Einführungsteils gestellten Testfragen sowie die Lösungen zu den an gleicher Stelle gegebenen Aufgaben.

A Lösungen

A.1 Antworten zu den Testfragen in Kapitel 1

Frage 1: Welcher Knotentyp ist nicht »Child« seines Elternknotens?

a) Der Textknoten

b) Der Attributknoten

c) Der Kommentarknoten

d) Der Processing-Instructions-Knoten

e) Der Namensraumknoten

Richtig sind b) und e). Sowohl Attribut- als auch Namensraumknoten zählen nicht als Kindknoten ihres Elements. Sie stehen damit außerhalb der Dokumentreihenfolge und sind auch nicht Teil des normalen Dokumentbaums. Für XPath-Ausdrücke sind beide Knotentypen nur über ihre jeweilige spezielle Achse erreichbar. Alle anderen hier genannten Knoten befinden sich auf den Elementachsen.

Frage 2: Was versteht man unter dem »value« eines Elementknotens?

a) Text- und Elementinhalt

b) Attributwerte und Textinhalte

c) Eigene Textinhalte und Textinhalte von Child-Elementen

d) Nur die unmittelbaren Child-Textknoten

Richtig ist c). Unter »value« wird stets der Stringwert eines Knotens verstanden. Für einen Elementknoten besteht dieser aus allen in ihm unmittelbar (als eigener direkter Inhalt) und mittelbar (als Inhalte von Kindelementen und deren Kindelementen) enthaltenen Textknoten. Dazu zählen auch Whitespace-Nodes. Für den Stringwert werden alle diese Textknoten in Dokumentreihenfolge verkettet.

Knoten.

Frage 3: Der Document Node ist ...

a) ... das Wurzelelement des verarbeiteten XML-Dokuments.

b) ... das Wurzelelement des XSLT-Stylesheets.

c) ... ein virtueller Knoten, der das verarbeitete XML-Dokument enthält.

Richtig ist c). Ein Dokument Node ist generell ein namen- und elternloser Knoten, der ein Dokument beinhaltet. In der Baumstruktur eines Quelldokuments stellt er den Wurzelknoten dar (nicht das Wurzelelement). Während einer Transformation werden Dokumentknoten erzeugt, die zum Teil als Dokumente serialisiert werden (Ausnahme: temporäre Bäume).

Frage 4: Warum ist der XSL-Namensraum-URI bei `xsl:stylesheet` wichtig?

a) Der Prozessor holt sich Anweisungen unter dieser Adresse.

b) Damit erst werden XSLT-Elemente als Befehle kenntlich.

c) Ganz unwichtig – Hauptsache das Präfix lautet `xsl`.

Richtig ist b). Der XSL-Namensraum-URI dient zur Identifizierung der XSLT-Elemente über ihren qualifizierten Bezeichner. Hierzu müssen diese mittels eines im Grunde beliebig wählbaren – Präfixes an den festgelegten URI-String `http://www.w3.org/1999/XSL/Transform` gebunden sein. Weder wird unter dieser Adresse nach einer realen Ressource gesucht noch muss das an den URI gebundene Präfix zwingend `xsl` lauten.

Frage 5: Welche XSLT-Instruktion stellt eine Nodesequenz zusammen?

a) `xsl:stylesheet`

b) `xsl:template`

c) `xsl:apply-templates`

d) `xsl:value-of`

Richtig ist c). Die Instruktion `xsl:apply-templates` stellt ausgehend von Kontextknoten eine Nodesequenz zusammen, deren Zusammensetzung durch ihr `select`-Attribut bestimmt wird. Liegt kein `select`-Attribut vor, so besteht die Sequenz aus allen Kindknoten des Kontextknotens. Da die Sequenz nur auf der Elementachse zusammengestellt wird, sind keine Attributknoten (und auch keine Namensraumknoten) in ihr enthalten.

Frage 6: Womit wird der XSLT-Prozess eingeleitet?

a) Mit einem Template mit Match auf den Document Node

b) Mit dem ersten Template in Dokumentreihenfolge

c) Mit dem Template mit höchster Priorität

Richtig ist a). Das erste ausgeführte Template ist jenes mit einem Match auf den Dokumentknoten des Quelldokuments (oder das entsprechende Default-Template), da dieser zu Beginn den Kontextknoten darstellt. Die Quelltextreihenfolge der Template-Regeln im Stylesheet ist dabei ohne Bedeutung.

Ausnahme von dieser Regel ist ein explizit als Einstiegs-Template – *initial template* – angeordnetes benanntes(!) Template. Bei Saxon 9 wird ein Start-Template mit der Option `-it:`*`name_des_templates`* befohlen. In diesem Fall braucht kein Quelldokument vorzuliegen.

A.2 Antworten zu den Testfragen in Kapitel 2

Frage 1: Was für eine Sequenz wählt der Pfadausdruck `beispiel[test]`?

a) Alle Kindknoten `<test>` eines `<beispiel>`-Elements

b) Alle Elemente `<beispiel>`, die einen Kindknoten `<test>` besitzen

c) Alle Elemente `<beispiel>` mit einem Attribut `test`

Richtig ist b). Die Ergebnismenge eines Pfadausdrucks wird stets durch den Knotentest des letzten Location Steps bestimmt. In diesem Fall muss es sich also um Elementknoten mit dem Namen `<beispiel>` handeln. Durch ein Predicate wird diese Menge lediglich eingeschränkt. In diesem Fall sind nur die Elemente `<beispiel>` enthalten, die über ein Kindelement `<test>` verfügen.

Frage 2: Was gibt die Funktion `position()` zurück?

a) Den aktuellen Knoten

b) Die Position des aktuellen Knotens im Dokument

c) Den Wert des aktuellen Knotens als Ganzzahl

d) Die Position des aktuellen Knotens in einer Sequenz

Richtig ist d). Die Funktion bezieht sich immer auf die aktuelle Sequenz und gibt die Position des aktuellen Items in Form einer Ganzzahl zurück. Wichtig ist die Funktion, weil der Rückgabewert eben nicht von der Dokumentreihenfolge ab-

hängt. Einen anderen Wert als eine Ganzzahl gibt die Funktion unter keinen Umständen zurück.

