Cursistenmap

XML deel 2: XPath en XSLT

Dit naslagwerk kunt u na afloop van de cursus meenemen.

Versie 1.1, maart 2014

Inhoudsopgave

Inhoudsopgave

1 Inleiding

Er zijn tegenwoordig weinig applicaties meer te vinden waarin XML geen rol speelt. Dit geldt met name voor web applicaties, die vaak deels gevoed worden uit externe gegevens. Een belangrijke reden van de populariteit van XML is namelijk dat brongegevens in het ene formaat, zoals bijvoorbeeld gebruikt door een externe partij, eenvoudig kunnen worden omgezet naar een ander doelformaat.

Deze cursus behandelt de transformatie-taal XSLT. Hierbij gaan we ervan uit dat u al een zekere XSLT basiskennis heeft opgedaan, zoals behandeld in de cursus XML Introductie. Een deel van deze basiskennis zullen we in het eerste hoofdstuk kort herhalen, om daarna snel dieper in te gaan op XSLT en XPath. Vervolgens behandelen we de extra functionaliteit van XSLT 2.0 en XPath 2.0. waarbij de verschillen met XSLT 1.0 aan de orde komen.

1 Inleiding

2 XSLT versie 1.0

2.1 Inleiding

In de cursus XML Introductie is het gebruik van XSLT versie 1.0 al aan de orde geweest. In dit hoofdstuk gaan we daar uitgebreider op in. Om te beginnen zullen we de de opbouw van stylesheets nader toelichten. Met XPath Axes kunnen verschillende elementen in de structuur benaderd worden. We zullen zien hoe dit in een transformatie gebruikt kan worden. Ook zal het gebruik van templates zal uitgewerkt worden. Daarnaast komen diverse XPath functies en XSLT functies aan bod.

2.2 Het XSLT stylesheet

Om de vertaling van het ene document naar het andere document te maken gebruiken we XSLT stylesheets. XSLT is een taal die vastgelegd is door het w3c. Dit kan worden gebruikt door een verwijzing naar de namespace van XSLT: http://www.w3.org/1999/XSL/Transform op te nemen. We zullen nog kort de opbouw van een XSLT stylesheet bespreken.

2.2.1 Opbouw XSLT

Een stylesheet heeft net als andere XML documenten een rootelement. Bij een stylesheet is dit het element `<stylesheet>`. Dit element komt uit de namespace http://www.w3.org/1999/XSL/Transform. Daarom moet er in dit element ook naar deze namespace verwezen worden door middel van het attribuut `xmlns`.

```
<xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform">
```

Vanaf nu zullen we elementen uit deze namespace steeds benoemen inclusief de prefix xsl.

Naast het attribuut `xmlns` is een het attribuut `version` ook een verplicht attribuut.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Het stylesheet element bevat vaak verwijzingen naar andere namespaces die gebruikt worden in de stylesheet. Daarnaast kunnen de volgende attributen voorkomen:

`extension-element-prefixes="..."`
 	
`(... = door spaties gescheiden lijst prefixes van namespaces)`
	Er kunnen externe modulen aan een stylesheet worden toegevoegd, voor extra functionaliteit, zoals bijvoorbeeld het berekenen van een minimum waarde. Via dit attribuut kan worden aangegeven dat de betreffende elementen instructies bevatten, die niet bedoeld zijn om weer te geven in de uitvoer.

`exclude-result-prefixes="..."`
 	
`(... = door spaties gescheiden lijst prefixes van namespaces)`
	Standaard wordt de namespace van een bronelement ook weergegeven in de uitvoer. Met dit attribuut kunnen we de prefixes opsommen van uitgezonderde namespaces: deze elementen krijgen in de uitvoer geen namespace mee.

We komen verderop in deze cursus op deze attributen terug.

Binnen het stylesheet element kunnen diverse andere elementen voorkomen. De volgende lijst elementen kan worden gebruikt. Deze elementen zijn optioneel.

`<xsl:import href="..."/>`	Importeert een andere stylesheet.
`<xsl:include href="..."/>`	Voegt een andere stylesheet in.
`<xsl:strip-space elements="..."/>`	Geeft een lijst op van elementen die niet in de uitvoer terechtkomen als ze alleen whitespace (spaties, tabs, returns e.d.) bevatten. Met `elements="*"` kunnen alle elementen worden aangeduid.
`<xsl:preserve-space elements="..."/>`	Alleen in combinatie met strip-space-elements: geeft uitzonderingen op bovenstaande lijst aan. Deze komen dus wel in de uitvoer terecht als ze alleen whitespace bevatten.
`<xsl:output method="..."/>`	Geeft aan wat het type is van de uitvoer (xml, html of text).
`<xsl:key name="..." match="..." use="..."/>`	Met behulp van een key kunnen bepaalde elementen sneller worden gevonden. Vergelijkbaar met een index voor database tabellen.
`<xsl:decimal-format name="..."/>`	Geeft aan hoe decimalen worden weergegeven.
`<xsl:namespace-alias stylesheet-prefix="..." result-prefix="..."/>`	Hiermee kan een bron-namespace naar een andere resultaat-namespace worden omgezet.
`<xsl:attribute-set name="..."> ... </xsl:attribute-set>`	Een attribute-set definieert een verzameling attributen die aan een uitvoer-element kan worden toegekend.
`<xsl:variable name="...">...</xsl:variable>`	Hiermee kan een variabele worden aangemaakt met een waarde, waar verderop in de stylesheet naar kan worden terug verwezen.
`<xsl:param name="...">...</xsl:param>`	Bij het uitvoeren van stylesheets kunnen ook parameters worden meegegeven. Met xsl:param kan zo'n parameterwaarde worden opgevangen om verderop te gebruiken.
`<xsl:template match="..."> ... </xsl:template>`	Templates geven aan wat er moet gebeuren als een element wordt gevonden in het bronbestand dat overeenkomt met het patroon in match="...".

Veel van deze elementen zullen we verderop in deze cursus nader toelichten.

Binnen het element stylesheet vinden we meestal het element: `<xsl:template match="/">`
Op deze manier geven we aan dat bij het transformeren bij het root element van het originele XML document begonnen moet worden. Binnen het element template kan de structuur van het nieuw te bouwen document worden opgegeven.

Voorbeeld:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:pers="http://pers.nl">
    <xsl:template match="/">
      <leden>
        <naam>
        </naam>
      </leden>
    </xsl:template>
</xsl:stylesheet>
```

In plaats van het element `<xsl:stylesheet>` wordt ook wel het element `<xsl:transform>` gebruikt. De attributen die binnen deze elementen kunnen worden gebruikt zijn gelijk.

2.2.2 Value-of

Om een waarde uit het originele XML document te halen kan het element `<xsl:value-of/>` worden gebruikt. Dit element heeft een attribuut `select`. Bij dit attribuut kan met behulp van een XPath expressie de gewenste node naam worden opgeven. De inhoud van de betreffende node uit het originele document wordt op deze manier aan het nieuwe document toegevoegd.

Voorbeeld:

```
<leden>
    <naam>
      <xsl:value-of select="/pers:voornaam"/>
      <xsl:value-of select="/pers:achternaam"/>
    </naam>
</leden>
```

Let op: indien het select attribuut meerdere elementen ophaalt, wordt alleen van het eerste element de tekstwaarde opgehaald. Dat betreft dan ook alle tekst van eventuele sub-elementen van dat eerste element. In het volgende hoofdstuk wordt XSLT versie 2.0 besproken, waarin deze functionaliteit verandert.

2.2.3 For-each

Een ander belangrijk element is het element `<xsl:for-each>`. Als een node meer dan één keer voorkomt, kan `<xsl:for-each>` worden gebruikt om al deze nodes te doorlopen. Ook dit element heeft een attribuut `select` om de juiste node te selecteren.

Voorbeeld:

```
<leden>
    <xsl:for-each select="/pers:personen/pers:persoon">
      <naam>
        <xsl:value-of select="pers:voornaam"/>
        <xsl:value-of select="pers:achternaam"/>
      </naam>
    </xsl:for-each>
</leden>
```

2.2.4 Andere veel gebruikte elementen

Om het resultaat van een `<xsl:for-each>` te sorteren, kan het element `<xsl:sort>` gebruikt worden. Dit element heeft een attribuut `select` om aan te geven op welke node gesorteerd moet worden.

Voorbeeld:

```
<nummers>
        <xsl:for-each select="/cd:cdlijst/cd:cd/cd:tracklist/cd:track">
          <xsl:sort select="./cd:titel"/>
            <nummer>
              <titel>
                <xsl:value-of select="./cd:titel"></xsl:value-of>
              </titel>
              <artiest>
                <xsl:value-of select="../../cd:artiest"></xsl:value-of>
              </artiest>
              <jaar>
                <xsl:value-of select="./cd:jaar"></xsl:value-of>
              </jaar>
              <duur>
                <xsl:value-of select="./@speelduur"></xsl:value-of>
              </duur>
            </nummer>
          </xsl:for-each>
</nummers>
```

Bij een transformatie kan gebruik gemaakt worden van een als... dan... constructie. Hiervoor wordt het element `<xsl:if>` gebruikt. Dit element heeft een attribuut `test`. Deze test bevat een expressie die waar of niet waar oplevert. De code binnen dit element wordt alleen uitgevoerd als de test waar is.

Voorbeeld:

```
<leden>
     <xsl:for-each select="/pers:personen/pers:persoon">
       <xsl:if test="pers:nationaliteit = 'Nederlands'">
         <naam>
           <xsl:value-of select="pers:voornaam"/>
           <xsl:value-of select="pers:achternaam"/>
         </naam>
       </xsl:if>
     </xsl:for-each>
</leden>
```

Als er meerdere opties zijn kunnen we dit oplossen door meerdere `<xsl:if>` elementen te combineren. Maar het is in dat geval eenvoudiger om het element `<xsl:choose>` te gebruiken. Een `<xsl:choose>` bevat één of meer `<xsl:when>` elementen. Ook een `<xsl:when>` heeft een attribuut `test`. Optioneel kan bij een `<xsl:choose>`een `<xsl:otherwise>` worden gebruikt.

Voorbeeld

```
<keuze>
     <kleur>
       <xsl:choose>
         <xsl:when test="pers:color ='blue'">blauw</xsl:when>
         <xsl:when test="pers:color ='red'">rood</xsl:when>
         <xsl:when test="pers:color ='green'">groen</xsl:when>
         <xsl:when test="pers:color ='yellow'">geel</xsl:when>
         <xsl:when test="pers:color ='orange'">oranje</xsl:when>
         <xsl:when test="pers:color ='purple'">paars</xsl:when>
         <xsl:otherwise>paars met groene stippen</xsl:otherwise>
       </xsl:choose>
     </kleur>
</keuze>
```

2.2.5 XPath

Bij de hiervoor besproken elementen is regelmatig gebruik gemaakt van een XPath expressie. Een XML document bestaat uit een boomstructuur van elementen (ook wel nodes genoemd). Om gegevens uit een XML document te halen zal de boomstructuur doorlopen moeten worden. Hiervoor kan een XPath expressie worden gebruikt. Met XPath kunnen we opgeven welk element uit de boomstructuur we willen benaderen. XPath beschrijft de weg door de boomstructuur.

Een XPath expressie begint in de boomstructuur bij de node die in het attribuut `match` van het element `<xsl:template>` is opgeven. Om aan te geven dat bij het root element moet worden gestart wordt `"/"` als waarde van `match` opgeven.

In de volgende tabel enkele voorbeelden van een XPath expressie.

Expressie	beschrijving
nodename	Selecteert alle nodes met deze naam op deze plek in het pad
/	Selecteert de root node. Merk op dat het eerste element daaronder dus *niet* de root node is.
//nodename	Selecteert alle nodes in het document die aan de opgegeven naam voldoen, op een willekeurige plek in de boomstructuur.
.	Selecteert de huidige node.
..	Selecteert de parent van de huidige node.
@attributename	Selecteert een attribuut met deze naam.
*	Een wildcard voor elke element node.
@*	Een wildcard voor elke attribuut node.
node()	Een willekeurige node, niet de root node of een attribuut
text()	Een tekst node

XPath expressies worden geëvalueerd ten opzichte van het huidige element. Zo heeft een template een *match* attribuut dat bepaalt welk element de context vormt voor onderliggende XPath expressies. Bij een for-each loop geeft het *select* attribuut aan wat de context is.

Stel bijvoorbeeld dat we de volgende template hebben:

```
<xsl:template match="persoon">
       <xsl:value-of select="./naam" />
</xsl:template>
```

In dit geval geeft xsl:template *match="persoon"* aan dat het element persoon de huidige node is. In xsl:value-of select="./naam" zien we de XPath expressie ./naam, waarbij de punt staat voor de huidige node. Met andere woorden: hier wordt verwezen naar de sub-node naam onder de node persoon. Hierbij is de puntnotatie voor de huidige node optioneel: een verkorte notatie voor "het element naam onder de huidige node" is de volgende:

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

In het volgende voorbeeld doorlopen we een aantal elementen met een for-each loop:

```
<xsl:template match="persoon">
       <xsl:for-each select="@*">
           <xsl:value-of select="." />
        </xsl:for-each>
</xsl:template>
```

Deze template heeft betrekking op een element persoon. Met `xsl:for-each select="@*"` worden alle attributen van dat element doorlopen. Van elk attribuut dat op die manier wordt gevonden wordt met `xsl:value-of select="."` de waarde weergegeven.

Hieronder volgt een lijst met voorbeelden van expressies

Voorbeeld pad expressie	**Resultaat**
bestelling/product[2]	Selecteert het tweede *product* dat een child node is van *bestelling*.
bestelling/product[last()]	Selecteert het laatste *product* dat een child node is van *bestelling*.
//bestelling/product[last()=1]	Selecteert het product van bestellingen met maar één product (het laatste product is tevens het eerste product).
bestelling/product[position() <= 2]	Selecteert de eerste twee producten (child nodes) van *bestelling*.
//product[2]/@kleur	Selecteert de waarde van het attribuut *kleur*, van het tweede *product*
//product[@kleur]	Selecteert alle nodes *product* met een attribuut *kleur*.
//product[@kleur='chroom']	Selecteert alle nodes *product* die een attribuut *kleur* met de waarde *'chroom'* hebben.
//product[@kleur='chroom']/omschrijving	Selecteert de node *omschrijving* van alle producten in de kleur *'chroom'*.
//product[prijs > 700]	Selecteert alle producten waarvan de *prijs* boven de 700 ligt

Merk op dat we geen < operator mogen gebruiken in XSLT. Dit teken geeft het begin van een tag aan. We gebruiken < in plaats van < als, dus *position() <= 2* is te lezen als "positie <= 2".

2.3 Meer XSLT elementen

De hierboven genoemde elementen zijn in XML introductie voor een groot deel al aan de orde geweest. Voor de volledigheid worden hieronder alle XSLT elementen in alfabetische volgorde genoemd. Een groot deel daarvan zullen we in de volgende paragrafen bespreken.

Element	Beschrijving
`apply-imports`	Past een template regel toe van een geïmporteerde stylesheet.
`apply-templates`	Past een template regel toe op het huidige element, of op de child elementen van het huidige element.
`attribute`	Voegt een attribuut toe aan een element.
`attribute-set`	Groepeert een set attributen, en geeft daar een naam aan.
`call-template`	Roept een template met een bepaalde naam aan.
`choose`	Wordt gebruikt in combinatie met <when> en <otherwise> om meerdere conditionele tests te formuleren.
`comment`	Maakt een commentaar element aan.
`copy`	Maakt een kopie van de huidige node, zonder child nodes en attributen.
`copy-of`	Maakt een kopie van de geselecteerde node, inclusief child nodes en attributen.
`decimal-format`	Bepaalt met welke karakters en symbolen een getal moet worden weergegeven, als dit getal naar string wordt geconverteerd met de format-number() functie.
`element`	Maakt een nieuw element aan.
`fallback`	Geeft alternatieve code aan om uit te voeren, als de processor een bepaald XSLT element niet herkent.
`for-each`	Doorloopt elke node in een opgegeven node set.
`if`	Bevat een template dat alleen zal worden toegepast als de meegegeven conditie geldig is.
`import`	Importeert de inhoud van een stylesheet in de huidige stylesheet. Template regels van de huidige stylesheet hebben voorrang op template regels van de geïmporteerde stylesheet.
`include`	Voegt de inhoud van een stylesheet toe aan het huidige stylesheet. De toegevoegde templates worden volgens dezelfde voorrangsregels geëvalueerd als de templates in de huidige stylesheet.
`key`	Declareert een key en kent daar een naam aan toe. De key kan verderop in de stylesheet worden gebruikt met de key() functie om snel bepaalde nodes op te zoeken (vergelijkbaar met een index).
`message`	Schrijft een boodschap naar de output (wordt gebruikt om fouten te rapporteren)
`namespace-alias`	Verandert een namespace in de stylesheet naar een andere namespace in de output.
`number`	Bepaalt de numerieke positie van de huidige node. Dit element wordt ook gebruikt om een getal in een bepaald formaat weer te geven.
`otherwise`	Geeft een default actie op voor het <choose> element.
`output`	Definieert het formaat van het output document.

Element	Beschrijving
`param`	Declareert een lokale of een globale parameter.
`preserve-space`	Definieert de elementen waarvoor whitespace moet worden behouden. Whitespace wordt default behouden: het gebruik van preserve-space heeft alleen zin in combinatie met een <strip-space> element (zie aldaar).
`processing-instruction`	Schrijft een processing instruction naar de output. Processing instructions zijn informatie voor de applicatie, die beginnen met <? en eindigen met ?> Een voorbeeld is de xml declaratie : `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`
`sort`	Sorteert de output.
`strip-space`	Geeft aan voor welke elementen de white space moet worden verwijderd. Uitzonderingen hierop kunnen met het element <preserve-space> worden gespecificeerd.
`stylesheet`	Het root element van een stylesheet.
`template`	Geeft aan wat er moet gebeuren als een gespecificeerde node wordt gevonden.
`text`	Schrijft tekst naar de output.
`transform`	Alternatief van <stylesheet>, met zelfde betekenis: het root element van een stylesheet.
`value-of`	Haalt de waarde op van de geselecteerde node.
`variable`	Declareert een lokale of globale variabele.
`when`	Geeft een actie bij een conditie van een <choose> element.
`with-param`	Geeft de waarde aan voor een parameter die aan een template wordt doorgegeven.

2.3.1 Tekst en whitespace

Voordat de eigenlijke transformatie plaatsvindt, zal de XML parser alle whitespace, en alle elementen die alleen whitespace bevatten (zoals tabs, spaties en returns), verwijderen. De voornaamste reden daarvoor is dat veel whitespace alleen bedoeld is om een mooie opmaak voor elkaar te krijgen, zoals het inspringen van child elementen. Voor een XSLT uitvoer zijn we meestal niet geïnteresseerd in de harde returns en tabs die nodig zijn voor "pretty print".

Toch kan het soms wenselijk zijn om whitespace te behouden of om deze toe te voegen. We kunnen daar het xsl:text element voor gebruiken.

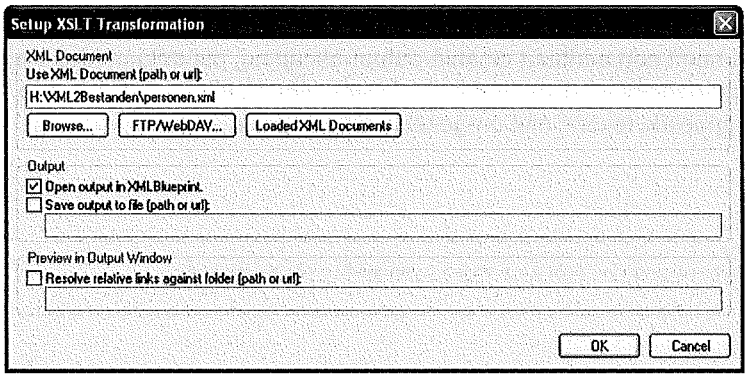
Bekijk het volgende voorbeeld:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:pers="http://www.pers.nl" exclude-result-prefixes="pers">

   <xsl:template match="/">

      <namen>
         <xsl:for-each select="/pers:personen/pers:persoon">
            <naam>
               <xsl:value-of select="pers:voornaam"></xsl:value-of>
               <xsl:value-of select="pers:achternaam"></xsl:value-of>
            </naam>
         </xsl:for-each>
      </namen>

   </xsl:template>
</xsl:stylesheet>
```

Deze stylesheet zet de namen van personen in een nieuw document. De voornaam en de achternaam worden steeds gecombineerd.

In XML Blueprint kunnen de stylesheet en het XML document onder meer op de volgende wijze worden gekoppeld:

Open het document en de stylesheet. Klik bij het geopende stylesheet op de knop **Setup XSLT Transformation**  . Het gelijknamige venster verschijnt. Klik de knop Loaded XML Documents en kies het geopende document.



Bevestig met **OK**. De transformatie kunnen we uitvoeren met de knop **Run XSLT Transformation**  .

