# FHNW: Vorlesung XML Sommersemester 2006

## Test XSLT

## MUSTERLÖSUNG (Lösungen in grün & underline)

## 10.7.2006, $10^{30}$ – $11^{30}$

Bitte schreiben Sie *sorgfältig* und *gut lesbar*! Danke!
Für diesen Test dürfen Sie *keinerlei Unterlagen* verwenden, insbesondere auch keine elektronischen Hilfsmittel wie Laptop oder Organizer!

1. Nehmen Sie an, Sie haben einen Dokumententypen, der auf *verschiedenen Ebenen* `chapter` Elemente zulässt (d.h. möglicherweise auch *ineinander geschachtelt*), und irgendwo innerhalb der `chapter` (*in beliebiger Tiefe*, also nicht auf einer festgelegten Ebene unterhalb der `chapter` Elemente!) können dann `img` Elemente auftreten. Nun sollen Sie für ein beliebiges `img` Element im Dokument seine Sequenznummer im jeweiligen `chapter` bestimmen, also feststellen, das *wievielte `img` Element im aktuellen `chapter`* es ist. Beschreiben Sie kurz Ihren Ansatz (wie Sie das Problem angehen) und schreiben Sie einen *entsprechenden XPath* auf.
   Die einfachste Lösung ist, die `img` Elemente zu zählen, die das gleiche `chapter` Elternelement haben und vor dem momentanen `img` liegen. Um es etwas übersichtlicher zu machen, ist die Lösung hier mit zwei XPaths dargestellt, das Resultat des zweiten XPaths ist die gesuchte Lösung (unter Verwendung der `current()` Funktion liesse sich die Lösung natürlich auch in einem XPath aufschreiben):
   ```
   chapter-parent-id := generate-id(ancestor::chap[1])
   count(preceding::img[generate-id(ancestor::chap[1])
                                = $chapter-parent-id]) + 1
   ```
2. In sehen in einem XSLT 1.0 Stylesheet den XPath `$var//name[1]`. Beschreiben Sie, *unter welchen Umständen* dieser Ausdruck korrekt ist (d.h. ausgewertet wird und nicht zu einem Laufzeitfehler führt), und unter welchen Umständen er es nicht ist.
   Es gibt am einfachsten drei Fälle zu unterscheiden:
   a. Ist `var` *weder als Variable noch als Parameter definiert*, so ist der XPath sicher nicht korrekt.
   b. Ist `var` eine Variable oder ein Parameter, der keinen Node Tree enthält, oder der ein Node Tree ist, aber sich auf einen *Temporary Result Tree* bezieht (also auf ein innerhalb des XSLT erzeugten XPath Node Tree), so ist der XPath ebenfalls nicht korrekt.
   c. Nur wenn sich `var` auf das *Eingabedokument* bezieht (also Knoten aus diesem enthält) oder ein Parameter ist, der als *Übergabewert einen XPath Node Tree* erhalten hat, ist der XPath korrekt.

3. Die XPath Node Tests in XPath können nur Elemente mit einem fixen Namen oder Elemente mit beliebigem Namen selektieren. Sie haben die Aufgabe, alle die Elemente in einem Dokument zu selektieren, deren Name mit einem "L" beginnt. Wie gehen Sie vor? Schreiben Sie den entsprechenden XPath auf und erklären Sie ihn.
   Dieses Problem lässt sich am einfachsten über Prädikate lösen. Man selektiert also alle Elemente und sortiert dann genau die heraus, die die gestellte Bedingung erfüllen: `//*[starts-with(local-name()), "L"]`
4. Sie erhalten eine *Tabelle in HTML* und sollten diese *transponieren*, also die Spalten in Zeilen überführen und die Zeilen in Spalten. Aus einer 6x10 Tabelle sollen Sie also z.B. eine 10x6 Tabelle erzeugen. Die Tabelle ist in der üblichen `table/tr/td` Struktur und Sie können davon ausgehen, dass die Tabelle *komplett ausgefüllt und regelmässig* ist, also keine Spezialitäten wie `colspan` oder `rowspan` enthält. Schreiben Sie XSLT Code (die Syntax ist nicht wichtig, aber Ihre Notation muss ohne Probleme als XSLT interpretierbar sein!), der diese Transposition implementiert. Kommentieren Sie Ihren Code, so dass er nachvollziehbar ist!

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="table">
  <!-- das tabellen-element kopieren. -->
  <xsl:copy>
   <!-- über alle spalten iterieren... -->
   <xsl:for-each select="tr[1]/td">
    <!-- ...sich die aktuelle spalte merken... -->
    <xsl:variable name="col" select="position()"/>
    <!-- ...und für jede spalte eine zeile erzeugen. -->
    <tr>
     <!-- nun über alle eingabezeilen iterieren... -->
     <xsl:for-each select="../../tr">
      <!-- ...sich die aktuelle Zeile merken... -->
      <xsl:variable name="row" select="position()"/>
      <!-- ...und den zelleninhalt kopieren. -->
      <xsl:copy-of select="../tr[$row]/td[$col]"/>
     </xsl:for-each>
    </tr>
   </xsl:for-each>
  </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

5. Beschreiben Sie das Prinzip der `keys` in XSLT. Die beiden wichtigen Aspekte der `keys` sind das *Deklarieren* eines Keys (beschreiben Sie die Deklaration und was dies beim XSLT Prozessor bewirkt) und die *Anwendung* des `keys` in einem XPath. Beschreiben Sie anhand eines kleinen Beispiels, wozu `keys` typischerweise benutzt werden.
   Die Deklaration eines Keys geschieht am Beginn eines XSLT Stylesheets, diese Deklaration instruiert den XSLT Prozessor, dass später auf die im Key selektierten Knoten zugegriffen werden soll. `<xsl:key name="zipkey" match="person" use="address/zip"/>` ist eine solche Deklaration, alle `person` Elemente des Dokuments werden mit dem Wert des `adresse/zip` Elements (von der `person` aus selektiert) in den Key eingetragen. Typische XSLT Prozessoren bauen aus diesem Grunde eine optimierte Zugriffsstruktur für diese Nodes auf, über die schnellere Zugriffe möglich sind. Mit der XSLT `key()` Funktion kann dann auf den Key zugegriffen werden, `key("zipkey","8092")` liefert also ein Node Set,

dessen Inhalt alle `person` Elemente sind, deren `adresse/zip` Kind den Wert 8092 hat. Dieses Beispiel zeigt eine typische Benutzung, vor allem wenn man viele `person` Elemente hat, auf die man oftmals gefiltert nach der PLZ zugreifen will.