Frage 3: Welche Instruktion kopiert auch Attribute eines Elements?

a) Die Instruktion `xsl:copy`

b) Die Instruktion `xsl:copy-of`

c) Die Instruktion `xsl:attribute`

d) Die Instruktion `xsl:attribute-set`

Richtig ist b). Die Instruktion `xsl:copy-of` kopiert generell den gesamten Zweig der ausgewählten Knoten mitsamt Attributen und gegebenenfalls Namensräumen. Dagegen kopiert `xsl:copy` stets nur den Kontextknoten. Attributknoten können ebenfalls kopiert werden, sobald sie Kontextknoten sind, aber eben nicht unmittelbar beim Kopieren ihres Elternelements.

Frage 4: Welches Attribut von `xsl:number` bestimmt die zu zählenden Knoten?

a) Das `format`-Attribut

b) Das `value`-Attribut

c) Das `count`-Attribut

d) Das `level`-Attribut

Richtig ist c). Auch bei multihierarchischer Bezifferung werden die zu berücksichtigenden Elementbezeichner im `count`-Attribut genannt. Die Reihenfolge ihrer Nennung ist dabei irrelevant, da `xsl:number` die Dokumenthierarchie selbst zur Bestimmung der Zuordnung zum Formatstring verwendet. Wird das `value`-Attribut übergeben, so wird nicht gezählt, sondern der so erzeugte Wert lediglich formatiert.

Frage 5: Auf welcher XPath-Achse zählt `xsl:number`?

a) Auf der Preceding-Achse

b) Auf der Preceding-Sibling-Achse

c) Auf der Following-Sibling-Achse

d) Auf der Following-Achse

Richtig ist b). Der Zählvorgang ist stets rückwärts gewandt, wobei auf der Ancestor-Achse des Kontextknotens so weit zurückgegangen wird, bis ein Knoten gefunden wird, auf den das `count`-Attribut passt. Dessen Preceding-Siblings werden

dann gezählt. Mittels des `select`-Attributs kann ein anderer Knoten als Ausgangspunkt gewählt, mittels des `from`-Attributs eine zusätzliche Begrenzung des Arbeitsbereichs der Funktion gesetzt werden.

Frage 6: Welche dieser Funktionen bildet eine eigene Sequenz?

a) Die Funktion `fn:count()`

b) Die Funktion `fn:last()`

c) Die Funktion `fn:sum()`

d) Die Funktion `fn:position()`

Richtig sind a) und c). Die Funktionen `fn:count()` und `fn:sum()` übernehmen als Argument einen XPath-Ausdruck, der eine Sequenz erzeugt, die anschließend von der Funktion als Eingangswert übernommen und verarbeitet wird. Die Funktionen `fn:last()` und `fn:position()` hingegen sind Kontextfunktionen, die keine Argumente übernehmen, sondern sich stets auf die aktuell bestehende Sequenz beziehen.

A.3 Antworten zu den Testfragen in Kapitel 3

Frage 1: Bei mit `xsl:import` importierten Stylesheet-Modulen ...

a) ... hat das zuerst importierte Stylesheet-Modul Vorrang.

b) ... hat das zuletzt importierte Stylesheet-Modul Vorrang.

c) ... sind beide Stylesheet-Module gleichrangig.

Richtig ist b). Das zuletzt importierte Stylesheet-Modul hat Vorrang. Es besitzt zusammen mit seinem ganzen Importzweig (alle von ihm unmittelbar oder mittelbar importierten weiteren Module) eine höhere Importpräzedenz als durch vorhergehende Importinstruktionen importierte Module. Da kein gleichzeitiger Import möglich ist, kann der Fall von identischen Regeln gleicher Importpräzedenz eigentlich nicht vorkommen, es sei denn, es ist zusätzlich die Inklusion von Modulen mit im Spiel.

Frage 2: Bei mit `xsl:include` inkludierten Stylesheet-Modulen ...

a) ... gilt bei Konflikten die zuerst stehende Regel.

b) ... gilt bei Konflikten die zuletzt stehende Regel.

c) ... gelten beide Regeln, da hier keine Importpräzedenz greift.

Richtig ist b). Die zuletzt stehende Regel gilt. Dies ist Konvention bei XSLT-Prozessoren. Ob es sich bei der zuletzt stehenden Regel um eine lokale Regel oder um eine inkludierte Regel handelt, hängt wiederum völlig von der Position der Inklusionsanweisung ab. Obwohl für Inklusionen keine Importpräzedenz gilt, kann es dennoch keine zwei identische und gleichzeitig gültige Regeln geben. Vor einer importierten Regel hat eine inkludierte, gleich einer lokalen Regel, jederzeit Vorrang.

Frage 3: Was ist bei benannten Templates wichtig?

a) Es muss ein Template mit dem aufgerufenen Namen existieren.

b) Das `mode`-Attribut kann beim Aufruf nicht verwendet werden.

c) Das benannte Template darf kein `select`-Attribut besitzen.

d) Es kann ein Parameter übergeben werden.

Richtig sind a) und d). Wird mittels `xsl:call-template` ein benanntes Template aufgerufen, so ist es ein Fehler, wenn kein Template dieses Namens existiert. Kein Fehler ist es allerdings, wenn dieses Template gleichzeitig ein `select`- oder `mode`-Attribut besitzt, weil es parallel als Template-Regel genutzt werden soll. Beim Aufruf darf ein Parameter übergeben werden, für den, falls es sich nicht um einen Tunnelparameter handelt, auch eine diesen auffangende `xsl:param`-Deklaration vorhanden sein sollte.

Frage 4: Ein globaler Parameter ...

a) ... wird immer in einer Template-Regel deklariert.

b) ... besitzt einen Default-Wert.

c) ... kann während der Laufzeit des Stylesheets einen neuen Wert erhalten.

d) ... behält seinen Wert während der gesamten Laufzeit des Stylesheets.

Richtig ist d). Korrekt in Bezug auf einen globalen Parameter ist die Aussage, dass sein Wert über die gesamte Laufzeit des Stylesheets unverändert bleibt. Einen expliziten Default-Wert muss er zwar nicht besitzen – in der Regel wird er von außen mit einem Wert belegt. Ist jedoch kein Default-Wert bestimmt, weil weder ein `select`-Attribut noch ein Sequenzkonstruktor vorhanden sind, wird der Parameterwert auf den leeren String gesetzt. Parmeter in Template-Regeln hingegen sind stets nur lokal gültig.