Het resultaat van de transformatie levert wel de juiste informatie op, maar er moet een spatie tussen de voor- en achternaam komen. Dit kan niet door simpel een spatie in het xsl document te typen. Hiervoor gebruiken we het element xsl:text. Dit element is bedoeld om letterlijke tekst toe te voegen.

In het volgende voorbeeld wordt een spatie toegevoegd tussen voornaam en achternaam, door die spatie binnen het <xsl:text> element mee te geven:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform
   xmlns:pers="http://www.pers.nl" exclude-result-prefixes="pers">

   <xsl:template match="/">

      <namen>
         <xsl:for-each select="/pers:personen/pers:persoon">
            <naam>
               <xsl:value-of select="pers:voornaam"></xsl:value-of>
               <xsl:text> </xsl:text>
               <xsl:value-of select="pers:achternaam"></xsl:value-of>
            </naam>
         </xsl:for-each>
      </namen>

   </xsl:template>
</xsl:stylesheet>
```

Op dezelfde manier kunnen we ook harde returns toevoegen. Bekijk bijvoorbeeld wat er gebeurt als er een harde return in plaats van een spatie wordt geplaatst tussen <text> en </text>.

Meer controle over whitespace is te bereiken met de functie `normalize-space()`. Deze functie zal verderop in dit hoofdstuk worden besproken.

Een andere toepassing van het text-element is het weergeven van speciale karakters, zoals &, < en >. Deze karakters kunnen niet zomaar in XML worden gebruikt, omdat ze een speciale betekenis hebben, en worden standaard in de uitvoer als character entities weergegeven, in dit geval & en < en >.

Het element xsl:text heeft een attribuut disable-output-escaping, die default (dus als dit attribuut niet wordt meegegeven) de waarden "no" heeft. Als dit attribuut de waarde "yes" krijgt, worden deze speciale tekens niet omgezet in character entities.

Voorbeeld:

```
<xsl:template match="/">
   <xsl:text disable-output-escaping="no">Als a > b en b > c dan geldt: c &lt; a </xsl:text>
</xsl:template>
```

Uitvoer:

```
Als a &gt; b en b &gt. c, dan geldt c &lt; a.
```

Merk op dat in de stylesheet het teken < als < moet worden meegegeven.

Als we hier `disable-output-escaping="yes"` van maken, dan krijgen we netjes de volgende uitvoer:

```
Als a > b en b > c dan geldt: c < a
```

Overigens zijn de verdere mogelijkheden van xsl:text beperkt, omdat er alleen letterlijke tekst aan kan worden meegegeven. Het volgende kan dus bijvoorbeeld niet:

```
<xsl:template match="/">
   <xsl:text><xsl:value-of select="." /></xsl:text>
</xsl:template>
```

omdat er geen andere elementen binnen xsl:text zijn toegestaan.

2.3.2 Element

In de transformaties die we tot nu toe gezien hebben worden nieuwe element steeds als nieuwe tag toegevoegd. Als we het element naam nodig hebben wordt er ergens in de xsl `<naam>` en `</naam>` opgenomen. Dit kan ook met behulp van het element `<xsl:element>`. De naam van het element wordt aangeven bij het attribuut `name`.

We kunnen bijvoorbeeld de code

```
<naam>
    <xsl:value-of select="pers:voornaam"/>
</naam>
```

vervangen door het volgende code.

```
<xsl:element name="naam">
    <xsl:value-of select="pers:voornaam"/>
</xsl:element >
```

In het bovenstaande voorbeeld zien we verschillende manieren om een element te definiëren. Het resultaat zal gelijk zijn. Toch zit er wel verschil in. Met een xsl:element kunnen we de stylesheet dynamischer maken. Bekijk hiervoor het volgende voorbeeld:

element.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
        <xsl:apply-templates select="groepen" />
    </xsl:template>

    <xsl:template match="groepen">
        <personen>
            <xsl:for-each select="groep">
                <xsl:element name="{mv}">
                    <xsl:for-each select="personen/persoon">
                        <xsl:copy-of select="naam"/>
                    </xsl:for-each>
                </xsl:element>
            </xsl:for-each>
        </personen>
    </xsl:template>
</xsl:stylesheet>
```

element.xml

```
<?xml version="1.0" encoding="utf-8"?>

<groepen>
    <groep>
        <mv>jongens</mv>
        <personen>
            <persoon>
                <naam>Willem Veenstra</naam>
            </persoon>
            <persoon>
                <naam>Vincent Schouten</naam>
            </persoon>
        </personen>
    </groep>
    <groep>
        <mv>meisjes</mv>
        <personen>
            <persoon>
                <naam>Joleen Dijkstra</naam>
            </persoon>
        </personen>
    </groep>
</groepen>
```

In de stylesheet vinden we `<xsl:element name="{mv}">`. Een attribuut kan dynamisch een waarde krijgen door deze waarde met een XPath expressie op te halen. Deze XPath expressie moet dan tussen accolades worden meegegeven. In dit geval wordt dus de waarde van het *mv* element gelezen en als naam aan het nieuwe element toegekend. De uitvoer wordt dan als volgt.

```
<?xml version="1.0" encoding="UTF-16"?>
<personen>
    <jongens>
        <naam>Willem Veenstra</naam>
        <naam>Vincent Schouten</naam>
    </jongens>
    <meisjes>
        <naam>Joleen Dijkstra</naam>
    </meisjes>
</personen>
```

We zien dat de elementen `<jongens>` en `<meisjes>` zijn toegevoegd.

2.3.3 Attributen

Als we door een transformatie aan een element attributen willen toevoegen, kunnen we direct onder dit element `<xsl:attribute>` gebruiken. Het element `<xsl:attribute>` heeft zelf een verplicht attribuut `name` en een optioneel attribuut `namespace`. Bij het attribuut `name` kan de naam voor het attribuut worden opgeven. De waarde van het attribuut wordt opgenomen tussen de begintag en de eindtag van `<xsl:attribute>`.

In het volgende voorbeeld gaan we de geboortedatum als attribuut aan het element naam toevoegen.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:pers="http://www.pers.nl" exclude-result-prefixes="pers">

   <xsl:template match="/">

      <namen>
         <xsl:for-each select="/pers:personen/pers:persoon">
            <xsl:element name="naam">
               <xsl:attribute name="gebdat">
                  <xsl:value-of select="pers:geboortedatum"/>
               </xsl:attribute>
               <xsl:value-of select="pers:voornaam"></xsl:value-of>
               <xsl:text> </xsl:text>
               <xsl:value-of select="pers:achternaam"></xsl:value-of>
            </xsl:element>
         </xsl:for-each>
      </namen>

   </xsl:template>
</xsl:stylesheet>
```

Na de transformatie zijn we bij elke naam het attribuut `gebdat` terug. De datum zelf wordt met behulp van `<xsl:value-of>` en een XPath Expressie opgehaald.

Een tweede attribuut kan onder het eerste attribuut worden opgenomen.

Met het element `xsl:attribute` kan de naam van een attribuut dynamisch gegenereerd worden. Dit hebben we in de vorige paragraaf ook gezien bij het element `xsl:element`. In het bovenstaande voorbeeld hebben we de elementen `xsl:attribute` en `xsl:value-of` gebruikt om een attribuut een waarde te geven. Dit kan weer eenvoudiger door gebruik te maken van { }. Bekijk hiervoor het volgende voorbeeld.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:pers="http://www.pers.nl" exclude-result-prefixes="pers">
   <xsl:template match="/">

      <personen>
         <xsl:for-each select="/pers:personen/pers:persoon">
            <naam gebdat="{pers:geboortedatum}" oogkleur="{pers:ogen}">

               <xsl:value-of select="pers:voornaam"></xsl:value-of>
               <xsl:text> </xsl:text>
               <xsl:value-of select="pers:achternaam"></xsl:value-of>
            </naam>
            <xsl:copy-of select="pers:adres"/>
         </xsl:for-each>
      </personen>

   </xsl:template>
</xsl:stylesheet>
```

We zien dat het element naam is toegevoegd met twee attributen. Als waarde voor het attribuut gebdat zien we {pers:geboortedatum}. Hier wordt voor elke persoon de bijbehorende waarde van geboortedatum gebruikt om het attribuut een waarde te geven.

2.3.4 Attribuut set

Als een groep van attributen vaker gebruikt moet worden, kunnen deze attributen als groep worden gedefinieerd met het element `<xsl:attribute-set>`. Binnen dit element nemen we een lijst van attributen op met `<xsl:attribute>`.

```
<xsl:attribute-set name="extra">
  <xsl:attribute name="gebdat">
     <xsl:value-of select="pers:geboortedatum"/>
   </xsl:attribute>
   <xsl:attribute name="oogkleur">
      <xsl:value-of select="pers:ogen"/>
   </xsl:attribute>
</xsl:attribute-set>
```

Een attribute-set kan alleen gedefinieerd worden als child element van `<xsl:stylesheet>` of `<xsl:transform>`.

Om deze attributen aan een element toe te voegen verwijzen we naar deze groep met het attribuut `xsl:use-attribute-sets`. Als waarde geven we de naam van de attribuut groep.

```
<xsl:element name="naam" use-attribute-sets="extra">...</xsl:element>
```

Of

```
<naam use-attribute-sets="extra">...</naam>
```

Een element waaraan attributen zijn toegekend met `use-attribute-sets="..."` kan eventueel verder worden uitgebreid door daaronder `<xsl-attribute name="...">` elementen toe te voegen.

2.3.5 copy en copy-of

Als we bij een transformatie een groot deel van een xml document in zijn geheel willen overnemen in een nieuwe xml document kan het element `<xsl:copy-of>` worden gebruikt. Dit element zorgt er voor dat de huidige node inclusief child nodes wordt overgenomen. Bij het attribuut select wordt aangegeven om welke node het gaat.

Om een node zonder childnodes over te nemen kan het element `<xsl:copy>` worden gebruikt. Dit element zorgt er voor dat alleen huidige node zonder childnodes en zonder inhoud wordt over genomen in de nieuwe xml. Ook hier geldt weer dat de originele namespace wordt overgenomen. In tegenstelling tot `<xsl:copy-of>` heeft het element `<xsl:copy>` geen attribuut `select`. Er moet dus vooraf gezorgd worden dat de node die we willen gebruiken de huidige node is.

Een veel gebruikte toepassing van xsl:copy en xsl:copy-of is *identity transform:* een stylesheet die een letterlijk kopie maakt van het origineel. Hieronder staat een voorbeeld van zo'n transformation.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Hier staat dat alles onder het root element moet worden gekopieerd, inclusief dat root element zelf. Als het inderdaad de bedoeling is om het hele document te kopiëren, is dit een snelle manier.

Een meer gebruikt template maakt het mogelijk om uitzonderingen te maken. Dit is een recursief template die zichzelf aanroept: binnen elk gecopieerd element wordt met xsl:apply-templates gekeken of er nog andere elementen of attributen zijn. Als die er zijn, worden die ook gecopieerd.

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="@*|node()">
  <xsl:copy>
   <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

Deze template is van toepassing voor alle nodes (tekst nodes, elementen, processing-instructions, commentaar) en attributen (match="@*|node()"). Daarbinnen wordt met xsl:copy het betreffende element of attribuut gecopieerd. Met `<xsl:apply-templates select="@*|node()" />` wordt bedoeld: doe nu hetzelfde voor alle sub-nodes en attributen. In een volgende paragraaf zullen we het gebruik van templates nader toelichten.

In deze template staat dus dat in het algemeen nodes en attributen zullen worden gekopieerd. Door templates toe te voegen kunnen we hier uitzonderingen op maken. We zullen dat doen in het volgende voorbeeld.

Als we bijvoorbeeld een document werknemers.xml transformeren met deze template, dan zal de structuur er vrijwel hetzelfde uitzien als het origineel. Voegen we onder de bestaande template de volgende template toe aan de stylesheet, dan zullen de elementen salaris en toeslag niet worden opgenomen in het resultaat.

```
<xsl:template match="salaris|toeslag"></xsl:template>
```

In het voorgaande voorbeeld gebruikten we twee templates: één voor nodes en attributes in het algemeen, waarvoor geldt dat ze gekopieerd moeten worden, en één voor de elementen salaris en toeslag, waar niets mee moet gebeuren. Deze laatste template is specifieker dan de eerste template. Deze laatste template krijgt daarom voorrang.

2.3.6 Opmaak van getallen

XSLT kent een aantal elementen om de weergave van getallen te manipuleren. We bespreken hier `<xsl:number>` en `<xsl:decimal-format>`. Het element `<xsl:number>` kan gebruikt worden om een getal een opmaak te geven, of om de integer positie van een element op te halen. Bij de volgende voorbeelden zullen de bestanden *producten.xml* en *producten.xsl* gebruikt worden.

producten.xml

```
<?xml version="1.0" encoding="utf-8"?>
<producten xmlns="www.producten.nl">
    <product>
        <id>32</id>
        <soort>schoen</soort>
        <prijs>34.50</prijs>
        <aantal>20800</aantal>
    </product>
    <product>
        <id>563</id>
        <soort>slipper</soort>
        <prijs>12.95</prijs>
        <aantal>870500</aantal>
    </product>
    <product>
        <id>2</id>
        <soort>klomp</soort>
        <prijs>15.50</prijs>
        <aantal>4500</aantal>
    </product>
</producten>
```

producten.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:pro="www.producten.nl"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
    <xsl:template match="/">
        <producten>
            <xsl:for-each select="pro:producten/pro:product">
                <product>
                    <productid> <xsl:value-of select="pro:id"/></productid>
                    <soort><xsl:value-of select="pro:soort"/></soort>
                    <aantal><xsl:value-of select="pro:aantal"/></aantal>
                    <prijs><xsl:value-of select="pro:prijs"/></prijs>
                </product>
            </xsl:for-each>
        </producten>
    </xsl:template>
</xsl:stylesheet>
```

We zouden hier ook gebruik kunnen maken van identity.xsl. We komen verderop in dit hoofdstuk op deze stylesheet terug.

We willen dat het productid altijd uit 3 cijfers bestaat. Bij een productid van minder dan 3 cijfers moeten voorloopnullen worden toegevoegd.

Om dit te realiseren veranderen we de code bij productid.

```
<productid> <xsl:number format="001" value="pro:id"/></productid>
```

Het het attribuut `value` geven we aan welk getal moet worden opgemaakt. Dit kan een vast getal zijn of het resultaat van een XPath expressie. Wordt het attribuut `value` weggelaten, dan wordt een nummer gegenereerd, afhankelijk van de positie van het element: het eerste element krijgt de waarde 1, het tweede 2 enz.

Bij het attribuut `format` kunnen we kiezen uit een aantal mogelijkheden:

Voorbeeld pad expressie	**Resultaat**
format="1"	Gewone getallen 1 2 3 . .enz.
format="01"	Getallen worden voorzien van voorloop nullen 01 02 03. Voor elk aantal extra voorloop nullen een extra 0 in het format opnemen.
format="a"	Getallen worden omgezet naar letters. 1 wordt a, 2 wordt b, 3 wordt c, 26 wordt z, 27 wordt aa, 28 wordt ab enz.
format="A"	Gelijk aan format="a" maar dan in hoofdletters.
format="i"	Geeft een getal weer in Romeinscijfers i ii iii iv (kleine letters)
format="I"	Geeft een getal weer in Romeinscijfers I II III IV (hoofdletters)

Ook het verdelen van getallen in groepen van drie (zoals duizend- en milioen-tallen) kunnen we regelen met een `<xsl:number>` element. Hiervoor gebruiken we de attributen `grouping-size` en `grouping-separator`.

We passen in de stylesheet de inhoud van het element `<aantal>` als volgt aan:

```
<aantal>
  <xsl:number grouping-separator="." grouping-size="3"
              value="pro:aantal"/>
</aantal>
```

Na transformatie zal het resultaat er als volgt uitzien:

```
<?xml version="1.0" encoding="UTF-16"?>
<producten xmlns:pro="www.producten.nl">
   <product>
      <productid>032</productid>
      <soort>schoen</soort>
      <aantal>20.800</aantal>
      <prijs>34.50</prijs>
   </product>
   <product>
      <productid>563</productid>
      <soort>slipper</soort>
      <aantal>1.280.500</aantal>
      <prijs>12.95</prijs>
   </product>
   <product>
      <productid>002</productid>
      <soort>klomp</soort>
      <aantal>4.500</aantal>
      <prijs>15.50</prijs>
   </product>
</producten>
```

Met `<xsl:number>` kan alleen de opmaak van gehele getallen worden gedefinieerd. Willen we meer mogelijkheden dan kunnen we de XSLT functie `format-number()` gebruiken.
We bekijken eerst een voorbeeld:

```
<prijs><xsl:value-of select="format-number(pro:prijs, '€ #.00')"/></prijs>
```

Dit geeft als resultaat:

```
<prijs>€ 34.50</prijs>
```

We zien dat de functie gebruikt wordt in het `select` attribuut van `<xsl:value-of>`. De functie heeft twee argumenten: een waarde en een opmaak. Bij de waarde staat in dit geval een XPath expressie. Bij de opmaak wordt gebruik gemaakt van een masker. Het masker staat tussen single quotes.

We kunnen onder meer de volgende karakters gebruiken in het masker van format-number():

Symbool	Genereert	Voorbeeld	Invoer	Uitvoer
#	een cijfer [0-9]	###	4.2	4
.	decimaal scheidingsteken	###.##	4.2	4.2
,	scheidingsteken tussen groepen	###,###.##	1234.56	1,234.56
0	voorloopnullen, en eindnullen	000.00	4.3	004.30
%	procentteken	##%	25	25%
;	scheidingsteken tussen patroon voor negatieve en voor positieve getallen. Het tweede patroon geeft de weergave van het negatieve getal. Default wordt een negatief getal met een - ervoor weergegeven.	##.00;(##.00)	-3.2	(3.20)

Willen we ook de duizendtallen gegroepeerd hebben, dan zal het masker er dus als volgt uitzien:

```
<prijs><xsl:value-of select="format-number(pro:prijs, '€ #,###.00')"/></prijs>
```

De komma staat hier voor het groeperingsteken voor duizendtallen.

In het masker kan verder onderscheid gemaakt worden tussen een notatie voor positieve en negatieve getallen. Hiervoor worden twee maskers opgenomen gescheiden door een puntkomma. Het eerste masker wordt gebruikt voor positieve getallen en het tweede masker voor negatieve getallen. Ook kan een getal als percentage of promille worden geschreven door % of het unicode karakter voor promile #x2030 (‰) in het masker op te nemen.

In het bovenstaande voorbeeld hebben we gezien dat een punt als decimaalscheiding symbool wordt gebruikt. Willen we hier een komma gebruiken dan is het niet voldoende om de punt te vervangen door een komma. Er moet dan eerst aangegeven worden wat de decimale opmaak is. Dit kan met het element `<xsl:decimal-format>` . Dit element kan niet binnen een `<xsl:template>` gebruikt worden maar wordt erboven gedefinieerd. Bij dit element kunnen de volgende attributen worden gebruikt:

attribuut	beschrijving
name	Definieert een naam aan de gedefinieerde opmaak.
decimal-separator	Definieert het teken dat als decimaalscheiding symbool wordt gebruikt. (Standaard is dit '.')
grouping-separator	Definieert het teken dat als duizendtallenscheiding symbool wordt gebruikt. (Standaard is dit ',')
infinity	Definieert de tekst die voor een oneindige waarde wordt gebruikt. (Standaard is dit 'infinity')

attribuut	beschrijving
minus-sign	Definieert het teken dat als symbool voor een negatief waarde wordt gebruikt. (Standaard is dit '-')
NaN	Definieert de tekst die wordt gebruikt als de waarde geen getal is. (Standaard is dit 'NaN')
percent	Definieert het teken dat wordt gebruikt als symbool voor een percentage. (Standaard is dit '%')
per-mille	Definieert het teken dat wordt gebruikt als symbool voor een percentage. (Standaard is dit '‰')
digit	Definieert het teken dat wordt gebruikt als symbool voor een optioneel getal. (Standaard is dit '#')
zero-digit	Definieert het teken dat wordt gebruikt als symbool voor een getal. (Standaard is dit '0')
pattern-separator	Definieert het teken dat wordt gebruikt als symbool voor scheiding tussen maskers van negatieve en positieve getallen. (Standaard is dit ';')

Met het element `<xsl:decimal-format>` kunnen we de standaard waarden die de functie `format-number()` gebruikt wijzigen.

