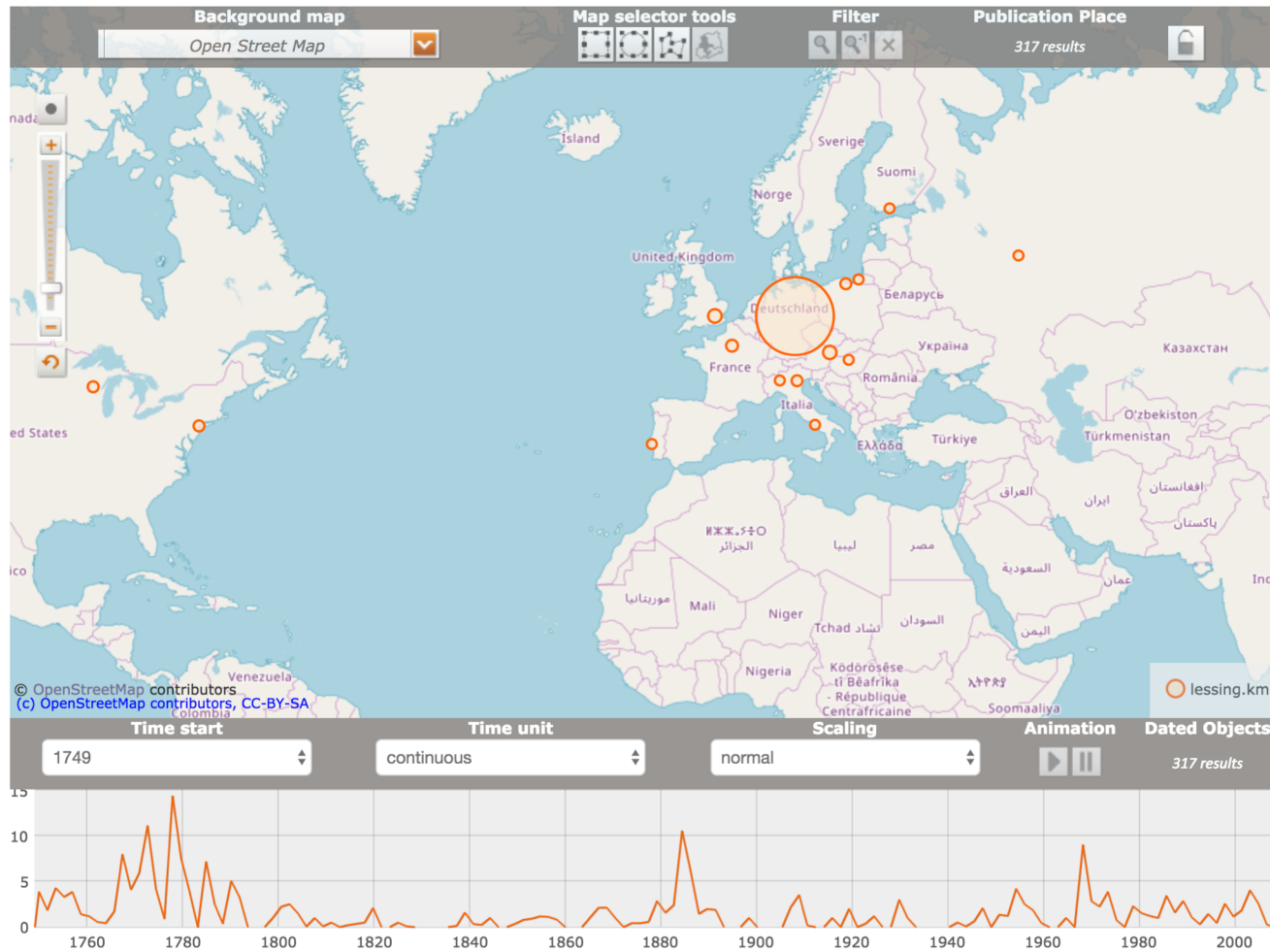
Reguläre Ausdrücke in XPath: Übung

In *teiCorpus.xml*:

- Suchen Sie alle Absätze, die Datumsangaben der Form TT. Monat YYYY („23. März 1080“) enthalten
 - Tipp: \w → alle Wortzeichen

```
//p[matches(., "\d{1,2}\. \w+ \d{1,4}")]
```

DARIAH-Geobrowser



Laden Sie folgende Dateien aus dem Google Drive herunter (Ordner: geobrowser)

- geodaten2csv_basis.xml
- Ortsindex.xml
- Universal-Kirchenzeitung - Nachrichten aus der Israelitschen Abteilung.csv

<https://geobrowser.de.dariah.eu/>

Aufgabe für diese Sitzung

- Die Ortsangaben mit Geokoordinaten aus dem Berlin-Corpus im Geobrowser anzeigen lassen
- Dafür müssen wir:
 - Verstehen, welche Art von Daten der Geobrowser benötigt
 - Verstehen, wie die Daten im Ortsindex des Corpus vorliegen
 - Ein XSL-Stylesheet erstellen, welches die Daten in das benötigte Format transformiert

CSV

- Der Geobrowser möchte **CSV-Daten** in einem bestimmten Format haben
- **CSV: Comma Separated Values** (Klartext-Tabellenformat)

Nachname, Vorname, Telefonnummer

Fischer, Franz, 0221 470 4056

Sahle, Patrick, 0221 470 3894

Dängeli, Peter, 0221 470 4055

CSV: String-Qualifier

- Problem: Was ist, wenn der Inhalt eines Feldes das Trennzeichen (,) enthält?