Frage 5: Bei zwei gleich benannten Variablen ...

a) ... verdeckt die lokale die globale Variable.

b) ... ist immer die globale Variable sichtbar.

c) ... gilt die zuletzt deklarierte.

d) ... gibt es bei zwei lokalen Variablen einen Fehler.

Richtig ist a). Eine lokale Variable verdeckt bei Namensgleichheit generell die globale für die Laufzeit ihrer Template-Regel. Zwei gleich benannte globale Variablen mit gleicher Importpräzedenz stellen immer einen Fehler dar. Für lokale Variablen ist dies zwar nicht unbedingt der Fall, dennoch sollten Sie von dem Versuch absehen, innerhalb einer Template-Regel eine Variable doppelt zu deklarieren.

Frage 6: Welche dieser Aussagen zu `id()` und `generate-id()` sind richtig?

a) `id()` kann in der DTD deklarierte Identifier finden.

b) `id()` kann durch `generate-id()` generierte Identifier finden.

c) `generate-id()` kann mehrere Identifier pro Element erzeugen.

d) `generate-id()` erzeugt je Knoten immer den gleichen Wert.

Richtig sind a) und d). Obwohl `id()` und `generate-id()` einander ähnliche Namen besitzen, sind ihre Aufgaben unterschiedlich. Die Funktion `id()` sucht Knoten anhand des Wertes ihres deklarierten Identifiers. Sowohl die Deklaration als auch der Wert sind explizit und real. Die Funktion `generate-id()` erzeugt einen Laufzeitidentifier, der im Zusammenhang mit einem Knoten stets denselben Wert annimmt. Weder kann `generate-id()` auf reale Identifier zugreifen (die Funktion kann einem Knoten mit `ID` durchaus einen zusätzlichen Laufzeitidentifier geben!) noch kann man mit `id()` sinnvoll auf die rein virtuellen `generate-id()`-Werte zugreifen.

Frage 7: Welche dieser Aussagen zu Schlüsseln sind richtig?

a) Es kann mehrere gleich benannte `key`-Listen geben.

b) Einem Knoten können mehrere Schlüsselwerte zugeordnet sein.

c) Ein Schlüssel muss in der DTD deklariert werden.

d) Ein Schlüssel muss im Haupt-Stylesheet stehen.

Richtig ist b). Schlüssel sind in XSLT durch den XSLT-Prozessor erstellte Tabellen, die Zeiger auf Knoten beinhalten, wobei ein Knoten mehrmals repräsentiert sein kann. Sie werden also während bzw. zu Beginn der Verarbeitung generiert und haben nichts mit einer DTD zu tun. Mehrere gleich benannte Schlüssellisten kann es in der Tat nicht geben, jedoch nicht, weil dies einen Konflikt erzeugt, sondern

weil diese stets zu einer Liste verschmolzen werden. Hierfür spielt es keine Rolle, ob der Schlüssel im Hauptmodul oder anderswo deklariert wurde.

A.4 Lösungen zu den Aufgaben in Kapitel 1

Aufgabe 1a: Stringwert ausgeben

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xhtml"
   encoding="ISO-8859-1"
   indent="yes"
   doctype-public =
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   doctype-system=
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Stringwert 1a)</title>
   </head>
   <body>
      <h1>Memo:</h1>
      <p><xsl:value-of select="memo"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.1 loesungen/01/memo1.xsl

Hier wurde die Template-Regel mit `match="/"` auf den Dokumentknoten ausgerichtet. Von diesem als Current Node ausgehend, kann der Stringwert des Dokumentinhalts innerhalb eines `<p>`-Containers als Literal Result Element mit `<xsl:value-of select="memo"/>` ausgelesen werden.

Alternativ könnte die Template-Regel auf `match="memo"` gesetzt werden. In diesem Fall würde der Dokumentknoten durch die entsprechende Default-Regel verarbeitet und die reale Template-Regel durch diese aufgerufen werden. Der Stringwert müsste, da `<memo>` selbst nun Current Node wäre, allerdings mit `<xsl:value-of select="."/>` extrahiert werden.

Aufgabe 1b: Stringwert ausgeben, alternative Version

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<!-- Output-Deklaration siehe 1.a) -->

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Stringwert 1b)</title>
   </head>
   <body>
      <h1>Memo:</h1>
      <p><xsl:apply-templates/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.2 loesungen/01/memo1b.xsl

Der Unterschied zur vorigen Lösung besteht lediglich in der Verwendung von `<xsl:apply-templates/>` anstelle von `<xsl:value-of select="memo"/>`. Hier wird allerdings nicht direkt ein Stringwert extrahiert, sondern eine Kaskade von Default-Regeln aufgerufen, die letztlich, da für keine Knoten explizite Template-Regeln vorliegen, alle Textknoten des Quelldokuments inklusive der Whitespace-Nodes ins Ergebnisdokument kopieren.

Aufgabe 2a: Stringwerte der Einzelelemente ausgeben

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
```

```
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<!-- Output-Deklaration siehe 1.a) -->

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Stringwert 2a)</title>
   </head>
   <body>
      <h1>Memo:</h1>
      <p><xsl:value-of select="memo/an"/></p>
      <p><xsl:value-of select="memo/von"/></p>
      <p><xsl:value-of select="memo/betreff"/></p>
      <p><xsl:value-of select="memo/text"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.3 loesungen/01/memo2.xsl

Wollen Sie die Stringwerte in einzelne Container ausgeben, so muss dies mit mehreren Instruktionen geschehen. Hierbei muss der Pfad vom Current Node zu den Elementknoten berücksichtigt werden; dieser führt in diesem Fall vom Dokumentknoten über das Wurzelelement.

Wurde die Template-Regel auf `match="memo"` gesetzt, so ist `<memo>` Current Node, und zum Auslesen würde jeweils der Bezeichner des Kindelements genügen: `<xsl:value-of select="an"/>` etc.