```
<xsl:decimal-format decimal-separator="," grouping-separator=".">
```

Nu kunnen we de opmaak van de prijs wel als volgt definiëren:

```
<prijs><xsl:value-of select="format-number(pro:prijs, '€ #,###.00')"/></prijs>
```

Als we meerdere decimale notaties in een document gebruiken dan kunnen we het attribuut `name` van `<xsl:decimal-format>` gebruiken. In de functie `format-number()` kunnen we dan een derde argument toevoegen met de naam van de gewenste notatie.

Bijvoorbeeld:

```
<xsl:decimal-format name="euro" decimal-separator="," grouping-separator="." />

<xsl:template match="/">
<xsl:value-of select="format-number(143563.3, '#.###,00', 'euro')"/>
</xsl:template>
```

2.3.7 Nummeren van elementen

Het element <xsl:number> kan niet alleen gebruikt worden om gehele getallen op te maken, maar ook om nummers te genereren. Een veel gebruikte manier is de volgende:

```
<xsl:number value="position()">
```

Hierbij wordt voor elk volgende element die matcht met de template een volgend nummer gegenereerd, ongeacht de positie in de hierarchie.

Het volgende voorbeeld toont een boomstructuur van hoofdstukken en paragrafen van een boek.

```
<?xml version="1.0" encoding="utf-8"?>
<boek>
   <hoofdstuk>
      <paragraaf>
         <paragraaf></paragraaf>
      </paragraaf>
      <paragraaf>
         <paragraaf></paragraaf>
      </paragraaf>
   </hoofdstuk>
   <hoofdstuk>
      <paragraaf>
         <paragraaf>
            <paragraaf></paragraaf>
         </paragraaf>
      </paragraaf>
      <paragraaf>
         <paragraaf></paragraaf>
      </paragraaf>
   </hoofdstuk>
   <hoofdstuk>
      <paragraaf>
      </paragraaf>
      <paragraaf>
         <paragraaf></paragraaf>
      </paragraaf>
   </hoofdstuk>
</boek>
```

Met de volgende stylesheet wordt voor elk hoofdstuk of paragraaf een element met dezelfde naam gegenereerd met de volgende truc: `<xsl:element name="{name()}">`. Binnen elk element wordt een volgnummer gegenereerd met `<xsl:number value="position()">`.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <inhoud>
         <xsl:apply-templates select=".//hoofdstuk|.//paragraaf" />
      </inhoud>
   </xsl:template>
   <xsl:template match="*">
      <xsl:element name="{name()}">
         <xsl:number value="position()"/>
      </xsl:element>
   </xsl:template>
</xsl:stylesheet>
```

Voer de transformatie uit en bekijk het resultaat.

We zien hier dat alle elementen worden doorgenummerd. Hoewel deze toepassing vaak handig is, geeft het hier niet het gewenste resultaat:

```
<inhoud>
   <hoofdstuk>1</hoofdstuk>
   <paragraaf>2</paragraaf>
   <paragraaf>3</paragraaf>
   <paragraaf>4</paragraaf>
   <paragraaf>5</paragraaf>
   <hoofdstuk>6</hoofdstuk>
   <paragraaf>7</paragraaf>
   <paragraaf>8</paragraaf>
   <paragraaf>9</paragraaf>
   <paragraaf>10</paragraaf>
   <paragraaf>11</paragraaf>
   <hoofdstuk>12</hoofdstuk>
   <paragraaf>13</paragraaf>
   <paragraaf>14</paragraaf>
   <paragraaf>15</paragraaf>
</inhoud>
```

In bovenstaand voorbeeld is niet xsl:number, maar de functie position() verantwoordelijk voor het gegenereerde volgnummer. Zonder het value="..." attribuut genereert <xsl:number> zelf een nummer. Wat voor nummer wordt gegenereerd wordt door enkele optionele attributen bepaald.

attribuut	uitleg	voorbeeld
`count="patroon"`	Geeft aan welke elementen of attributen worden meegenomen bij het nummeren. De default is om elementen te tellen van de huidige node.	`count="hoofdstuk|paragraaf"`
`from="patroon"`	Geeft aan vanaf welk element of attribuut moet worden geteld. Default worden ze allemaal meegenomen.	`from="hoofdstuk"`
`level="single"` of `level="multiple"` of `level="any"`	De default is single: dan worden alle voorgaande elementen die overeenkomen met de node van de template tot de huidige node geteld. Bij level="multiple" wordt eerst een lijst gemaakt van bovenliggende nodes, om op basis daarvan een hierarchische nummering mogelijk te maken. Bij level="any" wordt vanaf de huidige node geteld hoeveel nodes voldoen aan count="patroon".	`level="multiple"`
`format="string"`	De string geeft aan hoe het gegenereerde nummer moet worden weergegeven, zie hieronder.	`format="1.a"`

We bespreken nu enkele alternatieve gebruiken van <xsl:number>.

```
<xsl:number>
```

Zoals gezegd wordt nu level="single" aangehouden. Elementen van hetzelfde niveau in boomstructuur worden doorgeteld. Dus elk hoofdstuk krijgt een volgend nummer, maar niet elke paragraaf: sommige paragrafen zijn genest, en voor zo'n nieuw niveau begint het tellen opnieuw.

```
<xsl:number level="any" >
```

Hierbij wordt bij elke node gekeken hoeveel voorgaande nodes er zijn van dat type, ongeacht het niveau in de hiërarchie. Zo ontstaan hoofdstuk 1 t/m 5 en paragraaf 1 t/m 12.

```
<xsl:number level="multiple" >
```

Nu zien we de hiërarchie terug in de nummering: voor elk level van de paragrafen wordt een extra nummer gegenereerd, zoals een paragraaf 2.1 en een paragraaf 1.1.1. Als we de hoofdstukken mee willen nemen in die nummering kunnen count="hoofdstuk|paragraaf" meegeven. Normaal wordt een volgnummer aangemaakt voor het type van de huidige node, dus òf voor een paragraaf, òf voor een hoofdstuk. Met count="hoofdstuk|paragraaf" geven we aan dat beide elementen meedoen bij het berekenen van het volgnummer.

```
<xsl:number level="multiple" count="hoofdstuk|paragraaf"  format="1.a">
```

Merk op dat de hoofdstukken nu genummerd zijn, en dat de paragrafen worden aangeduid met letters. De formaten die kunnen worden meegegeven met format="*string*" zijn genoemd in paragraaf **Fout! Verwijzingsbron niet gevonden.**.

2.3.8 Variabelen

Soms is het nodig om een opgehaalde waarde te bewaren in een variabele, om hier op een ander punt van de verwerking naar te kunnen verwijzen. Hiervoor is binnen XSLT het element `<xsl:variable>` beschikbaar. Dit element heeft een attribuut `name`. Deze gebruiken we als we naar de huidige waarde van de variabele willen verwijzen. Het verwijzen naar een variabele doen we door `$variabelenaam` in de code op te nemen.

Een veelgebruikte toepassing van variabelen is het koppelen van twee delen van een document.

Stel dat we in een XML document een lijst hebben met lijst cd's en daaronder een lijst tracks. De tracks staan in dit geval dus los van de cd's: er is geen parent-child relatie tussen beide lijsten. De tracks hebben wel een attribuut cd, die verwijst naar een attribuut id van een cd. In het volgende voorbeeld genereren we hieruit een lijst cd's, met binnen elke cd een lijst tracks van die cd.

Bekijk de code van de volgende stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
      <cds>
      <xsl:for-each select="//cd">
         <xsl:variable name="id" select="@id"/>
         <cd titel="{cdtitel}">
            <xsl:for-each select="//track[@cd=$id]">
               <track titel="{titel}" />
            </xsl:for-each>
         </cd>
      </xsl:for-each>
      </cds>
   </xsl:template>
</xsl:stylesheet>
```

We doorlopen hier eerst alle cd's met <xsl:for-each> en maken een cd aan. Daarna moeten we de tracks van die cd zien te vinden. Zonder variabele zal dit niet gaan. We kunnen bijvoorbeeld niet zeggen `<xsl:for-each select ="track[@cd=../@id]" />` of iets dergelijks, want de cd's staan in cdlijst3.xml los van de tracks. Er is met andere woorden geen parent-child relatie tussen de cd's en de tracks. Wat we wel kunnen doen is elke cd doorlopen, daarvan het id bewaren in een variabele, en vervolgens alle tracks opzoeken met het betreffende id.

Vandaar dat we binnen `<xsl:for-each select="//cd" >` eerst een variabele aanmaken:

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

Later kunnen we dan - ergens in het document, buiten de parent-child relaties om - op zoek gaan naar de tracks van deze cd. Dat gebeurt met de volgende regel:

```
<xsl:for-each select="//track[@cd=$id]" >
```

Hier is $id de verwijzing naar onze variabele, met de naam id.

Als een XSLT variabele is aangemaakt en gevuld, kan de waarde ervan niet meer gewijzigd worden. Wel kan bijvoorbeeld binnen een for-each loop voor elk element dat wordt vewerkt opnieuw een variabele met dezelfde naam worden aangemaakt en gevuld.

2.4 Templates

XSLT kent twee benaderingen om een XML document te verwerken. Tot nu toe hebben we vooral gewerkt met xsl:for-each om delen uit een document op te halen, en daar iets mee te doen. Dit wordt wel de "pull"-benadering genoemd. De stylesheet trekt als het ware de benodigde elementen en attributen uit het document, waarna ze bewerkt kunnen worden. Een voordeel van deze benadering is dat het de code vereenvoudigt: je hebt relatief weinig templates nodig, en de for-each loop is voor de meeste ontwikkelaars een vertrouwde manier van programmeren. Een nadeel is dat deze manier van werken weinig flexibel is: alleen de elementen en attributen die je zelf uit het document trekt worden verwerkt, eventuele andere elementen blijven buiten beschouwing.

Tegenover pull- technologie staat de push technologie. Hierbij worden verschillende templates gedefinieerd. Elke template representeert een aantal regels, die van toepassing zijn op bepaalde elementen of attributen. Het document wordt vervolgens als het ware door de stylesheet heen geduwd. De templates vangen de elementen op, doen daar wat mee, en geven de rest van het document door om eventueel door andere templates te worden verwerkt.
Deze benadering is voor ontwikkelaars vaak even wennen, maar biedt wel meer flexibiliteit dan de pull-benadering. We zullen dit in de komende paragrafen laten zien.

2.4.1 Default templates

De push benadering gaat uit van twee bouwstenen:

1. templates met een match="..." attribuut, waarin staat voor welke elementen en/of attributen deze templates gelden

2. het aanroepen van volgende templates met <xsl:apply-templates>

Het volgende voorbeeld laat dit in de meest simpele vorm zien:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

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

</xsl:stylesheet>
```

Deze stylesheet heeft één template. Zodra de root gevonden wordt, doen we daar niks mee, maar passen we alle bestaande templates toe voor de volgende elementen of attributen die we vinden. Merk op dat er geen andere templates zijn. Deze stylesheet zal dus geen uitvoer geven.

Stel dat we deze stylesheet gebruiken op een XML document, dan krijgen we toch uitvoer. De waarden van de text nodes worden getoond:

```
    Alex
    Velten

      Perkstraat 5
      Hogerdam

    Nederlands
    18-11-1993
    blauw


[...]


    Belgisch
    29-01-1999
    blauw
```

De XSLT processor maakt gebruik van twee default templates, voor het geval er geen andere templates zijn gedefinieerd. Deze hebben de volgende code:

```
<xsl:template match="*|/">
   <xsl:apply-templates />
</xsl:template>

<xsl:template match="text()|@*">
   <xsl:value-of select="." />
</xsl:template>
```

In de eerste template staat: als je de root of een element tegenkomt, voer dan op alle child nodes de bestaande templates uit. De code `<xsl:apply-templates/>`, zonder select attribuut, zorgt ervoor dat de templates voor sub-elementen en voor text-nodes worden uitgevoerd. De template voor attributen wordt hierbij dus nooit bereikt.

Voor sub-elementen wordt dezelfde template nog een keer uitgevoerd. Immers: `match="*|/"` geldt voor de root, en alle andere elementen, dus ook sub-elementen.

Zodra een tekst-node wordt gevonden, dan gaat de tweede default template in werking: `match="text()|@*"`. Met xsl:value-of wordt hiervan de waarde naar de uitvoer weggeschreven. Default worden dus geen attributen gevonden, dat gebeurt alleen als er met select in `<xsl:apply-templates/>` naar attributen wordt verwezen.

Stel bijvoorbeeld dat we alleen een stylesheet hebben waarin het volgende staat, dan zullen ook attribuut-waarden worden weergegeven:

```
<xsl:template match="*">
  <xsl:apply-templates select="*|text()|@*"/>
</xsl:template>
```

Kortom: zonder templates treden de default templates in werking, en zullen alleen de tekstwaarden van de elementen worden weergegeven.

2.4.2 Van pull naar push

De kracht van de push benadering wordt vooral duidelijk als een klein deel van een XML document gewijzigd moet worden. Bekijk bijvoorbeeld nog eens de code van personen.xml:

```
<personen xmlns="http://www.pers.nl">
  <persoon>
    <voornaam>Alex</voornaam>
    <achternaam>Velten</achternaam>
    <adres>
      <straat>Perkstraat 5</straat>
      <plaats>Hogerdam</plaats>
    </adres>
    <nationaliteit>Nederlands</nationaliteit>
    <geboortedatum>18-11-1993</geboortedatum>
    <ogen>blauw</ogen>
  </persoon>
  <persoon>
    <voornaam>Simon</voornaam>
    <achternaam>Scholten</achternaam>
    <adres>
      <straat>Tuinstraat 54</straat>
      <plaats>Hogerdam</plaats>
    </adres>
    <nationaliteit>Nederlands</nationaliteit>
    <geboortedatum>26-04-1995</geboortedatum>
    <ogen>bruin</ogen>
  </persoon>
  <persoon>
    <voornaam>Marith</voornaam>
    <achternaam>Boersma</achternaam>
    <adres>
      <straat>Grasstraat 18</straat>
      <plaats>Hogerdam</plaats>
    </adres>
    <nationaliteit>Belgisch</nationaliteit>
    <geboortedatum>29-01-1999</geboortedatum>
    <ogen>blauw</ogen>
  </persoon>
</personen>
```

Stel dat we hierin het element adres willen veranderen, zodat deze alleen de tekst bevat van het adres, zonder de sub-elementen straat en plaats. Om dat met de pull benadering voor elkaar te krijgen moeten we precies weten welk element waar te verwachten is. Dit kan bijvoorbeeld met de volgende stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:pers="http://www.pers.nl" exclude-result-prefixes="pers" xmlns="http://www.pers.nl">
    <xsl:template match="/">
        <personen>
            <xsl:for-each select="/pers:personen/pers:persoon">
                <persoon>
                    <xsl:copy-of select="pers:voornaam" />
                    <xsl:copy-of select="pers:achternaam" />
                    <adres>
                    <xsl:value-of select="pers:adres/pers:straat" />, <xsl:value-of select="pers:adres/pers:plaats" />
                    </adres>
                    <xsl:copy-of select="pers:nationaliteit" />
                    <xsl:copy-of select="pers:geboortedatum" />
                    <xsl:copy-of select="pers:ogen" />
                </persoon>
            </xsl:for-each>
        </personen>
    </xsl:template>
</xsl:stylesheet>
```

Merk op dat hier handmatig de hele structuur van het bron-document is overgenomen, met als uitzondering het element adres. Vooral voor grotere documenten is zo'n kleine wijziging dus nogal bewerkelijk. Daarnaast moet op deze manier van te voren heel precies bekend zijn waar welk element te vinden is in het bron document. Stel dat in het bron document later extra elementen worden opgenomen, zoals <telefoon> of <emailadres>, dan zal de stylesheet ook weer moeten worden bijgewerkt.

Merk verder op dat in <xsl:stylesheet> enkele attributen zijn opgenomen, om ervoor te zorgen dat de namespace http://www.pers.nl correct in de uitvoer terecht komt.

Om in de stylesheet elementen van die namespace op te kunnen halen, hebben we een prefix nodig: `xmlns:pers="http://www.pers.nl"`
Om deze prefix niet in de uitvoer terecht te laten komen voegen we het volgende toe:
`exclude-result-prefixes="pers"`
Ten slotte moeten we ervoor zorgen dat de default namespace van de uitvoer weer gelijk is aan die van de invoer: `xmlns="http://www.pers.nl".`

De push benadering is in dit geval handiger. Als we uitgaan van de in paragraaf 2.3.5 behandelde *identity transform* stylesheet, hoeven we alleen de uitzondering toe te voegen. De identity tranfsorm stylesheet heeft de volgende code:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="@*|node()">
  <xsl:copy>
   <xsl:apply-templates select="@*|node()"/>
  </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

Hierin staat dus dat alle attributen, elementen of tekst nodes letterlijk moeten worden overgenomen met xsl:copy. Met xsl:apply-templates select="@*|node()" geven we aan dat datzelfde geldt voor de sub-elementen.

Deze simpele maar doeltreffende stylesheet kunnen we gebruiken in een ander stylesheet, door deze te importeren. In een volgende paragraaf zullen we het importeren nader toelichten. Deze techniek is gebruikt in de volgende stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
exclude-result-prefixes="pers" xmlns:pers="http://www.pers.nl" xmlns="http://www.pers.nl">

 <xsl:import href="identity.xsl" />

  <xsl:template match="pers:adres">
   <adres><xsl:value-of select="pers:straat"/>, <xsl:value-of select="pers:plaats"/></adres>
  </xsl:template>

</xsl:stylesheet>
```

Het <stylesheet> element ziet er hetzelfde uit als in adres_foreach.xsl: dezelfde attributen zijn opgenomen om een default namespace van http://www.pers.nl te genereren. Vervolgens is identity.xsl geïmporteerd, die er dus voor zorgt dat default alle elementen, attributen en tekst nodes letterlijk vanuit het bron bestand naar de uitvoer worden gekopieerd. We hoeven nu alleen nog de uitzondering toe te voegen: een template die bepaalt wat er moet gebeuren met een <adres> element.

Stel weer even dat er later aan personen.xml <telefoon> en <emailadres> elementen worden toegevoegd. In deze stylesheet hoeft dan helemaal niets te veranderen! Alle elementen worden gewoon gekopieerd, alleen <adres> elementen krijgen een speciale behandeling.

2.4.2.1 Een HTML pagina opbouwen

Een andere toepassing waarbij de push benadering gebruikt kan worden is het opbouwen van een HTML-pagina uit XML gegevens. We bekijken weer eerst de pull benadering. Hier wordt een HTML-pagina gegenereerd voor een CD-lijst.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">
   <xsl:output method="html"/>
   <xsl:template match="/">
      <html>
         <head> ... </head>
         <body>
            <h2>CD overzicht:</h2>
            <table>
               <thead>
               <tr>
                  <th>artiest</th>
                  <th>titel</th>
                  <th>speelduur</th>
               </tr>
               </thead>
               <tbody>
               <xsl:for-each select="/cd:cdlijst/cd:cd">
                  <xsl:sort select="cd:artiest"/>
                     <tr>
                     <td><xsl:value-of select="./cd:artiest"/></td>
                     <td><xsl:value-of select="./cd:cdtitel"/></td>
                     <td align="right"><xsl:value-of select="./@speelduur"/></td>
                  </tr>
               </xsl:for-each>
               </tbody>
            </table>
         </body>
      </html>

   </xsl:template>
</xsl:stylesheet>
```

Hierin wordt een HTML-tabel aangemaakt en gevuld met de gegevens uit CDLijst1.xml, gesorteerd op artiest. Merk op dat een pull benadering in dit geval goed te verdedigen valt: we moeten in de stylesheet precies aangeven wat we waar in de tabel willen hebben. Alle andere gegevens in het document zijn voor deze toepassing niet relevant.
We kunnen zoiets ook met templates regelen, zoals in de volgende stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">
   <xsl:output method="html"/>
   <xsl:template match="/cd:cdlijst">
      <html>
         <head> ... </head>
         <body>
            <h2>CD overzicht:</h2>
            <table>
               <thead>
               <tr>
                  <th>artiest</th>
                  <th>titel</th>
                  <th>speelduur</th>
               </tr>
               </thead>
               <tbody>
                  <xsl:apply-templates><xsl:sort select="cd:artiest"/></xsl:apply-templates>
               </tbody>
            </table>
         </body>
      </html>
   </xsl:template>

   <xsl:template match="cd:cd">
      <tr>
         <xsl:apply-templates select="cd:artiest"/>
         <xsl:apply-templates select="cd:cdtitel"/>
         <xsl:apply-templates select="@speelduur"/>
      </tr>
   </xsl:template>

   <xsl:template match="cd:artiest|cd:cdtitel">
      <td>
         <xsl:value-of select="."/>
      </td>
   </xsl:template>

   <xsl:template match="@speelduur">
      <td align="right">
         <xsl:value-of select="."/>
      </td>
   </xsl:template>

</xsl:stylesheet>
```

Het eerste wat opvalt is dat de code er nu niet korter op is geworden, maar zelfs langer! Ook met gebruik van templates blijft het noodzakelijk om vanuit de stylesheet te sturen waar welke data terechtkomen, om een nette tabel aan te maken. In zekere zin is dit dus ook een pull-benadering, zij het met gebruik van templates.