Nachname, Vorname, Telefonnummer, Adresse

Fischer, Franz, 0221 470 4056, Universitätsstraße 22, 50931 Köln

Sahle, Patrick, 0221 470 3894, Meister-Eckehart-Straße 22, 50931 Köln

Dängeli, Peter, 0221 470 4055, Universitätsstraße 22, 50931 Köln

- Lösung: Einpacken der Werte in Redezeichen

"Nachname", "Vorname", "Telefonnummer", "Adresse"

"Fischer", "Franz", "0221 470 4056", "Universitätsstraße 22, 50931 Köln"

"Sahle", "Patrick", "0221 470 3894", "Meister-Eckehart-Straße 22, 50931 Köln"

"Dängeli", "Peter", "0221 470 4055", "Universitätsstraße 22", "50931 Köln"

CSV: Escaping

- Problem: Was ist, wenn der Inhalt eines Feldes Redezeichen enthält?

```
"Nachname", "Vorname", "Telefonnummer", "Adresse"  
"Fischer", "Franz "Fröhlich"", "0221 470 4056", "Universitätsstraße 22, 50931 Köln"  
"Sahle", "Patrick", "0221 470 3894", "Meister-Eckehart-Straße 22, 50931 Köln"  
"Dängeli", "Peter", "0221 470 4055", "Universitätsstraße 22", "50931 Köln"
```

- Lösung: „Escaping“ durch doppelte Redezeichen

```
"Nachname", "Vorname", "Telefonnummer", "Adresse"  
"Fischer", "Franz ""Fröhlich""", "0221 470 4056", "Universitätsstraße 22, 50931 Köln"  
"Sahle", "Patrick", "0221 470 3894", "Meister-Eckehart-Straße 22, 50931 Köln"  
"Dängeli", "Peter", "0221 470 4055", "Universitätsstraße 22", "50931 Köln"
```

Geobrowser-CSV: Beispiel

```
"Name","Address","Description","Longitude","Latitude","TimeStamp","TimeSpan:begin","TimeSpan:end","GettyID",""  
"Altona","Altona","Steinheim's Portrait","10.4167","51.8","1837-06-29","","","7005628",""  
"Alzey","Alzey","Einweihung einer Thorarolle","8.1167","49.75","1837-02-16","","","7005992",""  
"Augsburg","Augsburg","Corresp.-Ber., die relig. Angelegenheiten der Israeliten betr.","10.9","48.35","1837-04-  
13","","","7004324",""  
"Berlin","Berlin","Erbauung zweier christl. Kirchen von israel. Freigebigkeit","13.4167","52.5333","1837-02-  
23","","","7003712",""
```

Ortsindex → Geobrowser-CSV: Vorbereitung

- Schritt 1: Wie sieht unser Zielformat aus? Was brauchen wir?

→ Eine CSV-Datei mit den Spalten Name, Address, Description, Longitude, Latitude

(für unsere Zwecke: Name = Address, Description bleibt leer)

- Schritt 2: Wie sehen unsere Daten aus? Was haben wir?

Das sehen wir uns mal genauer an... → Ortsindex.xml

(müssen wir Strings in Redezeichen einpacken? Sind die Geokoordinaten einheitlich formatiert?)

Ortsindex → Geobrowser-CSV

Nützliche Funktionen	
<code>string-join(sequence, string)</code>	verkette alle Elemente mit einem gegebenen String dazwischen
<code>tokenize(string, regex)</code>	Spaltet einen String in mehrere auf
<code>matches(string, regex)</code>	Überprüft, ob ein regulärer Ausdruck auf einen String passt
<code>reverse(sequence)</code> <code>reverse(("a", "b", "c")) → (("c", "b", "a"))</code>	Dreht eine Sequenz um

Ziel: eine Kopfzeile mit den Spaltennamen

+ für jeden Ort eine Zeile mit 2 x Ortsnamen sowie Longitude und Latitude

"Name", "Address", "Description", "Longitude", "Latitude"

"Dresden", "Dresden", "", "13.73836", "51.049259"

"Leipzig", "Leipzig", "", "12.37475", "51.340333"

...