Aufgabe 2b: Reihenfolge kontrollieren

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<!-- Output-Deklaration siehe 1.a) -->

<xsl:template match="/">
<html>
```



```
   <head>
      <title>XSLT 2.0 und XPath 2.0: Reihenfolge 2b)</title>
   </head>
   <body>
      <h1>Memo:</h1>
      <p><b>Von: </b>
      <xsl:value-of select="memo/von"/></p>
      <p><b>An: </b>
      <xsl:value-of select="memo/an"/></p>
      <p><b>Betreff: </b>
      <xsl:value-of select="memo/betreff"/></p>
      <p><xsl:value-of select="memo/text"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.4 loesungen/01/memo2b.xsl

Da die Instruktion `xsl:value-of` direkt in das Quelldokument greift, muss für eine Variation der Ausgabereihenfolge lediglich die Position der entsprechenden Instruktion in der Template-Regel geändert werden. Entscheidend ist die den erforderlichen XPath-Ausdruck bestimmende hierarchische Position der gewünschten Information gegenüber dem Current Node, nicht ihre Position in der Dokumentreihenfolge des Quelldokuments.

Aufgabe 2c: Reihenfolge kontrollieren, alternative Version

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<!-- Output-Deklaration siehe 1.a) -->

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Reihenfolge 2c)</title>
   </head>
   <body>
      <h1>Memo:</h1>
```

```
      <p><b>Von: </b>
      <xsl:apply-templates select="memo/von"/></p>
      <p><b>An: </b>
      <xsl:apply-templates select="memo/an"/></p>
      <p><b>Betreff: </b>
      <xsl:apply-templates select="memo/betreff"/></p>
      <p><xsl:apply-templates select="memo/text"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.5 loesungen/01/memo2c.xsl

Die gleiche Wirkung mit `xsl:apply-templates` zu erzielen, erfordert einen ähnlichen Aufwand. Auch hier bestimmt die Position der Instruktion die Ausgabereihenfolge. Die Extraktion des Wertes erfolgt hier jedoch nicht unmittelbar, sondern durch die jeweils gezielte Zusammenstellung einer Nodesequenz aus den gewünschten Elementen (diese Sequenzen umfassen immer nur einen Knoten). Die so zur Verarbeitung angebotenen Knoten werden von den entsprechenden Default-Regeln verarbeitet.

Aufgabe 3: Mehrfachverwendung von Inhalten

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xhtml"
   encoding="ISO-8859-1"
   indent="yes"
   doctype-public =
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   doctype-system =
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<xsl:template match="/">
<html>
   <head>
      <title>
      <xsl:text>Memo von </xsl:text>
```



```
      <xsl:value-of select="memo/von"/>
      <xsl:text> an </xsl:text>
      <xsl:value-of select="memo/an"/>
      </title>
   </head>
   <body>
      <h1>Memo:</h1>
      <p><b>Von: </b>
      <xsl:apply-templates select="memo/von"/></p>
      <p><b>An: </b>
      <xsl:apply-templates select="memo/an"/></p>
      <p><b>Betreff: </b>
      <xsl:apply-templates select="memo/betreff"/></p>
      <p><xsl:apply-templates select="memo/text"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.6 loesungen/01/memo3.xsl

Die Ausgabe im Dokumentkopf der HTML-Seite wird mit `xsl:value-of` vorgenommen, da hier in jedem Fall nur Strings gebraucht werden. Die Ausgabe des Dokumentrumpfs wurde gegenüber der vorigen Lösung nicht weiter geändert.

Aufgabe 4a: Mehrere Template-Regeln

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xhtml"
   encoding="ISO-8859-1"
   indent="yes"
   doctype-public =
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   doctype-system =
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<xsl:template match="/">
<html>
   <head>
```

```
      <title>
      <xsl:text>Memo von </xsl:text>
      <xsl:value-of select="memo/von"/>
      <xsl:text> an </xsl:text>
      <xsl:value-of select="memo/an"/>
      </title>
   </head>
   <body>
      <h1>Memo:</h1>
      <xsl:apply-templates/>
   </body>
</html>
</xsl:template>

<xsl:template match="an">
   <p><b>An: </b><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="von">
   <p><b>Von: </b><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="betreff">
   <p><b>Betreff: </b><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="text">
   <p><xsl:apply-templates/></p>
</xsl:template>

</xsl:stylesheet>
```

Listing A.7 loesungen/01/memo4.xsl

Hier wurde für jedes auszugebende Element eine eigene Template-Regel geschrieben, welche die im jeweiligen Zusammenhang auszugebenden Literal Result Elements und eine `xsl:apply-templates`-Instruktion enthält.

Da die Ausgabe in Dokumentreihenfolge bleiben kann, wurde (vergleichbar mit Lösung 1b) eine `xsl:apply-templates`-Instruktion in der Template-Regel für den Dokumentknoten eingesetzt. Diese stellt eine Nodesequenz zusammen, die den Elementknoten `<memo>` enthält, der von der Default-Regel für Elementknoten verarbeitet wird. Diese stellt danach die Nodesequenz zusammen, die die weiteren Template-Regeln aktiviert.

Aufgabe 4b: Mehrere Template-Regeln, Reihenfolge kontrollieren

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xhtml"
   encoding="ISO-8859-1"
   indent="yes"
   doctype-public =
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   doctype-system =
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<xsl:template match="/">
<html>
   <title>
      <xsl:text>Memo von </xsl:text>
      <xsl:value-of select="memo/von"/>
      <xsl:text> an </xsl:text>
      <xsl:value-of select="memo/an"/>
   </title>
   <body>
      <h1>Memo:</h1>
      <xsl:apply-templates select="memo/von"/>
      <xsl:apply-templates select="memo/an"/>
      <xsl:apply-templates select="memo/betreff"/>
      <xsl:apply-templates select="memo/text"/>
   </body>
</html>
</xsl:template>

<xsl:template match="an">
   <p><b>An: </b><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="von">
   <p><b>Von: </b><xsl:apply-templates/></p>
</xsl:template>

<xsl:template match="betreff">
   <p><b>Betreff: </b><xsl:apply-templates/></p>
</xsl:template>
```

```
<xsl:template match="text">
   <p><xsl:apply-templates/></p>
</xsl:template>

</xsl:stylesheet>
```

Listing A.8 loesungen/01/memo4b.xsl

Die Variante mit veränderter Reihenfolge erfordert die Zusammenstellung von spezifischen Nodesequenzen in der gewünschten Abfolge. Dies entspricht der Lösung 2c mit dem Unterschied, dass in diesem Falle explizite Template-Regeln für die einzelnen Elemente vorhanden sind.