Bekijk nu het eerste template: `<xsl:template match="/cd:cdlijst">`. Merk op dat het <xsl:sort> element nu als child element is opgenomen in `<xsl:apply-templates>`. Hier staat `<xsl:sort select="cd:artiest" />`. Dit is opmerkelijk, want cd:artiest is geen child-element van cd:cdlijst. Dit kan als volgt worden begrepen: `<xsl:apply-templates />` gaat templates toepassen op de nodes onder /cd:cdlijst, in dit geval zijn dat de cd:cd elementen. Die elementen moeten worden gesorteerd op een van de nodes onder cd:cd, in dit geval cd:artiest.

Bekijk ook het tweede template: `<xsl:template match="cd:cd">`. Hierin zien we drie keer `<xsl:apply-templates />` staan. Op deze manier dwingen we af dat de drie kolommen van de tabel in de juiste volgorde worden gevuld: eerst de artiest, dan de titel en tot slot de speelduur. Als hier `<xsl:apply-templates match="*|@*" />` had gestaan, zou de volgorde afhangen van de volgorde in het XML-bestand. Hier willen we dat niet: we kiezen welbewust voor een pull-benadering, om de juiste gegevens onder de juiste kop op te halen.
De voordelen van het gebruik van templates zijn hier beperkt. Het blijft dus van belang de voor- en nadelen van de push of de pull benadering voor een bepaalde toepassing goed af te wegen.

2.4.1 De modus van een template

Zoals we hebben gezien wordt in de push benadering het XML document als het ware door de templates heen geduwd. Voor elke child-node wordt met `<xsl:apply-templates/>` gezocht naar het meest specifieke template om toe te passen. Soms is het echter wenselijk om een child-node vaker te bewerken, op verschillende manieren. Bijvoorbeeld een keer om de tekst op te maken, en nog een keer om een index of inhoudsopgave te genereren. Voor dergelijke gevallen is het handig om templates in verschillende modes te kunnen aanroepen. Om dit mogelijk te maken kunnen we in `<xsl:apply-templates>` en `<xsl: template>` het attribuut `mode` gebruiken.

In het volgende voorbeeld worden de namen van personen twee keer weergegeven: eerst in hoofdletters, daaronder in kleine letters. Daartoe wordt twee keer apply-templates aangeroepen met verschillende modes. Zo'n <xsl:apply-templates> met een mode wordt opgevangen door een <xsl:template> met dezelfde mode.

```
<xsl:template match="/">
<namen>
   <hoofdletters>
      <xsl:apply-templates mode="upper"></xsl:apply-templates>
   </hoofdletters>
   <klein>
      <xsl:apply-templates mode="lower"></xsl:apply-templates>
   </klein>
</namen>
</xsl:template>
```

Op het niveau van de root wordt dus twee keer <xsl:apply-templates> aangeroepen, met een verschillende mode. Het enige element dat op root-niveau wordt aangetroffen is het element pers:personen. Merk op dat er *geen* template voor dit element bestaat. Er bestaat echter wel een default template bij het gebruik van mode:

```
<xsl:template match="*|/" mode="x">
  <xsl:apply-templates mode="x"/>
</xsl:template>
```

Dit default template zorgt ervoor dat het mode attribuut netjes wordt doorgegeven van <xsl:template> naar <xsl:apply-templates>. Zo wordt de juiste template gevonden als pers:persoon wordt gevonden. De eerste template, met mode="upper" zet de namen om in hoofdletters, de tweede template zet de namen om in kleine letters.

```
<xsl:template match="pers:persoon" mode="upper">
   <naam>
   <xsl:value-of select="translate(concat(pers:voornaam,' ',pers:achternaam),
                        'abcdefghijklmnopqrstuvwxyz','ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
   </naam>
</xsl:template>

<xsl:template match="pers:persoon" mode="lower">
   <naam>
   <xsl:value-of select="translate(concat(pers:voornaam,' ',pers:achternaam),
                        'ABCDEFGHIJKLMNOPQRSTUVWXYZ','abcdefghijklmnopqrstuvwxyz')"/>
   </naam>
</xsl:template>
```

2.4.2 Templates aanroepen

Ook binnen stylesheets die wel gebruik maken van pull technologie kunnen we gebruik maken van meerdere templates. Hiervoor geven we de verschillende templates een naam (attribuut `name`). Met het element `<xsl:call-template>` geven we dan aan welk template we willen gebruiken. De verwijzing naar het gewenste template wordt weer met behulp van het attribuut `name` gedaan.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">
   <xsl:output method="html" />
   <xsl:template match="/">
      <html>
         <head> [...] </head>
         <body>
            <h2>CD overzicht:</h2>
            <xsl:choose>
               <xsl:when test="//cd:albumhoes">
                  <xsl:call-template name="met" />
               </xsl:when>
               <xsl:otherwise>
                  <xsl:call-template name="zonder" />
               </xsl:otherwise>
            </xsl:choose>
         </body>
      </html>
   </xsl:template>
   <xsl:template name="zonder">
      <table> [...] </table>
   </xsl:template>
   <xsl:template name="met"> [...] </xsl:template>
</xsl:stylesheet>
```

In het bovenstaande voorbeeld zien we een stylesheet met meerdere templates. Bij het uitvoeren wordt eerst een stuk html aangebracht, vervolgens wordt met behulp van xsl:choose gecontroleerd of er een element albumhoes aanwezig is. Als dit zo is wordt template "met" uitgevoerd, bestaat dit element niet dan wordt de template "zonder" uitgevoerd. De code van beide templates is in de afbeelding verborgen.

Als een document zonder albumhoes element wordt getransformeerd zal de ene template worden uitgevoerd, bij een document met albumhoes elementen zal de twede template worden uitgevoerd.

Ook bij het aanroepen van templates met call template moet heel nauwkeurig naar de structuur van de XML gekeken worden. Stukken XML worden naar binnen getrokken en verwerkt. Werken met call-templates lijkt dan ook het meest op de pull technologie.

2.4.3 Templates importeren

Binnen een stylesheet kunnen we ook gebruik maken van een bestaand stylesheet. Om een stylesheet te importeren is het element `<xsl:import>` te gebruiken. Dit element heeft een attribuut `href`. waarin de URI van het te importeren stylesheet wordt opgegeven.

Het <xsl:import> element is een "top-level" element, dat direct onder <xsl:stylesheet> of <xsl:transform> komt te staan, dus nog voor de templates.

We hebben reeds kennis gemaakt met <xsl:import> bij het aanroepen van identity.xsl in adres_applytemplates.xsl:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
exclude-result-prefixes="pers" xmlns:pers="http://www.pers.nl" xmlns="http://www.pers.nl">

 <xsl:import href="identity.xsl" />

   <xsl:template match="pers:adres">
      <adres><xsl:value-of select="pers:straat"/>, <xsl:value-of select="pers:plaats"/></adres>
   </xsl:template>

</xsl:stylesheet>
```

Merk op dat voor het <adres> element nu in principe twee templates bestaan: de template uit identity.xsl, met `match="@*|node()"`, en de lokale template, met `match="pers:adres"`. Waarom wordt voor adres-elementen de lokale template toegepast, in plaats van de algemenere template in identity.xsl? Dat heeft twee oorzaken:

- Templates uit geïmporteerde stylesheets hebben een lagere prioriteit dan lokale templates.
- De template uit de geïmporteerde stylesheet, met `match="@*|node()"` is minder specifiek dat de lokale template, met `match="pers:adres"`. Specifieke templates krijgen een hogere prioriteit.

N.B. Als er meerdere templates zijn met gelijke prioriteit, dan wordt de laatste daarvan uitgevoerd.

Bij het toepassen van templates met <xsl:apply-templates> krijgen lokale templates dus voorrang boven even specifieke geïmporteerde templates. We kunnen echter met <xsl:apply-imports> aangeven dat we juist de templates uit de geïmporteerde stylesheets willen toepassen.

In het volgende voorbeeld zullen bij <xsl:apply-templates/> de *lokale* templates worden uitgevoerd, en niet die van de geïmporteerde stylesheet.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
exclude-result-prefixes="pers" xmlns:pers="http://www.pers.nl" xmlns="http://www.pers.nl">

<xsl:import href="identity.xsl" />

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

 <xsl:template match="@*|text()">
   <xsl:value-of select="."/>
 </xsl:template>

</xsl:stylesheet>
```

De lokale template met `match="@*|text()"` in deze stylesheet is even specifiek als de template in de geïmporteerde stylesheet identity.xsl. Omdat de template uit de geïmporteerde stylesheets een lagere prioriteit heeft, wordt deze niet toegepast. De lokale template wordt wel toegepast, maar daar gebeurt niks in.

Als we <xsl:apply-templates/> wijzigen in <xsl:apply-imports/> zullen wel de geïmporteerde templates worden uitgevoerd.

Behalve het element <xsl:import> kan ook gebruik worden gemaakt van <xsl:include> om een stylesheet te importeren, of beter gezegd: om deze "in te voegen". Het grootste verschil tussen <xsl:import> en <xsl:include> is dat templates die met <xsl:include> worden ingevoegd behandeld worden alsof het lokale templates zijn. Ze krijgen dus niet automatisch een lagere prioriteit. Verder is <xsl:include> een top-level element, dus ook een child node van <xsl:stylesheet>, maar in tegenstelling tot <xsl:import> kan dit element ook tussen of na de templates komen te staan. Deze volgorde is van belang: een template met dezelfde match als een eerder template zal die eerdere template overschrijven.

2.5 XSLT functies

Naast elementen kent XSLT ook een groot aantal functies. De XPath functies worden gebruikt in XPath expressies om de huidige node te bewerken, of om te testen om wat voor soort node het gaat. Deze functies zullen we in een volgende paragraaf bespreken. Hier volgt vast een aantal andere XSLT functies.

Functie	Beschrijving
current()	Geeft de huidige node terug. Merk op dat `select="current()"` vergelijkbaar is met `select="."`.
document(`string [,node-set]`)	Functie om de node set van een extern xml document te benaderen.
element-available(`string`)	Test of een opgegeven XSLT element wordt ondersteund door de XSLT processor.
format-number(`number, format [,decimalformat]`)	Converteert een getal naar een tekst in de opgegeven notatie.
function-available(`string`)	Test of een opgegeven functie wordt ondersteund door de XSLT processor.
generate-id(`[node-set]`)	Deze functie geeft een unieke tekstwaarde waarmee een node kan worden geïdentificeerd.
key(`string, value`)	Deze functie geeft een node-set bij de opgeven waarde. Er wordt gerefereerd aan een `<xsl:key>` element.
system-property(`string`)	Deze functie geeft een aantal systeem eigenschappen. Mogelijk parameter-waarden zijn: `xsl:version`, `xsl:vendor` en `xsl:vendor-url`.
unparsed-entity-uri(`string`)	Deze functie geeft de URI van een niet geparste entity.

We zullen nu twee functies hiervan nader bekijken.

2.5.1 document()

Met de functie document() kunnen gegevens uit een ander xml document gehaald worden. De functie heeft twee argumenten. Het eerste argument is verplicht en bevat de URI van het externe XML document, het tweede argument is optioneel en bevat een node set van het externe document.
De functie zorgt er voor dat het hele document wordt geladen, Met behulp van een XPath expressie kan deze weer worden doorlopen.

In het volgende voorbeeld gaan we met behulp van de functie document() twee xml documenten samenvoegen.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">
    <xsl:output method="xml"/>

    <xsl:template match="/">

        <cdlijst xmlns="http://www.cd.nl">
            <xsl:copy-of select="document('cdlijst1.xml')/cd:cdlijst/cd:cd"/>
            <xsl:copy-of select="document('cdlijst2.xml')/cd:cdlijst/cd:cd"/>
        </cdlijst>
    </xsl:template>


</xsl:stylesheet>
```

De stylesheet zorgt voor het root element `<cdlijst>` en voegt de juiste namespace toe. Vervolgens wordt met behulp van `<xsl:copy-of>` de inhoud van *cdlijst1.xml* opgehaald. In het select attribuut van dit element zien we de functie `document()` terug. Als argument heeft deze functie de bestandsnaam. Daar achter volgt een XPath expressie. Vervolgens wordt dit ook gedaan voor *cdlijst2.xml*.

Merk op dat het resultaat met deze stylesheet niet afhankelijk is van het originele xml document. Zelfs een xml document met enkel een root element is al voldoende om de transformatie succesvol uit te voeren, omdat het laden van het document door de functie wordt uitgevoerd.

2.5.2 key()

Om bepaalde elementen snel te kunnen vinden kunnen we gebruik maken van een <key> element en de key() functie. Het <key> element krijgt drie parameters:

attribuut	waarde	beschrijving
name	naam	de naam van de key
match	patroon	de nodes om op te zoeken
use	expressie	het element of attribuut dat wordt gebruikt als sleutel om de gezochte nodes te vinden

Een bijzonderheid van het *use* attribuut is dat we niet hoeven aan te geven waar in de boomstructuur dat attribuut te vinden is. Het key element zorgt ervoor dat er een extra node-set in het geheugen wordt aangemaakt van sleutelwaarden met bijbehorende nodes. Met de key() functie kunnen we daarin snel de gezochte nodes vinden. Deze functie heeft twee attributen: de naam van de key, en de waarde van de sleutel.
Bekijk de code van de volgende stylesheet. Deze is bedoeld om in een document met personen een specifieke persoon te vinden.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    exclude-result-prefixes="pers" xmlns:pers="http://www.pers.nl" xmlns="http://www.pers.nl">

    <xsl:key match="pers:persoon" name="pers_key" use="pers:achternaam" />

    <xsl:template match="/">
        <xsl:for-each select="key('pers_key', 'Scholten')">
            <xsl:copy-of select="pers:adres" />
        </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

Het key element pers_key zoekt naar personen (het match-attribuut). Als sleutel om naar personen te zoeken wordt de achternaam gebruikt (het use-attribuut). De key wordt in dit geval gebruikt in het select attribuut van een <xsl:for-each element>: de key() functie haalt alle personen op met de achternaam Scholten (in dit geval is dat er één). Van de gevonden elementen wordt ten slotte het adres getoond.

We kunnen ook de key uitbreiden, zodat niet alleen op achternaam kan worden gezocht, maar bijvoorbeeld ook op voornaam:

```
<xsl:key match="pers:persoon" name="pers_key" use="pers:achternaam|pers:voornaam" />
```

2.6 XPath Axes

Om naar nodes te verwijzen in een XML document hebben we tot nu toe XPath expressies gebruikt met de volgende bouwstenen:

Expressie	beschrijving
nodename	Selecteert alle nodes van met deze naam op deze plek in het pad
/	Selecteert de root (waarvan het root element het enige child element is).
//nodename	Selecteert alle nodes in het document die aan de opgegeven naam voldoen; het maakt niet uit waar ze in de structuur staan.
.	Selecteert de huidige node.
..	Selecteert de parent van de huidige node.
@attributename	Selecteert een attribuut met deze naam.
*	Een wildcard voor elke element node.
@*	Een wildcard voor elke attribute node.

In de bovengenoemde structuren gebruiken we steeds een absoluut pad om door een xml document te navigeren. In sommige gevallen kunnen we het gewenste resultaat makkelijker bereiken door gebruik te maken van *axes*. Een axis (as) definieert een pad relatief ten opzichte van de huidige node. Hieronder volgt een lijst met mogelijke axes:

Axis	Resultaat
ancestor	Selecteert alle bovenliggende nodes (parent, grandparent, etc.) van de huidige node
ancestor-or-self	Selecteert de huidige node en alle bovenliggende nodes (parent, grandparent, etc.) van de huidige node.
attribute	Selecteert alle attributen van de huidige node.
child	Selecteert alle child elementen van de huidige node.
descendant	Selecteert alle onderliggende nodes (children, grandchildren, etc.) van de huidige node.
descendant-or-self	Selecteert de huidige node en alle onderliggende nodes (children, grandchildren, etc.).
following	Selecteert alle nodes na de huidige node.

Axis	Resultaat
following-sibling	Selecteert alle nodes na de huidige node die op hetzelfde niveau als de huidige node staan.
namespace	Selecteert alle namespace nodes van de huidige node.
parent	Selecteert de bovenliggende node van de huidige node.
preceding	Selecteert alle nodes voor de huidige node.
preceding-sibling	Selecteert alle nodes voor de huidige node die op hetzelfde niveau als de huidige node staan.
self	Selecteert de huidige node.

Om een axis in een XPath expressie te gebruiken, wordt de naam op de juiste plaats in het pad opgenomen gevolgd door :: . Achter de laatste dubbele punt kunnen we dan de naam van een node opgeven of een * om alle nodes van dat niveau te kiezen.

Hier volgen enkele voorbeelden van XPath expressies, in dit geval op bestellingen:

1. **/bestelling/product/attribute::***
   (Selecteert alle attributen van het element *product*.)
2. **/bestelling/product/attribute::type**
   (Selecteert van elk *product* met een attribuut *type* dit attribuut.)
3. **/bestelling/product[3]/productid/following::***
   (Selecteert alle nodes na *productid* van het 3e *product*.)
4. **/bestelling/product[3]/productid/following-sibling::***
   (Selecteert alle voorgaande nodes op het zelfde niveau als *productid* van het 3e *product*.)
5. **/bestelling/product[3]/descendant::***
   (Selecteert alle childnodes van het 3e *product*.)
6. **/bestelling/descendant::***
   (Selecteert alle childnodes van *bestelling*.)
7. **/bestelling/descendant::prijs**
   (Selecteert alle *prijs* nodes van *bestelling*.)

2.7 XPath operatoren

Binnen een XPath expressie kunnen we gebruik maken van operatoren om bewerkingen uit te voeren, of om extra voorwaarden mee te geven. We kunnen bijvoorbeeld alleen de producten boven een opgeven prijs selecteren.
De meeste rekenkundige operatoren spreken voor zich. De operator | voor het combineren van node sets zijn we ook al eerder tegengekomen:

Operator	beschrijving	Voorbeeld
+	optellen	prijs + btw
-	aftrekken	prijs - btw
*	vermenigvuldigen	prijs * 1.19
div	delen	aantal div 4
mod	modulus (rest van een deling)	aantal mod 4
|	verwerkt twee nodesets	//boek | //cd

Merk op dat we div gebruiken in plaats van / om te delen. Immers: het karakter / heeft binnen XPath al een andere betekenis.

De bovenstaande operatoren leveren steeds als resultaat een waarde of een node-set op. Binnen XPath-expressies mag met haakjes gewerkt worden om een volgorde af te dwingen.

Er zijn ook operatoren die een vergelijking uitvoeren. Als de vergelijking klopt is het resultaat van de expressie "*true*" en klopt de vergelijking niet dan is het resultaat van de expressie "*false*".

Operator	beschrijving	Voorbeeld
=	is gelijk aan	aantal=10
!=	is ongelijk aan	aantal!=10
<	kleiner dan	aantal<10
<=	kleiner of gelijk aan	aantal<=10
>	groter dan	aantal >10
>=	groter of gelijk aan	aantal >=10
or	logische of	aantal =10 or aantal =20
and	logische en	aantal >10 and aantal <20

Het teken < mag niet zomaar als XML tekst worden gebruikt, omdat dat gezien wordt als het begin van een tag. In plaats daarvan wordt *<* gebruikt. De expressie *aantal <4* wordt dan *aantal < 4* voor een werkende code. Op dezelfde manier kan ook *>* gebruikt worden in plaats van >.

2.7.1 Vergelijking van node-sets

Dezelfde operatoren kunnen ook gebruikt worden bij het vergelijken van nodesets. Hierbij moeten we rekening houden met de volgende principes.

Twee node-sets zijn gelijk als er in beide node-sets een node voorkomt met gelijke string-waarden.

De expressie `//a = //b` geeft dus true() als er een element a voorkomt met een zelfde waarde als een element b, ook al hebben alle andere elementen ongelijke waarden. Iets degelijks geldt ook voor vergelijking tussen een node-set en een waarde. Dus `//aantal = 3` geeft true() als er een element aantal voorkomt met de waarde 3.

Omgekeerd geldt ook het volgende:

Twee node-sets zijn ongelijk als er in beide node-sets een node voorkomt met ongelijke string-waarden.

De expressie `//a != //b` geeft dus true() als er een element a voorkomt met een andere waarde dan een element b, ook al hebben verder alle elementen gelijke waarden.

Dit betekent dat het volgende heel wel mogelijk is!

```
//aantal = 3 and //aantal != 3
```

Dit betekent namelijk dat er in de node-set een element voorkomt met de waarde 3, en dat er in dezelfde node-set een element voorkomt met een waarde ongelijk aan 3.

De overige operatoren werken bij node-sets op dezelfde manier. Dus //aantal > 3 betekent dat er *een* (of meer) element aantal is met een waarde groter dan 3.