Ortsindex → Geobrowser-CSV

- Öffnen Sie das Grundgerüst für die XSL-Transformation:
geodaten2csv_basis.xsl
- Schritt 1: Generieren Sie die Kopfzeile der CSV-Datei (+
Zeilenumbruch)

"Name", "Address", "Description", "Longitude", "Latitude"

```
<xsl:template match="/">  
  <xsl:text>"Name","Address","Description","Longitude","Latitude"</xsl:text>  
  <xsl:value-of select="$newline" />  
</xsl:template>
```

Ortsindex → Geobrowser-CSV

- Schritt 2: Generieren Sie für jedes Place-Element eine Zeile, die den (deutschsprachigen) Namen des Ortes enthält

```
<xsl:for-each select="//tei:place">  
  <xsl:value-of select="(/tei:placeName[lang('de')], ./tei:placeName)[1]" />  
  <xsl:value-of select="$newline" />  
</xsl:for-each>
```

- Schritt 3: Formatieren Sie die Zeilen richtig – Ortsnamen zweimal hintereinander, mit Redezeichen umschlossen, mit Komma getrennt

"Dresden", "Dresden"

"Leipzig", "Leipzig"

```
<xsl:for-each select="//tei:place">
```

```
  <xsl:text>&quot;</xsl:text>
```

```
  <xsl:value-of select="(/tei:placeName[lang('de')], ./tei:placeName)[1]" />
```

```
  <xsl:value-of select="$separator" />
```

```
  <xsl:value-of select="(/tei:placeName[lang('de')], ./tei:placeName)[1]" />
```

```
  <xsl:text>&quot;</xsl:text>
```

```
  <xsl:value-of select="$newline" />
```

```
</xsl:for-each>
```


- Schritt 4: Als dritte CSV-Spalte benötigen wir einen leeren String („description“) – fügen Sie an der Richtigen Stelle zwei CSV-Trenner hintereinander ein

```
<xsl:for-each select="//tei:place[matches(tei:location/tei:geo, '-?\d+\.\?\d*°?,? -?\d\.\?\d*')]">
  <xsl:text>&quot;</xsl:text>
  <xsl:value-of select="(./tei:placeName[lang('de')], ./tei:placeName)[1]" />
  <xsl:value-of select="$separator" />
  <xsl:value-of select="(./tei:placeName[lang('de')], ./tei:placeName)[1]" />

  <xsl:value-of select="$separator" />
  <xsl:value-of select="$separator" />

  <xsl:text>&quot;</xsl:text>
</xsl:for-each>
```

- Schritt 5: Schränken Sie die Auswahl der Place-Elemente auf solche ein, die gültige Koordinaten haben (deren `tei:location/tei:geo`-Elemente auf den zuvor ermittelten Regulären Ausdruck passen)

`-?\d+\.\d*°?,-?\d+\.\d*`

Tipp: Prädikat mit `matches()`-Funktion

```
<xsl:for-each select="//tei:place[matches(tei:location/tei:geo, '-?\d+\.\d*°?,-?\d+\.\d*')]">
```

- Schritt 6: Schreiben Sie ein Template für `tei:geo`-Elemente, welches die Koordinaten in unser Zielformat bringt, indem es
 - Die Koordinaten in zwei Strings **aufspaltet** (Latitude und Longitude) denken Sie an die möglichen Trennzeichen und unseren Trenner-Regex:
`°? , ?\s`
 - Diese **umdreht** (der CSV Browser möchte erst Longitude, dann Latitude!)
→ `reverse(sequence)`
 - Und diese dann wieder mit unserem CSV-Trenner **zusammenfügt**!

“50.938056 6.956944” → 6.956944“; ,";50.938056

```
<xsl:template match="tei:geo">  
  <xsl:value-of select="string-join(reverse(tokenize(., '°?,?\s')), $separator)" />  
</xsl:template>
```

- Schritt 7: Wenden Sie das Koordinaten-Template an der richtigen Stelle an

```
<xsl:for-each select="//tei:place[matches(tei:location/tei:geo, '-?\d+\.\?\d*°?,? -?\d\.\?\d*')]">
  <xsl:text>&quot;;</xsl:text>
  <xsl:value-of select="(./tei:placeName[lang('de')], ./tei:placeName)[1]" />
  <xsl:value-of select="$separator" />
  <xsl:value-of select="(./tei:placeName[lang('de')], ./tei:placeName)[1]" />

  <xsl:value-of select="$separator" />
  <xsl:value-of select="$separator" />

  <xsl:apply-templates select="./tei:location/tei:geo"></xsl:apply-templates>

  <xsl:text>&quot;;</xsl:text>
  <xsl:value-of select="$newline" />
</xsl:for-each>
```

Übersichten und Referenzen

Übersichten

<https://www.data2type.de/xml-xslt-xslfo/xpath/referenz/>

https://www.w3schools.com/xml/xsl_functions.asp

https://www.w3schools.com/xml/xpath_intro.asp

Referenzen

<https://www.w3.org/TR/xpath-31/>

<https://www.w3.org/TR/xpath-functions-31/>

Kontakt

Fragen? Gerne melden!



Marcel Schaeben

m.schaeben@uni-koeln.de

Sebastian Zimmer

sebastian.zimmer@uni-koeln.de

Cologne Center for eHumanities (CCEH) | cceh.uni-koeln.de

Mit Dank für Material und Inspiration an:

James Cummings, Ulrike Henny, Patrick Sahle, Daniel Schopper