Aufgabe 5: Attribute auslesen, Output HTML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Beachten Sie das Fehlen des Defaultnamensraums! -->

<!-- KEINE Output-Deklaration, d.h. HTML-Erzeugung -->

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 5)</title>
   </head>
   <body>
      <h2><xsl:value-of select="buch/@autor"/>
      <xsl:text>: "</xsl:text>
      <xsl:value-of select="buch/@titel"/>
      <xsl:text>"</xsl:text></h2>
      <p><xsl:text>in der Übersetzung von </xsl:text>
      <i><xsl:value-of select="buch/@übersetzer"/></i></p>
      <p><xsl:value-of select="buch/@verlag"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="buch/@ersch-jahr"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.9 loesungen/01/buch-attrib-html.xsl

Das Auslesen von Attributen unterscheidet sich nicht wesentlich vom Auslesen des Stringwertes von Elementen. Der Unterschied besteht darin, dass im entsprechenden Location Step auf die Attributachse gewechselt werden muss. Das Ergebnis zeigt das üblichen Meta-Tag für HTML-Content. Umlaute werden mit Entities umschrieben, das Encoding ist UTF-8.

Ergebnis:

```
<html>
   <head>
   <meta http-equiv="Content-Type"
         content="text/html; charset=UTF-8">
   <title>XSLT 2.0 und XPath 2.0: Buchdatei 5)</title>
   </head>
   <body>
      <h2>Mike Miller: "XML - Das verflixte Attribut"</h2>
      <p>in der &Uuml;bersetzung von <i>Peter Panter</i></p>
      <p>Galileo Verlag, 2010</p>
   </body>
</html>
```

Listing A.10 loesungen/01/buch-attrib-html.html

Aufgabe 6a: Verwendung von `xsl:output, "xhtml"`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/1999/xhtml">

<!-- Defaultnamensraum für XHTML gesetzt -->

<!-- Outputsteuerung XHTML: -->
<xsl:output method="xhtml"
   encoding="ISO-8859-1"
   indent="yes"
   doctype-public =
   "-//W3C//DTD XHTML 1.0 Strict//EN"
   doctype-system =
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd" />

<xsl:template match="/">
<html>
   <head>
```

```
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 6a)</title>
   </head>
   <body>
      <h2><xsl:value-of select="buch/@autor"/>
      <xsl:text>: "</xsl:text>
      <xsl:value-of select="buch/@titel"/>
      <xsl:text>"</xsl:text></h2>
      <p><xsl:text>in der Übersetzung von </xsl:text>
      <i><xsl:value-of select="buch/@übersetzer"/></i></p>
      <p><xsl:value-of select="buch/@verlag"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="buch/@ersch-jahr"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.11 loesungen/01/buch-attrib-xhtml.xsl

Dank des entsprechend eingesetzten `xsl:output` unterscheidet sich die XHTML-Version in 6a) von der Lösung 5) in bestimmten Details. Diese bestehen im vorangestellten XML-Prolog, der DOCTYPE-Deklaration, dem XHTML-Namensraum im Wurzelelement und der Nichtverwendung von HTML-Entities für die Umlaute in der Datei. Hier wird ebenfalls das Content-Meta-Tag eingefügt, jedoch mit dem angeordneten Encoding ISO-8859-1.

Ergebnis:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type"
       content="text/xhtml; charset=ISO-8859-1"/>
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 6a)</title>
   </head>
   <body>
      <h2>Mike Miller: "XML - Das verflixte Attribut"</h2>
      <p>in der Übersetzung von <i>Peter Panter</i></p>
      <p>Galileo Verlag, 2010</p>
   </body>
</html>
```

Listing A.12 loesungen/01/buch-attrib-xhtml.html

Aufgabe 6b: Verwendung von `xsl:output, method = "xml"`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Beachten Sie das Fehlen des Defaultnamensraums! -->

<!-- Outputsteuerung XML: -->
<xsl:output method="xml"
   encoding="ISO-8859-1"
   indent="yes"/>

<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 6b)</title>
   </head>
   <body>
      <h2><xsl:value-of select="buch/@autor"/>
      <xsl:text>: "</xsl:text>
      <xsl:value-of select="buch/@titel"/>
      <xsl:text>"</xsl:text></h2>
      <p><xsl:text>in der Übersetzung von </xsl:text>
      <i><xsl:value-of select="buch/@übersetzer"/></i></p>
      <p><xsl:value-of select="buch/@verlag"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="buch/@ersch-jahr"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.13 loesungen/01/buch-attrib-xml.xsl

Wird das Dokument mit `method="xml"` ausgegeben, so kann der XHTML-Namensraum entfallen. Er muss jedoch nicht. – In diesem Fall erhält man auf Umwegen ebenfalls ein XHTML-Dokument, allerdings ohne Doctype-Deklaration. Eine Maskierung der Umlaute durch HTML-Entities findet hier selbstverständlich ebenfalls nicht statt.

Ergebnis:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 6b)</title>
   </head>
   <body>
      <h2>Mike Miller: "XML - Das verflixte Attribut"</h2>
      <p>in der Übersetzung von <i>Peter Panter</i>
      </p>
      <p>Galileo Verlag, 2010</p>
   </body>
</html>
```

Listing A.14 loesungen/01/buch-attrib-xml.html

In diesem Fall fehlt dem Dokument der XHTML-Namensraum und das für die HTML- und XHTML-Ausgabe automatisch eingefügte Content-Type-Meta-Tag: Dies ist für den XSLT-Prozessor also »nur« ein XML-Dokument, das zufällig HTML-Tag-Namen verwendet.

Aufgabe 6c: Verwendung von `xsl:output, method="text"`

Wird das Dokument mit gleich lautendem Stylesheet, allerdings mit modifizierter Output-Deklaration, die jetzt das Format `"text"` nennt, verarbeitet,

```
...

<xsl:output method="text"
   encoding="ISO-8859-1"
   indent="yes"/>

...
```

Listing A.15 loesungen/01/buch-attrib-text.xsl

so erhalten Sie ein einfaches Textdokument aus den Stringinhalten des Quelldokuments und der Literal Result Elements des Stylesheets. Das `indent`-Attribut wird ignoriert und alle HTML-Tags werden verworfen.