Om te testen of een element *niet* voorkomt in een node-set moeten we gebruiken van de boolean functie not():

`not(//aantal = 3)`	er is geen element met de waarde van 3
`not(//aantal != 3)`	alle elementen hebben de waarde 3

Van deze eigenschappen kunnen we gebruik maken om op zoek te gaan naar unieke waarden. We voor elke waarde de eerste node met die waarde vinden, via de volgende redenering: *er bestaan geen nodes voor de eerste node met dezelfde waarde als die eerste node.*

Zo vinden we op de volgende manier de unieke jaartallen van cd-tracks:

```
<xsl:for-each select="//cd:track[not(cd:jaar=preceding-sibling::*/cd:jaar)]/cd:jaar">
        <xsl:copy-of select="."></xsl:copy-of>
 </xsl:for-each>
```

2.8 XPath functies

Naast operatoren kunnen we in XPath expressies gebruik maken van functies. XPath functies kunnen gebruikt worden om een waarde op te halen of om op een waarde te testen. XPath functies kunnen in enkele categorieën worden ingedeeld:

- Boolean functies
- Node-set functies
- Numerieke functies
- String functies

Een functie geeft altijd een waarde terug. Bij de meeste functies kunnen argumenten meegegeven worden. Deze argumenten staan tussen haakjes en worden gescheiden door een komma. We zullen per categorie een aantal functies behandelen. In de syntax staat steeds de functie genoemd met tussen haakjes het type argument dat moet worden ingevuld. Een vraagteken achter een argument geeft aan dat dit argument optioneel is.

2.8.1 Node-set functies

De volgende functies worden gebruikt voor node-sets. Sommige functies kunnen een node-set mee krijgen als parameter. Deze is optioneel, hier aangeduid met een vraagteken.

Functie	Beschrijving
position()	Deze functie geeft de positie van de huidige node terug als getal.
last()	Deze functie geeft de positie van de laatste node terug als getal.
namespace-uri(node-set?)	Deze functie geeft de namespace uri van de huidige node (default) als string terug, of van de eerste node van de opgegeven node-set.
id(object)	Deze functie selecteert elementen met het opgegeven id.

Functie	Beschrijving
local-name(node-set?)	De functie local-name() geeft een string terug met de het locale deel van de naam van de huidige node (default), of van de eerste node van de opgegeven node-set.
name(node-set?)	De functie name doet bijna het zelfde als local-name() alleen wordt er nu een qname (qualfied name) terug gegeven. Dit is handig wanneer nodes met dezelfde naam in verschillende namespaces voorkomen.

2.8.2 Boolean functies

Boolean functies worden over het algemeen gebruikt binnen testen. Ze geven een "true" of een "false" terug:

Functie	Beschrijving
boolean(object)	Deze functie converteert het opgegeven argument naar een boolean. • Een getal wordt "false" als het 0 of NaN is. • Een tekst wordt "false" als de tekst lengte 0 heeft. • Een node set wordt false als deze leeg is .
not(boolean)	De functie not() geeft het tegengestelde van een boolean expressie.
true()	Deze functie heeft geen argumenten en geeft altijd "true" terug.
false()	Deze functie heeft geen argumenten en geeft altijd "false" terug.

2.8.3 String functies

De volgende functies zijn te gebruiken voor tekstwaarden en tekst nodes. Optionele parameters zijn weer aangeduid met een vraagteken.

Functie	Beschrijving
string(object?)	Deze functie converteert het opgeven argument naar een tekst. Het argument kan een number, een boolean of een node-set zijn.
concat(string, string, ..?)	De functie concat zet alle opgeven argumenten achter elkaar zodat ze samen 1 tekstwaarde vormen.
starts-with(string, string)	Deze functie geeft een boolean waarde terug. Het resultaat is "true" als het eerste argument begint met het tweede argument.
contains(string, string)	Deze functie geeft een boolean waarde terug. Het resultaat is "true" als het tweede argument in het eerste argument gevonden wordt.
substring(string, number, number?)	De functie substring geeft een deel van het eerste argument terug beginnend op de positie die wordt opgegeven bij het tweede argument. Het derde argument is optioneel; hiermee kan de lengte van de substring worden opgegeven. voorbeeld: substring("123456", 3) wordt 3456. substring("123456",3,2) wordt 34.
substring-before(string, string)	De functie substring-before geeft een deel van het eerste argument terug dat voor het tweede argument staat.
substring-after(string, string)	De functie substring-after geeft een deel van het eerste argument

Functie	Beschrijving
	terug dat na het tweede argument staat.
string-length(string?)	Deze functie telt het aantal karakters in het argument. en geeft dit als een getal terug.
normalize-space(string?)	Deze functie haalt bij het opgegeven argument alle spaties aan het begin en eind weg. Tussen twee woorden wordt het aantal spaties teruggebracht tot één. Het resultaat wordt als een string terug gegeven.
translate(string, string, string)	Deze functie vervangt in het eerste argument de tekens van het tweede argument door de overeenkomstige tekens van het derde argument.

Merk op dat er geen upper() of lower() functie bestaat voor XPath. Om een naam in hoofdletters om te zetten moeten we de translate() functie gebruiken:

```
<xsl:value-of select="translate(naam,
                      'abcdefghijklmnopqrstuvwxyz',
                      'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
```

2.8.4 Numerieke functies

Numerieke functies worden gebruikt om berekeningen uit te voeren. Deze functies geven altijd een getal terug. We kennen de volgende numerieke XPath functies:

Functie	Beschrijving
round(number)	Deze functie krijgt als argument een getal mee en geeft als resultaat de afgeronde waarde van het getal als integer terug.
floor(number)	Deze functie krijgt als argument een getal mee en geeft als resultaat de naar beneden afgeronde waarde van het getal als integer terug.
ceiling(number)	Deze functie krijgt als argument een getal mee en geeft als resultaat de naar boven afgeronde waarde van het getal als integer terug.
sum(node-set)	Deze functie krijgt als argument een node-set mee. Het resultaat is de optelling van de waarden van de opgegeven node in deze node-set.
count(node-set)	Deze functie krijgt als argument een node-set mee. Het resultaat is aantal keren dat de opgegeven node in deze node-set voorkomt.
number(object?)	Deze functie converteert het opgeven argument naar een getal. Als dit niet mogelijk is wordt NaN terug gegeven. Het argument kan een string, een boolean of een node-set of zijn. Van een node-set wordt eerst de string weergave bepaald. Een boolean geeft een "1" terug bij "true"en een "0" bij "false".

Om totalen te berekenen maken we dus gebruik van de functie sum. Voor zover alleen getallen bij elkaar opgeteld worden werkt dat heel eenvoudig. Het volgende voorbeeld laat zien dat het lastiger is om ook met aantallen rekening te houden.

Hier doorlopen we een aantal bestellingen, met een prijs en een aantal. Het volgende zou makkelijk zijn maar werkt niet:

```
<xsl:value-of select="sum(//prijs * //aantal)"></xsl:value-of>
```

De volgende constructie werkt wel. Hierbij wordt een template recursief aangeroepen, steeds met een volgende prijs * aantal.

```
<xsl:template name="bereken">
   <xsl:param name="artikelen"/>
   <xsl:choose>
      <xsl:when test="$artikelen">
         <xsl:variable name="recursief_resultaat">
            <xsl:call-template name="bereken">
               <xsl:with-param name="artikelen" select="$artikelen[position() > 1]"/>
            </xsl:call-template>
         </xsl:variable>
         <xsl:value-of select="number($artikelen[1]/aantal)*number($artikelen[1]/prijs)
        + $recursief_resultaat"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:value-of select="0"/>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

Bij het aanroepen van de template bereken wordt een lijst artikelen meegegeven. Als die lijst leeg is (test="$artikelen"), wordt 0 geretourneerd, anders wordt de template recursief aangeroepen met een slinkende lijst artikelen. Steeds wordt van het eerste artikel het totaal berekend (aantal * prijs) en opgeteld bij de variabele recursief_resultaat. De overige artikelen (position()>1) worden meegegeven bij het opnieuw aanroepen van de template. Uiteindelijk zijn zo van alle artikelen de totalen berekend en bij recursief_resultaat opgeteld. De laatste keer dat de template wordt aangeroepen wordt de waarde in <xsl:value-of ....> teruggegeven: deze bevat dan het totaal waar we naar op zoek waren.

We gaan deze template uitvoeren op de plaats waar we het totaal willen zien. Hiervoor gebruiken we `<xsl:call-template>`

```
<xsl:call-template name="bereken">
    <xsl:with-param name="artikelen" select="artikel"/>
</xsl:call-template>
```

Met xslt 2.0 is het bovenstaande eenvoudiger op te lossen. We zullen dit in het volgende hoofdstuk bespreken.

2.9 Parameters

In het voorbeeld hierboven zijn twee elementen gebruikt die we nog niet eerder hebben besproken. Dat zijn de elementen `<xsl:param>` en `<xsl:with-param>`. Deze elementen worden veel gebruikt in combinatie met templates. Parameters kunnen worden gebruikt om een waarde aan een template mee te geven. In ons voorbeeld wordt binnen een template een parameter gedefinieerd met `<xsl:param>`. Deze parameter krijgt de naam *artikelen*.

```
<xsl:param name="artikelen"/>
```

Bij het aanroepen van de template wordt deze parameter gevuld met een waarde. Door het element `<xsl:with-param>`. In ons geval is de waarde die wordt meegegeven een node-set.

```
<xsl:call-template name="bereken">
    <xsl:with-param name="artikelen" select="artikel"/>
</xsl:call-template>
```

Omdat het aanroepen van de stylesheet recursief is zien we ook in de stylesheet zelf weer een `<xsl:call-template>` met een `<xsl:with-param>`. Ook hier wordt weer een nodeset als waarde aan de parameter gegeven.

```
<xsl:call-template name="bereken">
    <xsl:with-param name="artikelen" select="$artikelen[position() > 1]"/>
</xsl:call-template>
```

Het is ook mogelijk om globale parameters aan een stylesheet mee te geven. Hierbij is het <xsl:param> element dus een child-element van <xsl:stylesheet>. Hoe de waarde van zo'n parameter wordt meegegeven aan de stylesheet is afhankelijk van de gebruikte XSLT processor. Raadpleeg de documentatie bij uw XSLT processor voor nadere informatie hierover.

3 XSLT 2.0

3.1 Inleiding

Sinds januari 2007 is XSLT 2.0 door het w3c vastgesteld. Hieraan gekoppeld is de eveneens nieuwe standaard voor XPath 2.0. In dit hoofdstuk gaan we XSLT 2.0 nader bekijken. We zullen daarnaast de mogelijkheden van XPath 2.0 behandelen. Veel van XSLT 1.0 is ook binnen XSLT 2.0 toe te passen. Toch kunnen we XSLT 2.0 niet alleen als een uitbreiding op de vorige versie zien. Sommige concepten zijn samen met het onderliggende datamodel op een aantal punten veranderd. We zullen in dit hoofdstuk dan ook aandacht besteden aan de verschillen met XSLT 1.0.

3.2 XSLT 2.0 transformaties

Voor het uitvoeren van transformaties met XSLT 2.0 is een aantal zaken van belang. Als eerste zal de juiste namespace gebruikt moeten worden. Deze verschilt niet van XSLT 1.0. Wel zal een ander versienummer opgenomen moeten worden. Het element stylesheet zal er dan als volgt uit zien:

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Het enig afwijkende met XSLT 1.0 is in dit geval dus het versienummer. Het attribuut version heeft nu de waarde 2.0 gekregen. Dit heeft wel gevolgen voor de parser. De standaard parsers kunnen deze transformatie niet verwerken. Hiervoor is een xslt 2.0 parser nodig. We zullen de transformatie daarom uitvoeren met Saxon 8B. Binnen XML Blueprint is Saxon 8B beschikbaar in het menu *Tools*. Voor de transformatie moet weer aangegeven worden welk bestand getransformeerd moet worden. Hiervoor kan weer de knop *Setup XSLT Transformation*  worden gebruikt.

Bij transformeren met Saxon 8B wordt altijd met een opgeslagen bestand gewerkt. XML Blueprint zal daarom waar nodig vragen of de wijzigingen moeten worden opgeslagen

3.2.1 Verschil bij element selectie

Vaak leveren XSLT 1.0 transformaties hetzelfde resultaat op als ze als XSLT 2.0 worden uitgevoerd. Maar dit is niet altijd zo. Bekijk bijvoorbeeld de volgende stylesheet:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:cd="http://www.cd.nl">
   <xsl:template match="/">
      <cd>
         <xsl:value-of select="//cd:cdtitel"/>
      </cd>
   </xsl:template>
</xsl:stylesheet>
```

Deze transformatie levert met XSLT 1.0 de eerste cdtitel uit de lijst. De overige titels worden genegeerd. Bij transformatie met XSLT 2.0 zien we juist alle titels verschijnen, door spaties gescheiden. Het resultaat van <xsl:value-of /> is nu een *sequence*, en reeks waarden. Met

het attribuut separator kunnen we bepalen hoe deze waarden gescheiden worden, bijvoorbeeld: `<xsl:value-of select="//cd:cdtitel" separator=", " />`

In de rest van dit hoofdstuk zullen we alle transformaties uitvoeren met Saxon 8B tenzij anders wordt aangegeven.

3.3 XSLT 2.0 elementen

De mogelijkheden voor transformaties zijn met XSLT 2.0 behoorlijk uitgebreid. Zo is het met XSLT 2.0 mogelijk om al in de stylesheet documenten te valideren met behulp van het element `<xsl:import-schema>`. Ook zijn er elementen die een attribuut `validation` hebben gekregen waarmee we een validatiemethode kunnen afdwingen. Dit geldt bijvoorbeeld voor de elementen: `<xsl:element>`, `<xsl:copy-of>` en `<xsl:document>`. Verder zijn er elementen toegevoegd om te groeperen en te sorteren. Ook zijn aan verschillende elementen extra attributen toegevoegd, zoals `type` en `collation`. In Appendix A vindt u een complete opsomming met alle elementen en hun attributen die binnen XSLT2.0 beschikbaar zijn. Hieronder volgt een lijst met een korte beschrijving van de nieuwe XSLT 2.0 elementen.

Element	Omschrijving
<xsl:analyze-string>	Dit element zoekt binnen een string naar een patroon dat overeenkomt met een reguliere expressie in het attribuut `regex`. Dit element maakt gebruik van de elementen `<xsl:matching-substring>` en `<xsl:non-matching-substring>` om te bepalen wat moet worden uitgevoerd.
<xsl:character-map>	Dit element wordt gebruikt om één karakter te vervangen door een ander karakter of een andere tekst. Dit element heeft een verplicht attribuut name. Naar deze naam kan worden verwezen in het attribuut `use-character-maps` van `<xsl:output>`
<xsl:document>	Met dit element kan een nieuwe document node worden gemaakt.
<xsl:for-each-group>	Met dit element kunnen groepen binnen een xml document gedefinieerd worden.
<xsl:function>	Dit element wordt gebruikt om zelf functies te declareren.
<xsl:import-schema>	Om zowel de import als de export te valideren met het schema kan het element `<xsl:import-schema>` worden gebruikt. Dit element verwijst naar een bestaand schema, of er wordt binnen dit element een nieuw schema gemaakt volgens de regels voor xml-schema. Het element `<xsl:import-schema>` is een child element van `<xsl:stylesheet>`.
<xsl:matching-substring>	Dit element wordt gebruikt binnen een `<xsl:analyze-string>` element en bepaalt wat er moet worden gedaan met een substring die voldoet aan de opgegeven reguliere expressie.
<xsl:namespace>	Dit element wordt gebruikt om een namespace attribuut aan een element toe te voegen.
<xsl:next-match>	Met dit element kunnen verschillende templates op hetzelfde element worden toegepast. Met het element `<xsl:next-match>` wordt gezocht of er nog een template is dat voor dit element gebruikt kan worden.
<xsl:non-matching-substring>	Dit element wordt gebruikt binnen een `<xsl:analyze-string>` element en bepaalt wat er moet worden gedaan met een substring die niet

Element	Omschrijving
	voldoet aan de opgegeven reguliere expressie.
<xsl:output-character>	Dit element wordt binnen een `<xsl:character-map>` gebruikt om de vervanging van een karakter te definiëren.
<xsl:perform-sort>	Met het element `<xsl:perform-sort>` kan een sequence (reeks elementen of waarden) worden gesorteerd, zonder gebruik te maken van een `<xsl:for-each>` of `<xsl:apply-templates>` element.
<xsl:result-document>	Als een XSLT document de output in verschillende documenten moet plaatsen, kan het element `<xsl:result-document>` worden gebruikt. Met het attribuut `href` kan de naam van het output document worden aangeven. Verder geeft het attribuut `format` aan van welk type het output document is (HTML,XML,...).
<xsl:sequence>	Met het element <xsl:sequence> wordt een sequence aangemaakt van items.

We zullen enkele van deze elementen hieronder nader toelichten.

3.3.1 analyze-string

Met dit element kunnen we een string vergelijken met een reguliere expressie. Bij het doorlopen van de string wordt deze verdeeld in substrings die wel en substrings die niet voldoen aan de reguliere expressie. Wat met die substrings moet gebeuren wordt geregeld in respectievelijk de elementen <xsl:matching-substring> en <xsl:non-matching-substring>.

Het volgende voorbeeld leest een reeks email-adressen in en verdeelt deze over een lijst correcte adressen en een lijst incorrecte adressen.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:import href="identity.xsl" />
    <xsl:template match="adres">
        <xsl:analyze-string regex="^(\w)+(\.(\w)+)?@(\w)+\.([a-z]){{2,}}$" select=".">
            <xsl:matching-substring>
                <correct>
                    <xsl:value-of select="." />
                </correct>
            </xsl:matching-substring>
            <xsl:non-matching-substring>
                <incorrect>
                    <xsl:value-of select="." />
                </incorrect>
            </xsl:non-matching-substring>
        </xsl:analyze-string>
    </xsl:template>
</xsl:stylesheet>
```

In dit voorbeeld is de gehele string beoordeeld: de reguliere expressie begint met ^ en eindigt met $ om aan te geven dat de expressie voor de gehele string moet gelden en niet voor een substring.

Een andere toepassing vam <xsl:analyze-string> is het bewerken van delen van een string. Het bron document is een simpele string van namen, door komma's en spaties gescheiden:

```
<?xml version="1.0" encoding="utf-8"?>
<baarden>Jan, Pier, Joris, Corneel</baarden>
```

De stylesheet zoekt daarin naar de gewone woorden, dus zonder de spaties en komma's:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:import href="identity.xsl" />
   <xsl:template match="baarden">
      <baarden>
      <xsl:analyze-string regex="\w+" select=".">
         <xsl:matching-substring>
            <baard>
               <xsl:value-of select="." />
            </baard>
         </xsl:matching-substring>
      </xsl:analyze-string>
      </baarden>
   </xsl:template>
</xsl:stylesheet>
```

Hier wordt een simpele reguliere expressie gebruikt om naar woord-karakters te zoeken. Deze stylesheet zoekt de losse woorden, zet daar een tag omheen, en doet niks met de overige karakters: de komma's en spaties komen dus niet in het resultaat terecht.

3.3.2 character-map

Het element character-map is bedoeld om een karakter om te zetten in een ander karakter of in een string. In het volgende voorbeeld wordt een transformatie naar HTML uitgevoerd. De stylesheet gebruikt een character-map element om harde returns (in Windows het hexadecimale karakter
) om te zetten naar een
 element. Daarnaast wordt als voorbeeld de letter é omgezet naar de HTML character entity é Zo'n HTML character entity is namelijk platform-onafhankelijk: de letter é kan in sommige browsers als onbekend vreemd teken worden gegenereerd. De code van de stylesheet ziet er als volgt uit:

```
<?xml version="1.0"?>
<!--voer de transformatie uit met het document character-map.xml-->
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:character-map name="html_karakters">
    <xsl:output-character character="&#xA;" string="&lt;br&gt;" />
    <xsl:output-character character="&#233;" string="&amp;eacute;" />
  </xsl:character-map>
  <xsl:output indent="yes" use-character-maps="html_karakters" method="html" />
  <xsl:template match="/">
    <html>
      <body>
        <xsl:value-of select="tekst" />
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Te zien is dat de character-map met de naam html_karakters gebruikt wordt in het xsl:output element:

```
use-character-maps="html_karakters".
```

Verder zien we dat de harde return wordt omgezet naar
 - waarbij < staat voor het < teken, en > voor het > teken. Samen staat er dus
: het HTML element voor een nieuwe regel (break).
De letter é (karakter 233) wordt omgezet naar é - waarbij & staat voor het ampersand teken &. Samen wordt dit dus é - de HTML character entity voor het é teken.

Door de mogelijkheden van character-map is het attribuut disable-output-escaping van <xsl:text> *deprecated* (verouderd) geworden. In paragraaf 2.3.1 is dit attribuut behandeld. We herhalen hier een deel van de tekst:

Het element xsl:text heeft een attribuut disable-output-escaping, die default (dus als dit attribuut niet wordt meegegeven) de waarden "no" heeft. Als dit attribuut de waarde "yes" krijgt, worden deze speciale tekens niet omgezet in character entities.

Voorbeeld:

```
<xsl:template match="/">
   <xsl:text disable-output-escaping="no">Als a > b en b > c dan geldt: c &lt; a </xsl:text>
</xsl:template>
```

Uitvoer:

```
Als a &gt; b en b &gt. c, dan geldt c &lt; a.
```

Merk op dat in de stylesheet het teken < als < moet worden meegegeven.

Als we hier `disable-output-escaping="yes"` *van maken, dan krijgen we netjes de volgende uitvoer:*

```
Als a > b en b > c dan geldt: c < a
```

In XSLT 2 lossen we dit op met een character-map waarin < en > worden omgezet naar dezelfde tekens (< en >) in plaats van de default: (< en >).
De code van de stylesheet ziet er dan bijvoorbeeld als volgt uit:

```
<?xml version="1.0"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:character-map name="xml_karakters">
    <xsl:output-character character="&lt;" string="&lt;" />
    <xsl:output-character character="&gt;" string="&gt;" />
    <xsl:output-character character="&amp;" string="&amp;" />
  </xsl:character-map>
  <xsl:output indent="yes" use-character-maps="xml_karakters" method="xml" />
  <xsl:template match="/">
    <uitvoer>
      <xsl:text>Als a > b &amp; b > c dan geldt: c &lt; a</xsl:text>
    </uitvoer>
  </xsl:template>
</xsl:stylesheet>
```

Merk op dat het attribuut disable-output-escaping nu ontbreekt in het <xsl:text> element. Output escaping is nu voorkomen door met de character-map af te dwingen dat de karakters >, < en & ongewijzigd in de uitvoer terecht moeten komen. Controleer dit door in het xsl:output element het attribuut use-character-maps="xml_karakters" weg te laten en de transformatie opnieuw uit te voeren.

3.3.3 next-match

In paragraaf 2.4.2 hebben we de voor- en nadelen van de pull en de push benadering van XSLT transformaties besproken. De push benadering gaat uit van templates: als de stylesheet een node tegenkomt in het bron-document wordt het meest specifieke matchende template daarop toegepast.

Soms zou het daarbij handig zijn als meerdere matchende templates worden toegepast. Vanaf XSLT 2.0 kan dit met behulp van het `<next-match/>` element. Dit lege element roept het volgende template aan dat voor dezelfde node van toepassing is.