Ergebnis:

```
BuchdateiMike Miller: "XML - Das verflixte Attribut"
in der Übersetzung von Peter Panter, Galileo Verlag, 2010
```

Listing A.16 loesungen/01/buch-attrib-text.txt

Dass das `encoding`-Attribut dagegen nicht ignoriert wird, zeigt sich, wenn man es versuchsweise weglässt. Auch `xsl:text`-Instruktionen wurden hier entfernt. Dies führt dazu, dass die ansonsten – erwünscht – wegfallenden Zeilenumbrüche im Ergebnis landen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Outputsteuerung Text: -->
<xsl:output method="text"/>

<xsl:template match="/">
   <html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Buchdatei 6c)</title>
   </head>
   <body>
      <h2><xsl:value-of select="buch/@autor"/>:
      "<xsl:value-of select="buch/@titel"/>"
      </h2>
      <p>in der Übersetzung von
      <i><xsl:value-of select="buch/@übersetzer"/></i></p>
      <p><xsl:value-of select="buch/@verlag"/>,
      <xsl:value-of select="buch/@ersch-jahr"/></p>
   </body>
</html>
</xsl:template>

</xsl:stylesheet>
```

Listing A.17 loesungen/01/buch-attrib-text-b.xsl

Die sich so ergebende Textdatei sieht nun folgendermaßen aus. Beachten Sie die Wirkung des fehlenden Encodings auf die Umlaute:

Ergebnis:

```
BuchdateiMike Miller:
"XML - Das verflixte Attribut"
in der Ã¤bersetzung von
Peter PanterGalileo Verlag,
2010
```

Listing A.18 loesungen/01/buch-attrib-text-b.txt

A.5 Lösungen zu den Aufgaben in Kapitel 2

Die folgenden Aufgaben basieren auf der Datei *buchhandel.xsl*.

Aufgabe 1: Filtern und Sortieren

Wesentlich ist das Filtern der Nodesequenz mit einem Predicate, um die Bücher mit einem Erscheinungsjahr ab 2000 auszufiltern. Die Sortierung erfolgt anschließend.

Der Übersichtlichkeit halber wird das Erscheinungsjahr ebenfalls ausgegeben, und zwar im Zusammenhang mit der Formatierung des Elements `<buchverlag>`.

```
<xsl:template match="buchhandel">
   <xsl:apply-templates
     select="buch[@ersch-jahr &lt; 2000]" >

        <xsl:sort select="@ersch-jahr"/>

   </xsl:apply-templates>
</xsl:template>

<xsl:template match="buchverlag">
   <br/><b>Verlag: </b>
   <xsl:apply-templates/>,
   <!-- Ausgabe Erscheinungsjahr -->
   <xsl:value-of select="../@ersch-jahr"/>;
</xsl:template>
```

Listing A.19 loesungen/02/buchhandel-a1.xsl (Auszug)

Ausgabeergebnis (Auszug):

Unter Korallen und Haien von *Hass, Hans*;
Verlag: Ozean Verlag, 1950; Preis: 22.95 Euro
(ISBN liegt nicht vor.)

Ganz oben von *Hansen, Carl*;
Verlag: Berg Verlag, 1960; Preis: 20.95 Euro
(ISBN liegt nicht vor.)

Listing A.20 loesungen/02/buchhandel-a1.html

Aufgabe 2: Filtern und Sortieren

Die Vorgehensweise ist analog zu Aufgabe 1. Das erforderliche Predicate muss mit einer logischen OR-Verknüpfung aus den beiden zu verschmelzenden Bedingungen gebildet werden. Die Resultatmenge entspricht der Vereinigungsmenge. Bei der Sortierung nach Autoren muss die Zahl der Knoten für das Sortierkriterium auf einen begrenzt werden. Dies geschieht mit einem numerischen Predicate.


```
<xsl:template match="buchhandel">
   <xsl:apply-templates
     select="buch[@ersch-jahr &gt; 1998 or @kat='reisen']" >

        <xsl:sort select="buchautor[1]"/>

   </xsl:apply-templates>
</xsl:template>
```

Listing A.21 loesungen/02/buchhandel-a2.xsl (Auszug)

Ausgabeergebnis (Auszug):

Yet Another Hitchhiker's Guide to the Galaxy von *Adams, Douglas*;
Verlag: Spacetrotter Publishing, 2001; Preis: 17.00 Dollar
(ISBN: 3-899-77610-5)

Hitchhiker's Guide to the Galaxy von *Adams, Douglas*;
Verlag: Spacetrotter Publishing, 1997; Preis: 12.00 Dollar
(ISBN: 3-899-77615-4)

Listing A.22 loesungen/02/buchhandel-a2.html

Aufgabe 3a: Filtern, Zählen und Summieren

Der Kern der Lösung besteht wiederum in einem zusammengesetzten Predicate mit logischer Verknüpfung. Dieses Predicate wird gleichermaßen bei der Zusammenstellung der verarbeiteten Knotensequenz als auch beim Zählvorgang, der Preissummierung und der Bildung des Durchschnittspreises eingesetzt.

```
<xsl:template match="buchhandel">
   <p><b>Statistik:</b></p>
   <p>Bücher in der Liste:
      <xsl:value-of
       select="fn:count( buch[buchverlag='Berg Verlag'
                  or buchverlag='Ozean Verlag'] )"/>
   </p>
   <p>Gesamtpreis aller Bücher:
     <xsl:value-of
       select="format-number(
                fn:sum( buch[buchverlag='Berg Verlag'
```

```
                        or buchverlag='Ozean Verlag']/
preis                         ),
                '##00.00'
               )"/>
   </p>
   <p>Durchschnittspreis aller Bücher:
     <xsl:value-of
       select="format-number(
                fn:sum( buch[buchverlag='Berg Verlag'
                        or buchverlag='Ozean Verlag']/preis
                       ) div fn:count(
                              buch[buchverlag='Berg Verlag'
                              or buchverlag='Ozean Verlag']
                                    ),
                '##00.00'
               )"/></p>
     <xsl:apply-templates
       select="buch[buchverlag='Berg Verlag'
                        or buchverlag='Ozean Verlag']" />
</xsl:template>
```

Listing A.23 loesungen/02/buchhandel-a3.xsl

Ausgabeergebnis (Auszug):

Bücher in der Liste: 8
Gesamtpreis aller Bücher: 135.85
Durchschnittspreis aller Bücher: 16.98

Unter Korallen und Haien von *Hass, Hans*;
Verlag: Ozean Verlag, 1950; Preis: 22.95 Euro
(ISBN liegt nicht vor.)

Listing A.24 loesungen/02/buchhandel-a3.html

Aufgabe 3b: Filtern und Berechnen

Die Filterung der Bücher nach Währungsangabe »Dollar« erfolgt beim Zusammenstellen der Nodesequenz. Das Predicate stellt einen Pfadausdruck dar, der vom Current Node zu dessen `<preis>`-Kindknoten und dessen `waehrung`-Attribut zeigt. Die Umrechnung und Formatierung des Wertes geschieht bei der Ausgabe des Preises direkt. (Es wurde ein Umrechnungsfaktor Dollar/Euro von 0,68054 zugrunde gelegt.)

Bitte beachten Sie, dass der mit Komma angegebene Umrechnungsfaktor im Stylesheet mit einem Punkt als Dezimaltrenner eingesetzt werden muss!

```
<xsl:template match="buchhandel">
   <p><b>Statistik:</b></p>
   <!-- Bücher mit Dollarpreis zählen: -->
   <p>Bücher in der Liste:
      <xsl:value-of
        select="fn:count(buch[preis/@waehrung='Dollar'])"/>
   </p>
   <!-- Bücher mit Dollarpreis ausgeben: -->
      <xsl:apply-templates
        select="buch[preis/@waehrung='Dollar']" />
</xsl:template>

<xsl:template match="preis">
    Preis: <xsl:apply-templates/><xsl:text> </xsl:text>
   <xsl:value-of select="@waehrung"/>
    entspricht
   <xsl:value-of
    select="format-number(. * 0.68054, '##0.00')"/> Euro
</xsl:template>
```

Listing A.25 loesungen/02/buchhandel-a3b.xsl

Ausgabeergebnis (Auszug):

Bücher in der Liste: 5

Where Wizards Stay Up Late von *Hafner, Katie; Lyon, Matthew*;
Verlag: Touchstone, 1996; Preis: 14.00 Dollar entspr. 9.52 Euro
(ISBN: 3-899-77609-7)

Yet Another Hitchhiker's Guide to the Galaxy von *Adams, Douglas*;
Verlag: Spacetrotter Publ., 2001; Preis: 17.00 Dollar entspr. 11.56 Euro
(ISBN: 3-899-77610-5)

Listing A.26 loesungen/02/buchhandel-a3b.html

Aufgabe 4: Bedingungen

Die Sortierung der Nodesequenz erfolgt mit `xsl:sort` unter Angabe des Titels als Sortierkriterium. Die Vermerke werden mittels `xsl:choose` eingefügt, wobei zwei Elemente `xsl:when` mit entsprechenden Bedingungen, aber kein `xsl:otherwise` eingesetzt werden.

```
<xsl:template match="buchhandel">
   <p><b>Bücher nach Titel:</b></p>
   <xsl:apply-templates>
      <xsl:sort select="buchtitel"/>
   </xsl:apply-templates>
</xsl:template>

<xsl:template match="buch">
<p><!-- neu oder vergriffen? -->
   <xsl:choose>
      <xsl:when test="@ersch-jahr &gt; 2003">
         <b>Neu erschienen:</b><br/>
      </xsl:when>
      <xsl:when test="@ersch-jahr &lt; 1975">
         <i>Vergriffen!</i><br/>
      </xsl:when>
   </xsl:choose>
   <!-- jetzt die Buchdaten ausgeben: -->
   <xsl:apply-templates/>
   <!-- liegt ISBN-Nummer vor? -->
   <xsl:choose>
      <xsl:when test="@isbn != ''">
         <br/>(ISBN: <xsl:value-of select="@isbn"/>)
      </xsl:when>
      <xsl:otherwise>
         <br/>(ISBN-Nummer liegt nicht vor.)
      </xsl:otherwise>
   </xsl:choose>
</p>
</xsl:template>
```

Listing A.27 loesungen/02/buchhandel-a4.xsl

Ausgabeergebnis (Auszug):

Vergriffen!
Ganz oben von *Hansen, Carl*;
Verlag: Berg Verlag, 1960; Preis: 20.95 Euro
(ISBN liegt nicht vor.)

Neu erschienen:
Irreguläre Ausdrücke von Esser, Klaus;
Verlag: Java Verlag, 2004; Preis: 27.00 Euro
(ISBN liegt nicht vor.)

Listing A.28 loesungen/02/buchhandel-a4.html

Aufgabe 5: Gruppieren und Nummerieren

Die Gruppierung erfolgt relativ einfach mit `xsl:for-each-group`, wobei das Erscheinungsjahr als `current-grouping-key()` und die Titel jeweils als Sequenz `current-group()` in einer `xsl:for-each`-Schleife ausgegeben werden. Die Nummerierung geschieht indirekt durch eine HTML-`<ol>`-Liste. Alternativ könnte `position()` hier eingesetzt werden, nicht aber `xsl:number` (es sei denn mit `value`-Attribut), da die Buchtitel nicht in Dokumentreihenfolge ausgegeben werden.

```
<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: Gruppieren</title>
   </head>
   <body>
      <h1>Bücher nach Erscheinungsjahr gruppiert</h1>
      <!-- Bildung der Gruppen: -->
      <xsl:for-each-group
           select="buchhandel/buch"
           group-by="@ersch-jahr">
      <xsl:sort select="@ersch-jahr"/>
      <h3>Erscheinungsjahr:
         <xsl:value-of select="current-grouping-key()"/>
      </h3>
      <ol>
         <!-- Ausgabe der Einzeltitel: -->
         <xsl:for-each select="current-group()/buchtitel">
           <li>
             <xsl:value-of select="."/>
           </li>
         </xsl:for-each>
      </ol>
      </xsl:for-each-group>
   </body>
</html>
</xsl:template>
```

Listing A.29 loesungen/02/buchhandel-a5.xsl

Ausgabeergebnis (Auszug):

Erscheinungsjahr: 2003

1.Java für Theoretiker Bd.1

2.Reguläre Ausdrücke
3.Java für Theoretiker Bd.2

Listing A.30 loesungen/02/buchhandel-a5.html

A.6 Lösungen zu den Aufgaben in Kapitel 3

Aufgabe 1: Benannte Templates

Die Lösung besteht aus einem benannten Template, das über `xsl:call-template` nach der Zusammenstellung und Verarbeitung der jeweiligen Inhalte des Kapitels aufgerufen wird. Das benannte Template verhält sich völlig statisch und liefert stets den gleichen Output.