```
<?xml version="1.0" encoding="utf-8"?>
<!--voer de transformatie uit met het document next-match.xml -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="/">
  <html>
   <body>
    <table border="1">
     <tr><th>Leden:</th></tr>
     <xsl:apply-templates />
    </table>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="lid[@datum='2003-10-03']">
  <tr><td><xsl:apply-templates /> (vanaf begin)</td></tr>
 </xsl:template>

 <xsl:template match="lid[substring(@datum,1,4) = '2010']">
  <tr><td><xsl:apply-templates /> (nieuw)</td></tr>
 </xsl:template>

 <xsl:template match="lid">
  <tr><td><xsl:apply-templates /></td></tr>
 </xsl:template>
</xsl:stylesheet>
```

Deze template genereert een HTML tabel met leden van een club. Dit is een voorbeeld van de push benadering in XSLT 1.0: we zien drie verschillende templates, die alle drie een rij uit de tabel genereren. In twee ervan gebeurt nog iets extra's: de tekst "(vanaf begin)" of "(nieuw)" komt achter de naam van het lid te staan.

Omdat voor elk <lid> element maar één van deze drie templates wordt toegepast, moet in alle drie de templates dezelfde code worden herhaald, namelijk het genereren van <tr> en <td> elementen. De stylesheet wordt beter onderhoudbaar als het genereren van deze elementen maar in één template hoeft te worden opgenomen.

In XSLT 2.0 kunnen we <xsl:next-match> gebruiken.
De onderste drie templates zien er nu als volgt uit:

```
<xsl:template match="lid[@datum='2003-10-03']">
 <xsl:next-match /> (vanaf begin)
</xsl:template>

<xsl:template match="lid[substring(@datum,1,4) = '2010']">
 <xsl:next-match /> (nieuw)
</xsl:template>

<xsl:template match="lid">
 <tr><td><xsl:next-match /></td></tr>
</xsl:template>
```

Nu is er nog één template waarin <tr> en <td> elementen worden gegenereerd. Het idee is nu als volgt:

- voer het algemene template uit: zet dus vast <td> en <tr> elementen neer;
- waar <xsl:next-match /> staat wordt gekeken of er ook een andere template van toepassing is
- bestaat er zo'n ander template, voer daar dan de code van uit
- bestaat zo'n template niet, voer dan de default template uit (zie paragraaf 2.4.1)

Het werkt echter nog niet naar behoren: eerst worden de meest specifieke templates uitgevoerd, en dan pas daarna de algemene. We krijgen daardoor zulke uitvoer:

```
<tr><td>Wim de Boer</td></tr> (vanaf begin)
```

terwijl we dit willen:

```
<tr><td>Wim de Boer (vanaf begin)</td></tr>
```

Om de volgorde aan te passen kunnen we het attribuut *priority* meegeven aan een template. De normale prioriteit van een template zit ergens tussen –0.5 en +0.5. Een hoger getal geeft een hogere prioriteit.

We voegen aan het template met match="lid" daarom een attribuut priority toe, met de waarde 1:

```
<xsl:template match="lid" priority="1">
  <tr><td><xsl:next-match /></td></tr>
</xsl:template>
```

Het aangepaste template wordt nu als eerste uitgevoerd, waardoor de tabel nu wel correct wordt gegenereerd:

Leden:
Bert de Jong
Wim de Boer (vanaf begin)
Ellen de Vries (vanaf begin)
Lisanne de Vries (vanaf begin)
Bert Veenstra (vanaf begin)
Marcel Buist (nieuw)
Anita Poortinga (vanaf begin)
Fabian Hermans (nieuw)
Lisa Smit (nieuw)
Robert Breukelman
Martien Jenninga

3.3.4 perform-sort

Het element perform-sort wordt gebruikt om een variabele aan te maken met gesorteerde gegevens. Voordat we daarop ingaan moeten we eerst een verschil tussen XSLT 1.0 en XSLT 2.0 behandelen.
Een variabele kan een losse waarde bevatten, maar ook meerdere nodes. In XSLT 1.0 wordt een variabele met meerdere nodes een "result-tree" genoemd. De functionaliteit daarvan is beperkt: je kunt de tree als geheel uitvoeren met xsl:copy-of, of je kunt er de string inhoud van uitvoeren met xsl:value-of.
In XSLT 2.0 is een variabele met meerdere nodes een "temporary tree": een tijdelijke node-set in het geheugen. Zo'n node-set kan op dezelfde manier doorlopen worden als een gewoon XML document.

In het volgende voorbeeld wordt zo'n variabele aangemaakt en gelijk gesorteerd. Het element <xsl:perform-sort> kan zo'n sortering uitvoeren zonder dat daar <xsl:for-each> of <xsl:apply-templates> voor nodig is.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:cd="http://www.cd.nl" exclude-result-prefixes="cd">

    <xsl:template match="/cd:cdlijst">

        <xsl:variable name="lijst">
            <xsl:perform-sort select="cd:cd" >
            <xsl:sort select="cd:cdtitel"></xsl:sort>
            </xsl:perform-sort>
        </xsl:variable>

        <cdlijst>
            <originele_volgorde>
                <xsl:value-of select="cd:cd/cd:cdtitel" separator="; "></xsl:value-of>
            </originele_volgorde>
            <gesorteerd>
                <xsl:value-of select="$lijst/cd:cd/cd:cdtitel" separator="; "></xsl:value-of>
            </gesorteerd>
        </cdlijst>

    </xsl:template>
</xsl:stylesheet>
```

Het <xsl:perform-sort> element heeft een select attribuut dat de nodes ophaalt die gesorteerd zullen worden. In dit geval bevat de variabele lijst dus cd:cd nodes. Met één of meer <xsl:sort> elementen binnen het <xsl:perform-sort> element wordt de nieuwe volgorde aangegeven.

3.3.5 result-document

Sinds XSLT 2.0 is het mogelijk om het resultaat van een transformatie naar meerdere documenten uit te voeren. We gebruiken daarvoor het <xsl:result-document> element. Dit element heeft een attribuut href, vergelijkbaar met het href attribuut van een HTML link, waarmee de locatie van het nieuwe document wordt opgegeven. Dit kan een relatief pad of een absoluut pad zijn. In het volgende voorbeeld wordt voor elke cd een apart XML bestand aangemaakt, waarbij de bestandsnaam gebaseerd is op de naam van de artiest:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:cd="http://www.cd.nl">

    <xsl:template match="/cd:cdlijst">

        <xsl:for-each select="cd:cd">
            <xsl:variable name="bestandsnaam">
                <xsl:value-of select="concat('///h:/artiest/' , cd:artiest, '.xml')"></xsl:value-of>
            </xsl:variable>

            <xsl:result-document href="{$bestandsnaam}">
                <xsl:copy-of select="."/>
            </xsl:result-document>
        </xsl:for-each>

    </xsl:template>
</xsl:stylesheet>
```

3.3.6 sequence

Het <xsl:sequence> element is bedoeld om een sequence in het geheugen aan te maken, bijvoorbeeld om een variabele mee te vullen voor latere verwerking. Een sequence is een serie waarden, of nodes zonder parent-node.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs">
  <xsl:output indent="yes"/>
    <xsl:template match="/">

        <xsl:variable name="nummers" as="xs:integer *">
            <xsl:sequence select="1 to 4"></xsl:sequence>
            <xsl:sequence select="(11, 13, 17)"></xsl:sequence>
            <xsl:for-each select="1 to 3">
              <xsl:sequence select="20 + 2 * ."></xsl:sequence>
            </xsl:for-each>
        </xsl:variable>

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

    </xsl:template>
</xsl:stylesheet>
```

Hier wordt een variabele aangemaakt, met een nieuw attribuut: as="xs:integer *". Vanaf XSLT 2.0 kan van een variabele worden aangegeven wat het XMLSchema-type is. Bij het uitvoeren van de transformatie wordt gecontroleerd of de inhoud wel voldoet aan het type. In dit geval is het type dus integer. Integer is een *atomic* type, een "ondeelbaar" type, ter onderscheiding van de element-typen die hierna worden behandeld.
Achter xs:integer staat een asterisk (*). Dit geeft aan dat de variabele 0 of meer integers bevat. Alternatieven zijn + (1 of meer), of ? (0 of 1). Als er niks achter staat wordt ervan uitgegaan dat het één waarde is. Andere *atomic* typen zijn onder meer xs:date, xs:time, xs:float, xs:boolean of xs:string.

Het element <xsl:sequence> wordt hierin drie keer gebruikt, om samen een reeks getallen aan te maken.

- Het eerste element krijgt select="1 to 4" mee, waarmee de reeks 1,2,3,4 wordt aangemaakt.
- In het tweede element staat select="(11, 13, 17)", waarmee de reeks 11, 13, 17 wordt aangemakt.
- Het laatste element staat in een <xsl:for-each> loop, die achtereenvolgens de waarden 1, 2 en 3 bevat. Het <xsl:sequence> element doet daar een bewerking op: select="20 + 2 * .". De punt slaat op de huidige waarde. Elk van de drie getallen wordt dus met 2 vermenigvuldigd, en bij 20 opgeteld, om zo de waarden 22, 24 en 26 aan te maken.

Een sequence kan ook gebruikt worden om een reeks nodes samen te stellen. Het type wordt hierbij samengesteld uit het node-type en (optioneel) de naam van het node-type (het type kan nog verder worden gespecificeerd, maar dat valt buiten het bestek van deze cursus). Enkele voorbeelden:

type	beschrijving
element()+	1 of meer elementen
node()	een node
attribute()?	een optioneel attribuut
element(cd:cd)*	0 of meer elementen van het type cd
item()*	0 of meer nodes of atomic types

Een bijzondere eigenschap van een sequence met nodes van een node-type, is dat er geen kopie wordt gemaakt in het geheugen (zoals bij <xsl:copy> en <xsl:copy-of>), maar dat naar de oorspronkelijke node in de boomstructuur wordt verwezen. Dat betekent ook dat vanaf de betreffende node met XPath expressies naar een volgende, vorige, of bovenliggende node kan worden verwezen. We zullen dat in het volgende voorbeeld toelichten.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:cd="http://www.cd.nl" xmlns:xs="http://www.w3.org/2001/XMLSchema"
   exclude-result-prefixes="xs">
   <xsl:output indent="yes" />
   <xsl:template match="/">
      <xsl:variable name="cop_voorbeeld" as="element()">
         <xsl:copy-of select="cd:cdlijst/cd:cd[1]/cd:cdtitel" />
      </xsl:variable>
      <xsl:variable name="seq_voorbeeld" as="element()">
         <xsl:sequence select="cd:cdlijst/cd:cd[1]/cd:cdtitel" />
      </xsl:variable>
      <uitvoer>
         <copy-of>
            <titel>
               <xsl:value-of select="$cop_voorbeeld" />
            </titel>
            <artiest>
               <xsl:value-of select="$cop_voorbeeld/../cd:artiest" />
            </artiest>
         </copy-of>
         <sequence>
            <titel>
               <xsl:value-of select="$seq_voorbeeld" />
            </titel>
            <artiest>
               <xsl:value-of select="$seq_voorbeeld/../cd:artiest" />
            </artiest>
         </sequence>
      </uitvoer>
   </xsl:template>
</xsl:stylesheet>
```

Bovenaan worden twee variabelen aangemaakt, die beide de titel bevatten van de eerste cd. De eerste variabele bevat een kopie, de tweede is als een sequence aangemaakt, en bevat dus een verwijzing naar het originele element zelf.

In de uitvoer worden beide titels getoond. Om ook de artiest te tonen moet een uitstapje naar de parent node worden gemaakt: select="$cop_voorbeeld**/../cd:artiest**" en select="$seq_voorbeeld**/../cd:artiest**".

We zien na de transformatie dat cd:artiest niet bereikbaar is vanaf de variabele $cop_voorbeeld: het element cd:cdtitel is gekopieerd, en heeft daarmee geen parent node. Vanaf de variabele $seq_voorbeeld kunnen we wel bij cd:artiest. Deze variabele verwijst naar het originele element in de boomstructuur, zodat daarvan wel de parent node beschikbaar is.

3.4 Groeperen

Een belangrijke toevoeging binnen XSLT 2.0 is het groeperen. Het is mogelijk om elementen op basis van een eigenschap te groeperen. Hiervoor wordt het element `<xsl:for-each-group>` gebruikt. Dit element heeft een attribuut `select`, waarmee we aangeven wat we willen groeperen. Hoe de groepen worden samengesteld wordt bepaald door het attribuut group-by.

In het volgende voorbeeld worden cd elementen gegroepeerd op basis van het sub-element land:

```
<xsl:for-each-group select="cd" group-by="land">
```

We zien bij het transformeren dat elke waarde bij land slechts één keer voor komt.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output indent="yes"/>
   <xsl:template match="lijst">
      <landen>
         <xsl:for-each-group select="cd" group-by="land">
            <land>
               <code>
                  <xsl:value-of select="land"/>
               </code>


         </land>
         </xsl:for-each-group>
      </landen>
   </xsl:template>
</xsl:stylesheet>
```

Als er groepen gemaakt zijn willen we vaak meer informatie over de groep weten bijvoorbeeld welke elementen er in de groep zitten. Hiervoor hebben we de functie `current-group()` nodig. Deze functie haalt alle nodes van de huidige groep op. Daarop kunnen we weer groepsfuncties loslaten, bijvoorbeeld om het aantal nodes en de totale prijs te berekenen, zoals in het volgende voorbeeld:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output indent="yes"/>
   <xsl:template match="lijst">
      <landen>
         <xsl:for-each-group select="cd" group-by="land">
            <land>
               <code>
                  <xsl:value-of select="land"/>
               </code>
               <artiesten>
                  <xsl:copy-of select="current-group()/artiest"/>
               </artiesten>
               <aantal>
                  <xsl:copy-of select="count(current-group())"/>
               </aantal>
               <totaal>
                  <xsl:copy-of select="sum(current-group()/prijs)"/>
               </totaal>
            </land>
         </xsl:for-each-group>
      </landen>
   </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
- <landen>
  - <land>
      <code>USA</code>
    - <artiesten>
        <artiest>Bob Dylan</artiest>
        <artiest>Many</artiest>
        <artiest>Will Smith</artiest>
        <artiest>Joe Cocker</artiest>
        <artiest>Percy Sledge</artiest>
        <artiest>Dolly Parton</artiest>
        <artiest>Otis Redding</artiest>
      </artiesten>
      <aantal>7</aantal>
      <totaal>65.78999999999999</totaal>
    </land>
  - <land>
      <code>UK</code>
    + <artiesten>
      <aantal>13</aantal>
      <totaal>116.85000000000002</totaal>
    </land>
  - <land>
      <code>EU</code>
    + <artiesten>
      <aantal>5</aantal>
      <totaal>46.650000000000006</totaal>
    </land>
  - <land>
      <code>Norway</code>
    + <artiesten>
      <aantal>1</aantal>
      <totaal>7.9</totaal>
    </land>
</landen>
```

We zien dat per land het aantal cd's wordt geteld en de totale prijs van deze cd's wordt weergegeven.

Bij het optellen van getallen zien we soms een vreemde afrondingsfout optreden. Bijvoorbeeld 0.5 wordt 0.49999999999999999999. Dit is een vaker voorkomend probleem bij verschillende programmeertalen. U kunt dit oplossen door met de functie round() het getal af te ronden.

Naast de functie current-group() bestaat er ook de functie current-grouping-key() die de sleutelwaarde retourneert van de huidige group.

3.4.1 Andere mogelijkheden met groeperen

Het element `<xsl:for-each-group>` kent nog een aantal attributen. Met deze opties kunnen we sturen hoe er gegroepeerd moet worden. Deze attributen komen in de plaats van het attribuut `group-by`.

We kennen de volgende vier mogelijkheden:

Attribuut	Omschrijving
group-by	Met group-by wordt aangegeven dat alle elementen gegroepeerd moeten worden op basis van de opgegeven eigenschap. Alle elementen worden eerst gesorteerd en vervolgens gegroepeerd.
group-adjacent	Met het group-adjacent wordt aangegeven dat alle elementen gegroepeerd moeten worden op basis van de opgegeven eigenschap. Alleen wordt elke keer dat er een andere waarde wordt gevonden een nieuwe groep gemaakt. De elementen worden niet eerst gesorteerd maar alleen gegroepeerd.
group-starting-with	Groepeert alle elementen en begint een nieuwe groep zodra het opgegeven element gevonden wordt.

Attribuut	Omschrijving
group-ending-with	Groepeert alle elementen en eindigt een groep zodra het opgegeven element gevonden wordt.

3.5 XPath 2.0

Met XSLT 2.0 is er ook een nieuwe versie van XPath ontwikkeld. XPath 2.0 is vooral een uitbreiding op XPath 1.0. Zo kan in XPath 2.0 van veel meer verschillende datatypen gebruik worden gemaakt. Alle standaard datatypen die binnen xml schema's gebuikt worden kunnen nu ook in XPath expressies worden gebruikt. Daarnaast is het mogelijk om zelf gedefinieerde typen uit een schema te gebruiken binnen XPath expressies. Verder zijn er meer functies beschikbaar en kunnen we zelf nieuwe functies maken.
Behalve uitbreidingen zijn er ook een paar verschillen tussen XPath 1.0 en XPath 2.0. Zo werken XPath expressies in XPath 2.0 veelal met sequences in plaats van node-sets. Dit heeft een aantal voordelen. Verder is het mogelijk om variabelen te gebruiken in XPath 2.0 expressies.

3.5.1 XPath sequences

Het werken met XPath 2.0 expressies is anders dan in XPath 1.0, omdat een XPath 2.0 expressie nu altijd een sequence oplevert Een sequence is een geordende lijst van 0 of meer items. Hier vinden we een belangrijk verschil met XPath 1.0. Een XPath 1.0 expressie levert namelijk een node of een ongeordende node-set. Een sequence kan waarden bevatten maar ook een lijst van nodes. Dit geeft meer controle en flexibiliteit bij de verwerking van expressies. Bijzondere sequences zijn de *empty sequence*, als een sequence geen items bevat, en de *singleton sequence*, als de sequence precies één item bevat.

We hebben eerder gezien hoe sequences kunnen worden opgebouwd: door de betreffende gegevens op te halen, bijvoorbeeld met <xsl:value-of> of door ze handmatig aan te maken.

Er zijn ook diverse functies en operatoren beschikbaar om sequences te bewerken. De volgende functies kunnen op sequences van nodes en van atomic types worden toegepast:

functie	**beschrijving**	**voorbeeld**	**uitvoer**
index-of	geeft de posities aan van gezocht item in een sequence	*index-of((1,3,5,3),3)*	*(2, 4)*
remove	verwijdert item op aangegeven positie	*remove((1,3,5),3)*	*(1, 3)*
empty	test of een sequence leeg is	*empty(1,3)*	*false*
exists	test of een sequence *niet* leeg is	*exists(1,3)*	*true*
distinct-values	geeft de unieke waarden uit een sequence	*distinct-values(1,3,3)*	*(1,3)*
reverse	draait de volgorde van items in een sequence om	*reverse(2,4,6)*	*(6,4,2)*
subsequence	haalt een deel van een sequence op, vanaf een startpositie, met een bepaalde lengte	*subsequence((1,2,3,4,5), 2, 3)*	*(2,3,4)*
unordered	retourneert de items in	*unordered(//titel)*	*een reeks titels*

	applicatie-afhankelijke (willekeurige) volgorde		

Daarnaast zijn er enkele groepsfuncties die op sequences kunnen worden toegepast. De met een asterisk (*) aangeduide functies zijn nieuw voor XPath 2.0.

functie	beschrijving	voorbeeld	uitvoer
count	geeft het aantal items van de opgegeven sequence	*count((1,3,5,3))*	*4*
min*	geeft de laagste waarde van de opgegeven sequence	*min((1,3,5,3))*	*1*
max*	geeft de hoogste waarde van de opgegeven sequence	*max((1,3,5,3))*	*5*
sum	geeft het totaal van de opgegeven sequence	*sum((1,3,5,3))*	*12*
avg*	geeft het gemiddelde van de opgegeven sequence	*avg((1,3,5,3))*	*3*

Aan deze nieuwe XPath functies is ook een nieuwe namespace gekoppeld, die default met de prefix fn wordt aangeduid: `xmlns:fn="http://www.w3.org/2005/xpath-functions"`.
Naast deze functies is er ook een aantal set operatoren beschikbaar om combinaties van sequences te maken. Dit betreft altijd sequences van nodes, niet van atomic items.

operator	beschrijving	voorbeeld	uitvoer
| of ,	geeft de union van twee sequences	*((item1, item2) | (item3))* of *((item1, item2), (item3))*	*item1, item2, item3*
intersect	geeft de overlap tussen twee sequences	*(item1, item2, item3) intersect (item1, item2, item4)*	*item1, item2*
except	geeft het het verschil tussen twee sequences	*(item1, item2, item3) except (item1, item2, item4)*	*item3*

De volgende stylesheet toont enkele maniere waarop sequences kunnen worden samengesteld:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output indent="yes" />
   <xsl:template match="/">
      <sequences>
         <sequence>
            <xsl:sequence select="(//cd[jaar='1973']/titel, //cd[land='EU']/titel)" />
         </sequence>
         <sequence>
            <xsl:sequence select="//cd[number(jaar)>1990]/(land|jaar) except (//cd[land ='UK']/(land|jaar))" />
         </sequence>
         <sequence>
            <xsl:sequence select="//cd[number(jaar)>1990]/(land|jaar) intersect (//cd[land ='UK']/(land|jaar))" />
         </sequence>
         <sequence>
            <xsl:value-of select="( (1 , 3, 4), ('aap', 'noot', 'mies'), (xs:date('2011-05-30'),xs:date('2011-05-31')))" />
         </sequence>
         <sequence>
            <xsl:value-of select="fn:remove (5 to 10,  3)" />
         </sequence>
         <sequence>
            <xsl:sequence select="fn:index-of ((5 to 10),  8)" />
         </sequence>
         <sequence>
            <xsl:sequence select="fn:insert-before ((5 to 8),  2, (1,3))" />
         </sequence>
         <sequence>
            <xsl:sequence select="(fn:sum((1,4,7)), fn:avg((1,4,7)), fn:min((1,4,7)), fn:max((1,4,7)), fn:count((1,4,7)))" />
         </sequence>
      </sequences>
   </xsl:template>
</xsl:stylesheet>
```

Merk op dat voor het aanmaken van een datum-object in dit geval geen XPath functie wordt gebruikt, maar van een constructor die bij XMLSchema hoort, zoals xs:date('2011-05-31'). In XSLT2.0 zullen we beide prefixes (fn en xs) dus regelmatig tegenkomen.

3.5.2 Andere XPath 2.0 functies

In het de vorige paragraaf is al een aantal XPath functies besproken met betrekking tot sequences. Het volgende overzicht toont een selectie uit andere XPath functies, gerangschikt naar soort.

De met een asterisk (*) aangeduide functies zijn nieuw voor XPath 2.0.

numerieke functies			
functie	**beschrijving**	**voorbeeld**	**uitvoer**
number	geeft de numerieke waarde	number('100')	100
abs*	geeft de absolute waarde	abs(-100)	100
ceiling	geeft het getal naar boven afgerond	ceiling(3.6)	4
floor	geeft het getal naar beneden afgerond	floor(3.6)	3
round	geeft het getal afgerond naar dichtstbijzijnde gehele getal	round(3.5)	4
round-half-to-even	geeft het getal afgerond naar dichtstbijzijnde even getal	round-half-to-even(2.6)	2

string functies			
functie	**beschrijving**	**voorbeeld**	**uitvoer**
concat	plakt strings aan elkaar	concat('a','b','c')	'abc'
string-join*	als concat, maar met optionele string als separator	string-join(('a','b','c'),' - ')	'a - b - c'
substring	geeft het deel van de string vanaf een beginpositie met een bepaalde lengte (als lengte niet wordt meegegeven: tot het eind)	substring('abcdefg',2,3)	'bcd'
string-length	geeft de lengte van de string	string-length('abcdefg')	7
upper-case*	geeft de string in hoofdletters	upper-case('abcdefg')	'ABCDEFG'
lower-case*	geeft de string in kleine letters	lower-case('ABCDEFG')	'abcdefg'
normalize-space	verwijdert whitespace (spaties, tabs e.d.) voor en na de string, en vervangt whitespace tussen strings door een spatie.	normalize-space(' hier staan veel spaties ')	'hier staan veel spaties'
substring-before	geeft het deel van de string voor een opgegeven string	substring-before('abcdefg','d')	'abc'
substring-after	geeft het deel van de string na een opgegeven string	substring-after('abcdefg','d')	'efg'

string functies			
functie	**beschrijving**	**voorbeeld**	**uitvoer**
starts-with	test of een string begint met de opgegeven string	starts-with('abcdefg','abc')	true
ends-with*	test of een string eindigt met de opgegeven string	ends-with('abcdefg','abc')	false
contains	test of de opgegeven string voorkomt in een string	contains('abcdefg','abc')	true
string-pad*	herhaalt de string een opgegeven aantal keer	string-pad('xyz',2)	'xyzxyz'
matches*	controleert of het opgegeven patroon voorkomt in een string	matches(telefoon,'\d{3,5}-,\d*')	true
translate	vervangt in een string de tekens uit de ene string door overeenkomstige tekens uit de andere string	translate('12:30','0123','abcd')	'bc:da'
tokenize*	breekt een string op in delen, met het opgegeven patroon als separator	tokenize('dit is een voorbeeld','/s+')	('dit', 'is', 'een', 'voorbeeld)

Nieuw voor XPath zijn de datum functies:

datum functies			
functie	**beschrijving**	**voorbeeld**	**uitvoer**
current-date*	huidige datum (inclusief tijdzone)	current-date()	2011-05-11+02:00
current-dateTime*	huidige datum en tijd	current-dateTime()	2011-05-11T10:54:32.792+02:00
current-time*	huidige tijd	current-time()	10:54:32.792+02:00
dateTime*	opgegeven datum inclusief tijd	dateTime("xs:date('2011-05-11'), xs:time('11:00:00+02:00')")	2011-05-11T11:00:00+02:00

Er zijn nog veel meer datum functies, die onderdelen uit een datum, een datum met tijd, of een *duration* halen. Een duration, van het type xs:dayTimeDuration, is het verschil tussen twee datums (al dan niet met tijd). Deze wordt weergegeven als:

P<jaar>Y><maanden>M<dagen>DT<uren>H<minuten>M<seconden>S
bijvoorbeeld: P0Y0M6DT22H21M30S

Er zijn twee subtypes van xs:duration waarmee in XPath 2.0 gerekend kan worden: xs:dayTimeDuration en xs:yearMonthDuration. De onderdelen die niet gebruikt worden zijn niet verplicht. Zo kunnen we een duration van 2 dagen schrijven als xs:dayTimeDuration('P2D'). We kunnen bijvoorbeeld op de volgende manier voor de komende 7 dagen onderzoeken welke datum het wordt:

```
<?xml version="1.0" encoding="utf-8"?>
<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" exclude-result-prefixes="xs fn">
    <xsl:output method="text" />
    <xsl:template match="/">
        <xsl:for-each select="1 to 7">
            <xsl:value-of select="fn:current-date()+xs:dayTimeDuration('P1D') * . "/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

Hier volgt een lijst met andere datumfuncties. De functionaliteit wordt duidelijk uit de naamgeving:

Meer datum functies		
years-from-duration*	year-from-dateTime*	year-from-date*
months-from-duration*	month-from-dateTime*	month-from-date*
days-from-duration*	day-from-dateTime*	day-from-date*
minutes-from-duration*	minutes-from-dateTime*	minutes-from-time*
seconds-from-duration*	seconds-from-dateTime*	seconds-from-time*

Het volgende voorbeeld toont hoe enkele datumfuncties gebruikt kunnen worden:

```
<?xml version="1.0" encoding="utf-8"?>
<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" exclude-result-prefixes="xs fn">
    <xsl:output indent="yes" />
    <xsl:template match="/">
        <datetime_functies>
            <huidige_datum>
                <xsl:value-of select="fn:current-date()" />
            </huidige_datum>
            <huidige_tijd>
                <xsl:value-of select="fn:current-time()" />
            </huidige_tijd>
            <ingestelde_datum>
                <xsl:value-of select="fn:dateTime(xs:date('2011-05-11'),xs:time('11:00:00+02:00'))" />
            </ingestelde_datum>
            <huidige_jaar>
                <xsl:value-of select="fn:year-from-dateTime(fn:current-dateTime())" />
            </huidige_jaar>
            <duration>
                <xsl:value-of select="fn:dateTime(xs:date('2011-05-14'),xs:time('12:20:00+02:00')) - fn:current-dateTime()" />
            </duration>
            <dagen_verschil>
                <xsl:value-of select="fn:days-from-duration(xs:date('2012-01-01')-fn:current-date())" />
            </dagen_verschil>
            <volgende_week>
                <xsl:value-of select="fn:current-date() + xs:dayTimeDuration('P7D') " />
            </volgende_week>
        </datetime_functies>
    </xsl:template>
</xsl:stylesheet>
```

De uitvoer zal er dan ongeveer zo uitzien (afhankelijke van de huidige datum en tijd):

```
<?xml version="1.0" encoding="UTF-8"?>
<datetime_functies>
  <huidige_datum>2011-05-18+02:00</huidige_datum>
  <huidige_tijd>14:57:11.108+02:00</huidige_tijd>
  <ingestelde_datum>2011-05-11T11:00:00+02:00</ingestelde_datum>
  <huidige_jaar>2011</huidige_jaar>
  <duration>-P4DT2H37M11.108S</duration>
  <dagen_verschil>228</dagen_verschil>
  <volgende_week>2011-05-25+02:00</volgende_week>
</datetime_functies>
```

3.5.3 XPath 2.0 operatoren

In het vorige hoofdstuk hebben we behandeld hoe node-set vergelijkingen werken. Nog even ter herinnering:

Twee node-sets zijn gelijk als er in beide node-sets een node voorkomt met gelijke string-waarden. Twee node-sets zijn ongelijk als er in beide node-sets een node voorkomt met ongelijke string-waarden.

XPath 2.0 heeft de standaard operatoren uitgebreid met operatoren die alleen true opleveren als één node met één waarde wordt vergeleken. Het gaat hierbij om de volgende operatoren:

Operator	beschrijving	Voorbeeld
eq	Is gelijk aan	aantal eq 10
ne	Is ongelijk aan	aantal ne 10
lt	Kleiner dan	aantal lt 10
le	Kleiner of gelijk aan	aantal le 10
gt	Groter dan	aantal gt 10
ge	Groter of gelijk aan	aantal ge 10

Bij een vergelijking als `aantal eq 10` wordt een foutmelding gegeven als aantal een sequence is van meer dan één node.

3.5.4 Uitbreidingen op de taal

Naast de genoemde operatoren en functies zijn er nog enkele geavanceerde uitbreidingen op de XPath taal. Hierin wordt in een expressie een variabele aangemaakt, waar verderop in deze expressie naar wordt verwezen.

De eerste hiervan is de for loop. Deze heeft de volgende syntax:

```
for $variabele in <items> return $variabele/<pad>
```

Hier wordt dus een variabele aangemaakt op basis van een expressie. Met return wordt aangegeven wat hiervan moet worden uitgevoerd.

De andere twee testen of sommige of alle items in een sequence voldoen aan een voorwaarde:

```
some $variabele in <items> satisfies <expressie>
every $variable in <items> satisfies <expressie>
```

In het volgende voorbeeld wordt met behulp van de for loop de totale prijs berekend van bestellingen door per product het aantal*prijs te berekenen, en deze uitkomsten bij elkaar op te tellen.

```
<?xml version="1.0" encoding="utf-8"?>
<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" exclude-result-prefixes="xs fn">
   <xsl:output indent="yes" />
   <xsl:template match="/">
      <xpath_uitbreidingen>
         <aantal_prijs>
         <xsl:sequence select="for $product in //product return ($product/aantal * $product/prijs)"></xsl:sequence>
         </aantal_prijs>
      <totaal>
         <xsl:sequence select="sum(for $prd in //product return ($prd/aantal * $prd/prijs))"></xsl:sequence>
      </totaal>
       <some>
         <xsl:value-of select="some $product in //product satisfies ($product/aantal * $product/prijs > 300)"></xsl:value-of>
      </some>
      <every>
         <xsl:value-of select="every $product in //product satisfies ($product/aantal * $product/prijs > 300)"></xsl:value-of>
      </every>
      </xpath_uitbreidingen>
   </xsl:template>
</xsl:stylesheet>
```

Met some en every wordt nagegaan of voor sommig of elk product de prijs * aantal groter is dan 300.

3.6 Zelf functies maken

Zoals vermeld is er in XSLT 2.0 een groot aantal functies beschikbaar. Toch komt het regelmatig voor dat we extra functionaliteit nodig hebben. Het is met XSLT 2.0 vrij eenvoudig om zelf nieuwe functies te maken. Hiervoor is het element `<xsl:function>` beschikbaar. Om een functie te kunnen maken moet deze in een namespace staan. Dit mag een bestaande namespace zijn maar meestal wordt een aparte namespace voor eigen functies gebruikt. In de naam van de functie moet dan de prefix van deze namespace staan. Meestal willen we aan een functie argumenten (parameters) mee geven. Dit kan met het element `<xsl:param>`. Vervolgens laten we de functie een aantal instructies uitvoeren, waarna we het resultaat van de functie teruggeven door middel van het element `<xsl:value-of>`.

In het volgende voorbeeld is bovenaan een functie rondaf_2dec gemaakt, in de namespace www.mijnfuncties.nl. Deze functie krijgt een paramater mee en retourneert in ingevoerde getal, afgerond op twee decimalen:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:mf="www.mijnfuncties.nl" extension-element-prefixes="mf">
    <xsl:output indent="yes"/>
    <xsl:function name="mf:rondaf_2dec">
        <xsl:param name="getal"/>

        <xsl:value-of select="format-number(round($getal * 100) div 100, '€ #.00')"/>
    </xsl:function>


    <xsl:template match="lijst">
        <landen>
            <xsl:for-each-group select="cd" group-by="land">
                <land>
                    <code>
                        <xsl:value-of select="land"/>
                    </code>
                    <aantal>
                        <xsl:value-of select="count(current-group())"/>
                    </aantal>
                    <totaal>
                        <xsl:value-of select="mf:rondaf_2dec(sum(current-group()/prijs))"/>
                    </totaal>
                </land>
            </xsl:for-each-group>
        </landen>
    </xsl:template>
</xsl:stylesheet>
```

Om te voorkomen dat de namespace van de functies in het resultaat document terecht komt kan het attribuut `extension-element-prefixes` aan het element `<xsl:stylesheet>` worden toegevoegd. Als waarde wordt de prefix van de namespace meegegeven

Het is gebruikelijk om alle eigen functies samen in één stylesheet opnemen. In een nieuw stylesheet kunnen we de stylesheet met functies importeren. Wel zal de namespace van de te importeren functies ook in het huidige stylesheet opgenomen moeten worden.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:myf="www.mijnfuncties.nl" extension-element-prefixes="mf" >
    <xsl:import href="///H:/XML2Bestanden/mijnfuncties.xsl"></xsl:import>

    <xsl:output indent="yes"/>

    <xsl:template match="lijst">
        <landen>
            <xsl:for-each-group select="cd" group-by="land">
                <land>
                    <code>
                        <xsl:value-of select="land"/>
                    </code>
                    <aantal>
                        <xsl:value-of select="count(current-group())"/>
                    </aantal>
                    <totaal>
                        <xsl:value-of select="mf:rondaf_2dec(sum(current-group()/prijs))"/>
                    </totaal>
                </land>
            </xsl:for-each-group>
        </landen>
    </xsl:template>
</xsl:stylesheet>
```

Op internet zijn veel XPath functies te vinden. U kunt in uw eigen functie stylesheet opnemen. Ook kunt u deze functies binnen uw eigen functies aanroepen.

Appendix A Volledige lijst met xslt elementen

Element	Beschrijving
<xsl:apply-imports>	Past een geïmporteerde stylesheet toe.
<xsl:apply-templates>	Past een template toe op het huidige element of op de child nodes van het huidige element.
<xsl:attribute>	Voegt een attribuut toe aan een element.
<xsl:attribute-set>	Hiermee wordt een attribuut set gedefinieerd.
<xsl:call-template>	Roept een template aan. Dit wordt vaak gebruikt in combinatie met *<xsl:with-param>*.
<xsl:choose>	Wordt gebruikt voor een keuze op basis van een test. Binnen dit element worden *<xsl:when>* en *<xsl:otherwise>* gebruikt om de keuzen te specificeren.
<xsl:comment>	Voegt een commentaar element aan het resultaat toe.
<xsl:copy>	Maakt een kopie van het huidige element. (exclusief child elementen, attributen en inhoud)
<xsl:copy-of>	Maakt een kopie van het huidige element. (inclusief child elementen, attributen en inhoud)
<xsl:decimal-format>	Specificeert een decimale notatie. Maakt gebruik van de functie *format-number()*.
<xsl:element>	Definieert een element in het resultaat.
<xsl:fallback>	Specificeert een alternatieve code die wordt uitgevoerd als het xslt element niet wordt ondersteund door de processor.
<xsl:for-each>	Wordt gebruikt om een serie nodes te doorlopen.
<xsl:if>	Wordt gebruikt om code op basis van een test wel of niet uit te voeren.
<xsl:import>	Importeert een bestaand stylesheet in een ander stylesheet. (Het geïmporteerde stylesheet heeft een lagere prioriteit.)
<xsl:include>	Importeert een bestaand stylesheet in een ander stylesheet. (Beide stylesheet hebben dezelfde prioriteit.)
<xsl:key>	Definieert een sleutel waarde die kan worden gebruikt in een stylesheet. Aan deze sleutel wordt gerefereerd door middel van de functie *key()*.
<xsl:message>	Voegt een bericht toe aan het resultaat. Wordt over het algemeen gebruikt voor foutmeldingen.
<xsl:namespace-alias>	Vervangt de prefix van een namespace in de stylesheet door een andere prefix in het resultaat document.
<xsl:number>	Definieert de opmaak van het nummer van de huidige node. Wordt ook gebruikt voor de opmaak van getallen.
<xsl:otherwise>	Dit element is een child element van *<xsl:choose>* en definieert wat moet worden gedaan als geen van de tests een TRUE oplevert.
<xsl:output>	Definieert het type van het resultaat document.
<xsl:param>	Declareert een globale parameter in de stylesheet.
<xsl:preserve-space>	Geeft aan dat een waarde bestaand uit spaties moet worden overgenomen.

Element	Beschrijving
<xsl:processing-instruction>	Schrijft een processing instruction naar het resultaat document.
<xsl:sort>	Sorteert het resultaat van een *<xsl:foreach>*.
<xsl:strip-space>	Verwijdert een waarde die enkel uit spaties bestaat.
<xsl:stylesheet>	Het root element van een stylesheet.
<xsl:template>	Binnen dit element worden de regels vast gelegd die moeten worden uitgevoerd voor alle nodes die gevonden worden bij het attribuut match.
<xsl:text>	Wordt gebruikt om een tekst in het resultaat document te zetten.
<xsl:transform>	Het root element van een stylesheet.
<xsl:value-of>	Haalt de waarde van een node op uit het originele xml document.
<xsl:variabele>	Declareert een lokale of globale variabele.
<xsl:when>	Wordt gebruikt binnen een *<xsl:choose>* element om de verschillende keuzes te definiëren. Hierbij wordt gebruik gemaakt van een attribuut test.
<xsl:with-param>	Definieert een waarde van een parameter die wordt gebruikt binnen een template.

Appendix B Syntax XSL 2.0 elementen

Inleiding bijlage

De syntax van alle beschikbare XSLT 2.0 elementen is hieronder weergeven, inclusief de mogelijk attributen. Optionele attributen zijn gemarkeerd met een vraagteken. Ook is aangeven welke elementen als parent element zijn toegestaan.

xsl:analyze-string

Categorie: instruction
Model:

```
<xsl:analyze-string
  select = expression
  regex = { string }
  flags? = { string }>
  <!-- Content: (xsl:matching-substring?, xsl:non-matching-
substring?, xsl:fallback*) -->
</xsl:analyze-string>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is. Een sequence constructor maakt een sequence van nodes of atomic waarden aan, en is de XSLT2 benaming voor een template.
- elk literal result element (een element dat niet tot de xsl-namespace behoort en als uitvoer-element dient).

xsl:apply-imports

Categorie: instruction
Model:

```
<xsl:apply-imports>
  <!-- Content: xsl:with-param* -->
</xsl:apply-imports>
```

Toegestaan *als child element van:*

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:apply-templates

Categorie: instruction
Model:

```
<xsl:apply-templates
  select? = expression
  mode? = token>
  <!-- Content: (xsl:sort | xsl:with-param)* -->
</xsl:apply-templates>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:attribute

Categorie: instruction

Model:

```
<xsl:attribute
  name = { qname }
  namespace? = { uri-reference }
  select? = expression
  separator? = { string }
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:attribute>
```

Toegestaan als child element van:

- `xsl:attribute-set`
- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:attribute-set

Categorie: declaration

Model:

```
<xsl:attribute-set
  name = qname
  use-attribute-sets? = qnames>
  <!-- Content: xsl:attribute* -->
</xsl:attribute-set>
```

Toegestaan *als child element van:*

- `xsl:stylesheet`
- `xsl:transform`

xsl:call-template

Categorie: instruction

Model:

```
<xsl:call-template
  name = qname>
  <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

Toegestaan *als child element van:*

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:character-map

Categorie: declaration

Model:

```
<xsl:character-map
  name = qname
  use-character-maps? = qnames>
  <!-- Content: (xsl:output-character*) -->
</xsl:character-map>
```

Toegestaan *als child element van:*

- `xsl:stylesheet`
- `xsl:transform`

xsl:choose

Categorie: instruction
Model:

```
<xsl:choose>
  <!-- Content: (xsl:when+, xsl:otherwise?) -->
</xsl:choose>
```

Toegestaan *als child element van:*

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:comment

Categorie: instruction
Model:

```
<xsl:comment
  select? = expression>
  <!-- Content: sequence-constructor -->
</xsl:comment>
```

Toegestaan *als child element van:*

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:copy

Categorie: instruction
Model:

```
<xsl:copy
  copy-namespaces? = "yes" | "no"
  inherit-namespaces? = "yes" | "no"
  use-attribute-sets? = qnames
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:copy>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:copy-of

Categorie: instruction
Model:

```
<xsl:copy-of
  select = expression
  copy-namespaces? = "yes" | "no"
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip" />
```

Toegestaan *als child element van:*

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:decimal-format

Categorie: declaration

Model:

```
<xsl:decimal-format
  name? = qname
  decimal-separator? = char
  grouping-separator? = char
  infinity? = string
  minus-sign? = char
  NaN? = string
  percent? = char
  per-mille? = char
  zero-digit? = char
  digit? = char
  pattern-separator? = char />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:document

Categorie: instruction

Model:

```
<xsl:document
  validation? = "strict" | "lax" | "preserve" | "strip"
  type? = qname>
  <!-- Content: sequence-constructor -->
</xsl:document>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:element

Categorie: instruction

Model:

```
<xsl:element
  name = { qname }
  namespace? = { uri-reference }
  inherit-namespaces? = "yes" | "no"
  use-attribute-sets? = qnames
  type? = qname
  validation? = "strict" | "lax" | "preserve" | "strip">
  <!-- Content: sequence-constructor -->
</xsl:element>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:fallback

Categorie: instruction

Model:

```
<xsl:fallback>
  <!-- Content: sequence-constructor -->
</xsl:fallback>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:for-each

Categorie: instruction

Model:

```
<xsl:for-each
  select = expression>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:for-each-group

Categorie: instruction

Model:

```
<xsl:for-each-group
  select = expression
  group-by? = expression
  group-adjacent? = expression
  group-starting-with? = pattern
  group-ending-with? = pattern
  collation? = { uri }>
  <!-- Content: (xsl:sort*, sequence-constructor) -->
</xsl:for-each-group>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:function

Categorie: declaration

Model:

```
<xsl:function
  name = qname
  as? = sequence-type
  override? = "yes" | "no">
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:function>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:if

Categorie: instruction

Model:

```
<xsl:if
  test = expression>
  <!-- Content: sequence-constructor -->
</xsl:if>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:import

Categorie: declaration

Model:

```
<xsl:import
  href = uri-reference />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:import-schema

Categorie: declaration

Model:

```
<xsl:import-schema
  namespace? = uri-reference
  schema-location? = uri-reference>
  <!-- Content: xs:schema? -->
</xsl:import-schema>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:include

Categorie: declaration

Model:

```
<xsl:include
  href = uri-reference />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:key

Categorie: declaration

Model:

```
<xsl:key
  name = qname
  match = pattern
  use? = expression
  collation? = uri>
  <!-- Content: sequence-constructor -->
</xsl:key>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:matching-substring

Model:

```
<xsl:matching-substring>
  <!-- Content: sequence-constructor -->
</xsl:matching-substring>
```

Toegestaan als child element van:

- `xsl:analyze-string`

xsl:message

Categorie: instruction

Model:

```
<xsl:message
  select? = expression
  terminate? = { "yes" | "no" }>
  <!-- Content: sequence-constructor -->
</xsl:message>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element
- `xsl:function`

xsl:namespace

Categorie: instruction

Model:

```
<xsl:namespace
  name = { ncname }
  select? = expression>
  <!-- Content: sequence-constructor -->
</xsl:namespace>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:namespace-alias

Categorie: declaration

Model:

```
<xsl:namespace-alias
  stylesheet-prefix = prefix | "#default"
  result-prefix = prefix | "#default" />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:next-match

Categorie: instruction

Model:

```
<xsl:next-match>
  <!-- Content: (xsl:with-param | xsl:fallback)* -->
</xsl:next-match>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:non-matching-substring

Model:

```
<xsl:non-matching-substring>
  <!-- Content: sequence-constructor -->
</xsl:non-matching-substring>
```

Toegestaan als child element van:

- `xsl:analyze-string`

xsl:number

Categorie: instruction

Model:

```
<xsl:number
  value? = expression
  select? = expression
  level? = "single" | "multiple" | "any"
  count? = pattern
  from? = pattern
  format? = { string }
  lang? = { nmtoken }
  letter-value? = { "alphabetic" | "traditional" }
  ordinal? = { string }
  grouping-separator? = { char }
  grouping-size? = { number } />
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:otherwise

Model:

```
<xsl:otherwise>
  <!-- Content: sequence-constructor -->
</xsl:otherwise>
```

Toegestaan als child element van:

- `xsl:choose`

xsl:output

Categorie: declaration

Model:

```
<xsl:output
  name? = qname
  method? = "xml" | "html" | "xhtml" | "text" | qname-but-not-ncname
  byte-order-mark? = "yes" | "no"
  cdata-section-elements? = qnames
  doctype-public? = string
  doctype-system? = string
  encoding? = string
  escape-uri-attributes? = "yes" | "no"
  include-content-type? = "yes" | "no"
  indent? = "yes" | "no"
  media-type? = string
  normalization-form? = "NFC" | "NFD" | "NFKC" | "NFKD" | "fully-normalized"
| "none" | nmtoken
  omit-xml-declaration? = "yes" | "no"
  standalone? = "yes" | "no" | "omit"
  undeclare-prefixes? = "yes" | "no"
  use-character-maps? = qnames
  version? = nmtoken />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:output-character

Model:

```
<xsl:output-character
  character = char
  string = string />
```

Toegestaan als child element van:

- `xsl:character-map`

xsl:param

Categorie: declaration

Model:

```
<xsl:param
  name = qname
  select? = expression
  as? = sequence-type
  required? = "yes" | "no"
  tunnel? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:param>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`
- `xsl:function`
- `xsl:template`

xsl:perform-sort

Categorie: instruction

Model:

```
<xsl:perform-sort
  select? = expression>
  <!-- Content: (xsl:sort+, sequence-constructor) -->
</xsl:perform-sort>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:preserve-space

Categorie: declaration

Model:

```
<xsl:preserve-space
  elements = tokens />
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:processing-instruction

Categorie: instruction

Model:

```
<xsl:processing-instruction
  name = { ncname }
  select? = expression>
  <!-- Content: sequence-constructor -->
</xsl:processing-instruction>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:result-document

Categorie: instruction

Model:

```
<xsl:result-document
  format? = { qname }
  href? = { uri-reference }
  validation? = "strict" | "lax" | "preserve" | "strip"
  type? = qname
  method? = { "xml" | "html" | "xhtml" | "text" | qname-but-not-ncname }
  byte-order-mark? = { "yes" | "no" }
  cdata-section-elements? = { qnames }
  doctype-public? = { string }
  doctype-system? = { string }
  encoding? = { string }
  escape-uri-attributes? = { "yes" | "no" }
  include-content-type? = { "yes" | "no" }
  indent? = { "yes" | "no" }
  media-type? = { string }
  normalization-form? = { "NFC" | "NFD" | "NFKC" | "NFKD" | "fully-
normalized" | "none" | nmtoken }
  omit-xml-declaration? = { "yes" | "no" }
  standalone? = { "yes" | "no" | "omit" }
  undeclare-prefixes? = { "yes" | "no" }
  use-character-maps? = qnames
  output-version? = { nmtoken }>
  <!-- Content: sequence-constructor -->
</xsl:result-document>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:sequence

Categorie: instruction

Model:

```
<xsl:sequence
  select = expression>
  <!-- Content: xsl:fallback* -->
</xsl:sequence>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:sort

Model:

```
<xsl:sort
  select? = expression
  lang? = { nmtoken }
  order? = { "ascending" | "descending" }
  collation? = { uri }
  stable? = { "yes" | "no" }
  case-order? = { "upper-first" | "lower-first" }
  data-type? = { "text" | "number" | qname-but-not-ncname }>
  <!-- Content: sequence-constructor -->
</xsl:sort>
```

Toegestaan als child element van:

- xsl:apply-templates
- xsl:for-each
- xsl:for-each-group
- xsl:perform-sort

xsl:strip-space

Categorie: declaration

Model:

```
<xsl:strip-space
  elements = tokens />
```

Toegestaan als child element van:

- xsl:stylesheet
- xsl:transform

xsl:stylesheet

Model:

```
<xsl:stylesheet
  id? = id
  extension-element-prefixes? = tokens
  exclude-result-prefixes? = tokens
  version = number
  xpath-default-namespace? = uri
  default-validation? = "preserve" | "strip"
  default-collation? = uri-list
  input-type-annotations? = "preserve" | "strip" | "unspecified">
  <!-- Content: (xsl:import*, other-declarations) -->
</xsl:stylesheet>
```

Alleen toegestaan als root element

xsl:template

Categorie: declaration

Model:

```
<xsl:template
  match? = pattern
  name? = qname
  priority? = number
  mode? = tokens
  as? = sequence-type>
  <!-- Content: (xsl:param*, sequence-constructor) -->
</xsl:template>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`

xsl:text

Categorie: instruction

Model:

```
<xsl:text
  [disable-output-escaping]? = "yes" | "no">
  <!-- Content: #PCDATA -->
</xsl:text>
```

Toegestaan als child element van:

- een XSLT element waarvan het content model een *sequence constructor* is
- elk literal result element

xsl:transform

Model:

```
<xsl:transform
  id? = id
  extension-element-prefixes? = tokens
  exclude-result-prefixes? = tokens
  version = number
  xpath-default-namespace? = uri
  default-validation? = "preserve" | "strip"
  default-collation? = uri-list
  input-type-annotations? = "preserve" | "strip" | "unspecified">
  <!-- Content: (xsl:import*, other-declarations) -->
</xsl:transform>
```

Alleen toegestaan als root element

xsl:value-of

Categorie: instruction

Model:

```
<xsl:value-of
  select? = expression
  separator? = { string }
  [disable-output-escaping]? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:value-of>
```

Toegestaan als child element van:

- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:variable

Categorie: declaration instruction

Model:

```
<xsl:variable
  name = qname
  select? = expression
  as? = sequence-type>
  <!-- Content: sequence-constructor -->
</xsl:variable>
```

Toegestaan als child element van:

- `xsl:stylesheet`
- `xsl:transform`
- `xsl:function`
- elk XSLT element waarvan het content model een sequence constructor is
- elk literal result element

xsl:when

Model:

```
<xsl:when
  test = expression>
  <!-- Content: sequence-constructor -->
</xsl:when>
```

Toegestaan als child element van:

- `xsl:choose`

xsl:with-param

Model:

```
<xsl:with-param
  name = qname
  select? = expression
  as? = sequence-type
  tunnel? = "yes" | "no">
  <!-- Content: sequence-constructor -->
</xsl:with-param>
```

Toegestaan als child element van:

- `xsl:apply-templates`
- `xsl:apply-imports`
- `xsl:call-template`
- `xsl:next-match`

Appendix C Opdrachten en uitwerkingen

Opdrachten bij hoofdstuk 2

1. Veel applicaties vereisen een vereenvoudigde XML-structuur, waarin alleen elementen zijn toegestaan, en geen attributen. Maak een stylesheet bestelling_elementen.xsl op basis van bestelling.xml. Alle attributen moeten als element worden aangemaakt.

   Tips:

   - Maak gebruik van de identity transform stylesheet.
   - Gebruik <xsl:element> om een element aan te maken, en de node-set functie name() om de naam ervan te genereren.

2. Als gevolg van voorgaande opdracht kunnen er mixed-content elementen ontstaan. Deze hebben zowel sub-elementen als text-nodes als inhoud, zoals in dit voorbeeld:

```
<prijs>
  <valuta>€</valuta>
  724.50
</prijs>
```

   Mixed content elementen zijn lastiger te verwerken dan elementen die alleen tekst of alleen andere elementen bevatten. Deze zijn daarom voor veel applicaties niet toegestaan.
   Zorg ervoor dat er geen mixed-content elementen meer gegenereerd worden, door in dit soort gevallen de tekst-waarde in een element <waarde> te zetten, dus als volgt:

```
<prijs>
   <valuta>€</valuta>
   <waarde>724.50</waarde>
</prijs>
```

   Tips:

   - Bedenk welke XPath expressie nodig is om de text-nodes te selecteren van elementen die ook één of meer attributen hebben.
   - Let op: ook white-space (spaties, tabs en enters) geldt als text-node en hoeft niet omgezet te worden. Met string-length() en normalize-space() kan worden gecontroleerd of de text-node ook andere tekst bevat.

3. Maak een stylesheet sorted_list.xsl op basis van cdlijst1.xml, waarmee een gesorteerde, door komma's gescheiden lijst van de cd-titels wordt gegenereerd. De transformatie dient de volgende uitvoer te geven:

```
Funhouse, Not Of This World, Plug It In, Pure Instinct, The Turn Of A friendly Card,
Wish You Where Here
```

Tips:

- gebruik een <xsl:for-each> loop
- maak gebruik van <xsl:choose> om het volgende te testen: als de positie niet gelijk is aan de laatste positie komt er geen komma achter de titel, anders wel
- met xsl:output kan het type uitvoer op text worden gezet

4. Maak een stylesheet cdcodes.xsl aan op basis van cdlijst1.xml. Elke cd en elke track krijgt een attribuut code. Cd's worden alfabetisch gecodeerd in hoofdletters, en tracks worden genummerd, met de cd-code als voorvoegsel (dus A.1, A.2, ... B.1, B.2 etc.). Dit betekent dat het sub-element nummer van het element track kan verdwijnen.

Tips:

- Maak gebruik van de identity transform stylesheet
- Maak een template aan die zowel voor cd- als voor track-elementen geldt
- Gebruik het XSLT element <xsl:number> om de code te genereren

Optioneel:

5. Maak een stylesheet functies_werknemers.xsl op basis van werknemers.xml. Laat per functie de namen zien. De uitvoer dient er (op de inspringing na) als volgt uit te zien:

```
<?xml version="1.0" encoding="utf-8"?>
<werknemers>
     <groep functie="KLERK">
          <naam>SMITS</naam>
          <naam>ADELAAR</naam>
          <naam>APPEL</naam>
          <naam>MANDERS</naam>
     </groep>
     <groep functie="VERKOPER">
          <naam>ALKEMA</naam>
          <naam>WALSTRA</naam>
          <naam>VERGEER</naam>
          <naam>DROST</naam>
     </groep>
     <groep functie="MANAGER">
          <naam>PIETERS</naam>
          <naam>KLAASEN</naam>
          <naam>HEUVEL</naam>
     </groep>
     <groep functie="ANALIST">
          <naam>SANDERS</naam>
          <naam>VERMEULEN</naam>
     </groep>
     <groep functie="DIRECTEUR">
          <naam>KRAAY</naam>
     </groep>
</werknemers>
```

Tips:

- bedenk eerst hoe alle unieke functies gevonden kunnen worden
- maak per functie een variabele aan met die functie, om de namen van werknemers met die functie te kunnen vinden
- maak gebruik van <xsl:attribute> bij het maken van elke groep

6. Gebruik nu in functies_werknemers.xsl geen variabele maar een key om de werknemers bij een functie te vinden.

7. Maak een stylesheet speelduur.xsl op basis van cdlijst1.xml. Na de transformatie moet de uitvoer er hetzelfde uitzien als cdlijst1.xml, alleen moet de speelduur van cd's en van tracks in seconden wordt weergegeven.

Opdrachten bij hoofdstuk 3

1. Maak een stylesheet werkn_per_functie2.xsl. Toon de namen van werknemers per functie. De uitvoer dient er (op de inspringing na) als volgt uit te zien:

```
<?xml version="1.0" encoding="utf-8"?>

<werknemers>
     <groep functie="KLERK">
          <naam>SMITS</naam>
          <naam>ADELAAR</naam>
          <naam>APPEL</naam>
          <naam>MANDERS</naam>
     </groep>
     <groep functie="VERKOPER">
          <naam>ALKEMA</naam>
          <naam>WALSTRA</naam>
          <naam>VERGEER</naam>
          <naam>DROST</naam>
     </groep>
     <groep functie="MANAGER">
          <naam>PIETERS</naam>
          <naam>KLAASEN</naam>
          <naam>HEUVEL</naam>
     </groep>
     <groep functie="ANALIST">
          <naam>SANDERS</naam>
          <naam>VERMEULEN</naam>
     </groep>
     <groep functie="DIRECTEUR">
          <naam>KRAAY</naam>
     </groep>
</werknemers>
```

   Maak hierbij gebruik van <xsl:for-each-group>.

2. Bewaar de stylesheet van de vorige opdracht als werkn_fun_docs.xsl. Pas deze zo aan dat elke groep als document wordt weggeschreven naar h:\functies. De bestanden krijgen als naam de naam van de functie in kleine letters, met als achtervoegsel .xml

3. Maak een HTML-tabel aan van de namen, functies en salarissen van werknemers.

   Zorg ervoor dat salarissen groter dan € 3000 vet gedrukt worden weergegeven. Het <TD> element krijgt dan als attribuut style="font-weight:bold" mee.
   Geef verder van de verkopers de hele rij een gele achtergrond. Het <TR> element krijgt dan het attribuut style="background-color:yellow".

   Tip:
   - Maak gebruik van <xsl:next-match> om deze uitzonderingen toe te voegen.
   - Let op: ook een default template kan als next-match herkend worden. Het kan dus nodig zijn om zo'n default template te overschrijven.

Optioneel:

4. Maak een functie mf:xor() – exclusive or – die van twee sequences alleen de nodes teruggeeft die in slechts één van beide sequences voorkomen. Dit kan op de volgende manier worden gegenereerd: neem alle elementen van beide sequences samen, en haal daar de intersectie tussen beide sequences vanaf. Test de functie door die werknemers te laten zien die als functie MANAGER hebben, *of* in kantoor 10 werken, dus zonder de manager HEUVEL, die ook in kantoor 10 werkt.

5. Maak een stylesheet verjaardagen.xsl aan. Toon daarin van de komende 10 jaar de dag in de week van uw verjaardag. Gebruik een min-teken als scheidingsteken tussen de verjaardagen.

   Tips:

   - Maak een variabele aan, met daarbinnen een for-each loop om voor elk jaar een nieuwe datum te genereren.

   - Zorg er in die variabele gelijk voor dat elke datum wordt weergegeven als dag van de week. Bedenk wat het type van de variabele moet zijn

   - Toon de variabele met xsl:value-of en gebruik daarbij een separator.