```
<xsl:template match="kapitel">
   <xsl:apply-templates/>
   <xsl:call-template name="kapitelende"/>
</xsl:template>

<xsl:template name="kapitelende">
   <div align="center">
      *** Ende des Kapitels ***
   </div>
</xsl:template>
```

Listing A.31 loesungen/03/a1-artikel.xsl

Ausgabeergebnis (Auszug):

... Hier wird dagegen das über den Namen aufgerufene Template direkt in den Prozess hineingezogen (Pull-Verfahren).

*** Ende des Kapitels ***

Listing A.32 loesungen/03/a1-artikel.html

Aufgabe 2: Variablen und Parameter

Der Titel des jeweiligen Kapitels wird ausgelesen und an das benannte Template, das den Trenner schreibt, per Parameter übergeben. Für Notfälle besitzt der Zielparameter einen Default-Wert.

```
<xsl:template match="kapitel">
   <xsl:apply-templates/>
   <xsl:call-template name="kapitelende">
      <xsl:with-param name="text" select="titel"/>
   </xsl:call-template>
</xsl:template>
```



```
<xsl:template name="kapitelende">
   <xsl:param name="text">diesem Kapitel</xsl:param>
   <div align="center">
      *** Ende von <xsl:value-of select="$text"/> ***
   </div>
</xsl:template>
```

Listing A.33 loesungen/03/a2-artikel.xsl

Ausgabeergebnis (Auszug):

... Hier wird dagegen das über den Namen aufgerufene Template direkt in den Prozess hineingezogen (Pull-Verfahren).

*** Ende von Kapitel 1 ***

Listing A.34 loesungen/03/a2-artikel.html

Aufgabe 3: Laufzeitidentifier

Mittels `generate-id()` wird ein Inhaltsverzeichnis mit Sprunglinks der Ausgabe eines in XML vorliegenden Artikels vorangestellt.

```
<xsl:template match="/">
<html>
   <head>
      <title>XSLT 2.0 und XPath 2.0: generate-id</title>
   </head>
   <body>
      <h1>Inhaltsverzeichnis mit generate-id()</h1>
      <ul>
      <xsl:for-each select="//titel">
         <li>
            <a href="#{generate-id()}">
               <xsl:value-of select="."/>
            </a>
         </li>
      </xsl:for-each>
      </ul>
      <xsl:apply-templates/>
   </body>
</html>
</xsl:template>

<xsl:template match="kapitel">
   <h3 id="{generate-id(titel)}">
      <xsl:value-of select="titel"/>
```

```
    </h3>
    <p>
       <xsl:value-of select="text"/>
    </p>
    <p>
       <a href="#">nach oben</a>
    </p>
    <hr width="60%" noshade="noshade" size="1"/>
</xsl:template>
```

Listing A.35 loesungen/03/a4-generate-id.xsl

Aufgabe 4: Unparsed Entities

Die Einbindung der in der DTD als »unparsed Entities« deklarierten Grafiken erfolgt im entsprechenden Template mittels der Funktion `unparsed-entity-uri()`. Um dem Bild die gewünschten Attributwerte zu geben, wird ein deklariertes Attributset mit `xsl:attribute-set` adressiert und im Literal Result Element `<img>` über dessen Attribut `xsl:use-attribute-sets` eingebunden.

Beachten Sie, dass `use-attribute-sets` in diesem Fall, da es innerhalb eines Literal Result Elements eingesetzt wird, mit Präfix `xsl` versehen wird, da es im XSLT-Namensraum ist! Dies ist erforderlich, da das Literal Result Element eben nicht im XSLT-Namensraum sein darf.

```
<xsl:attribute-set name="bild-att">
   <xsl:attribute name="hspace">10</xsl:attribute>
   <xsl:attribute name="vspace">0</xsl:attribute>
</xsl:attribute-set>

<xsl:template match="buch">
   <xsl:apply-templates select="bild"/>
   ...
</xsl:template>

<xsl:template match="bild">
   <img src="{unparsed-entity-uri(@quelle)}"
    xsl:use-attribute-sets="bild-att"/>
</xsl:template>
```

Listing A.36 loesungen/03/a4-buchdaten.xsl

Aufgabe 5: Unparsed Entities

Das Template für `<buch>`-Elemente wird um die Ausgabe eines HTML-`<img>` erweitert, das mittels einen Attributwert-Templates als Breitenangabe den aus dem Element `<verkauf>` (Child des Current Nodes) ausgelesenen Wert erhält.


```
<xsl:template match="buch">
   <xsl:apply-templates select="bild"/>
   ...
   <p><img src="rot.gif" height="5" width="{verkauf}"/>
   Verkaufte Exemplare: <xsl:value-of select="verkauf"/></p>
   <hr/>
</xsl:template>
```

Listing A.37 loesungen/03/a4-buchdaten.xsl

Aufgabe 6: Die XSLT-Funktion `document()`

Das externe Dokument wird mittels der XSLT-Funktion `document()` in eine globale Variable gespeichert. Der Inhalt der Variablen wird im Buch-Template referenziert, und zum aktuellen Buchtitel wird aus dem externen Dokument die passende Kurzkritik herausgesucht und ausgegeben.

```
<!-- globale Variable für externes Dokument: -->
<xsl:variable name="inhalt"
  select="document('a6-buchdaten-inhalte.xml')"/>

<!-- hier Elemente "buch" verarbeiten: -->
<xsl:template match="buch">
   <xsl:apply-templates select="bild"/>
   ...
   <!-- hier ist die Inhaltsangabe eingebunden: -->
   <xsl:variable name="der_titel" select="buchtitel"/>

   <p><xsl:apply-templates
        select="$inhalt//inhalt[@titel=$der_titel]"/></p>

   <!-- hier Breite der Verkaufsgrafik formatieren: -->
   <p><img src="rot.gif" height="5" width="{verkauf}"/>
       Verkaufte Exemplare:
      <xsl:value-of select="verkauf"/></p>
   <hr/>
</xsl:template>

<!-- hier Elemente "inhalt" verarbeiten: -->
<xsl:template match="inhalt">
   <p><xsl:value-of select="."/></p>
</xsl:template>
```

Listing A.38 loesungen/03/a4-buchdaten.xsl