

1 XPath (XML Path Language)

Innehållsförteckning

1	XPath (XML Path Language)	1
1.1	Inledning	2
1.2	Steg i sökvägar	2
1.3	Nodtyper	3
1.4	Returtyper	3
1.5	Kontextnoden	4
1.6	XPath-uttryck för att hämta element och attribut	5
1.6.1	Wildcard	6
1.6.2	Attribut	7
1.7	Nodtester och filter	7
1.8	Funktioner i XPath	8
1.8.1	Funktioner som opererar på ett nodset	8
1.8.2	Strängfunktioner	10
1.8.3	Numeriska funktioner	12
1.8.4	Booleska funktioner	12

1.1 Inledning

I förra dokumentet började vi använda XSLT för att transformera ett XML-dokument till ett nytt dokument. En viktig del i samband med XSLT är standarden XML Path Language (XPath) som bland annat används för att navigera bland elementen i ett XML-dokument. Det finns även en rad funktioner i XPath som kan användas samt möjlighet att begränsa hur många av elementen en viss mall i XSLT gäller för. I denna lektion ska vi titta på de möjligheter XPath erbjuder. Redan i förra dokumentet tittade vi en del på XPath. Som du såg använda vi olika uttryck där sökvägar var en viktig komponent. Sökvägar kunde antingen vara absoluta eller relativa. Ett exempel på en absolut sökväg är: /personregister/person Här vill vi välja elementet person som är ett barnelement till rotelementet personregister.

I denna lektion kommer vi att använda oss av ett exempel med personer för att illustrera de olika uttrycken i XPath. Det XML-dokument som används är (personregister.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<personregister>
  <person personnr="1234123412">
    <namn>
      <förnamn>Karin</förnamn>
      <efternamn>Nordin</efternamn>
    </namn>
    <adress>
      <gata>Nordmansvägen 5</gata>
      <postnr>11111</postnr>
      <postort>Östersund</postort>
    </adress>
    <telefon typ="hem">11111111</telefon>
    <telefon typ="mobil">95959595</telefon>
    <civilstatus>Gift</civilstatus>
    <titel>Chefsredaktör</titel>
  </person>
</personregister>
```

I lektionen kommer vi bland annat att använda XPath-uttryck i olika mallar (XSLT-elementet xsl:template). Vi kommer inte att visa hela XSLT-dokumentet utan endast de olika mallarna. Givetvis måste ett komplett XSLT-dokument upprättas, så som vi såg exempel på i förra dokumentet, för att det ska fungera på ett korrekt sätt.

1.2 Steg i sökvägar

En sökväg i XPath består av ett eller flera steg (separerade med /) och resultatet av en sökväg är ett nodset. Sökvägen /personregister/person består av två steg. Ett steg består i sin tur av en axel, ett nodtest och eventuellt ett filter enligt:




Med en axel kan vi avgöra i vilken riktning vi söker i nodträdet. Med `child` söker vi efter allabarn till aktuell kontextnod. Nodtestet anger vad vi söker efter (här söker vi efter alla `personelement`, som är barn till aktuell kontextnod) och med ett filter kan vi ytterligare specificera vad vi söker efter (filtret ovan returnerar endast de `person`-element som har ett `civilstatuselement` vars värde är `Gift`). Axel och nodtestet separeras med dubbla kolon `::` och varje filter omsluts av hakparenteser `[]`.

Det finns totalt 13 olika axlar som kan användas för att specificera sökriktningen. Anges ingen axel används som standard `child`. I de exempel vi hittills tittat på har det med andra ord varit barnelement vi sökt efter. I denna lektion används uteslutande `child` som axel om inget annat anges. I kursboken ges exempel på fler axlar.

1.3 Nodtyper

XPath är ett språk för att navigera i ett hierarkiskt nodträd. I XPath betraktas ett XML-dokument som en samling av noder av olika typer. En nodtyp i ett XML-dokument kan vara:

<i>Nodtyp</i>	<i>Beskrivning</i>
Rotnod	Rotelementet i dokumentet. 
Elementnod	Ett vanligt element.
Attributnod	Ett attribut som finns i ett element.
Textnod	Själva textinnehållet i ett element.
Processinstruktionsnod	Processinstruktioner i dokumentet (inte XML-deklarationen).
Kommentarsnod	Kommentarer i XML-dokumentet.
Namnrymdsnod	Den namnrymd som används i dokumentet.

1.4 Returtyper

När vi använder ett XPath-uttryck kommer det att returneras någon form av data som sen används i bland annat XSLT. Vi såg i förra dokumentet exempel på att vi kan hämta ut textinnehållet i elementet `förnamn` och skriva detta till resultatdokumentet:

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

Här returneras all text innanför elementet `förnamn`. Det vill säga `Karin`. Det finns totalt fyra olika typer av data som ett XPath-uttryck kan returnera. Dessa datatyper är:

- `string` – ett textvärde, som `Karin`, `Nordmansvägen 5` eller `11111`. Som du ser kan det vara ett tecken (bokstäver och siffror) eller en längre text där mellanrum kan vara inkluderat.

- `number` – ett talvärde som `5` eller `-2300`. Vi kan därmed använda data som finns i XMLdokument

för att till exempel utföra matematiska beräkningar. Resultatet kommer att vara ett decimaltal (för er som har programmerat en del är det datatypen `float` som används).

- `boolean` – detta är ett booleskt värde som antingen kan vara sant eller falskt (`true` eller `false`). Ett booleskt värde kan returneras om vi till exempel använder olika funktioner i XPath (kommer exempel senare).
- `node-set` – Ett nodset (en samling av noder). Detta kan vara flera element eller attribut. I exemplet ovan hade vi kunnat ha flera `person`-element innanför elementet `personregister`. Vi skulle då haft ett nodset bestående av `person`-element.

1.5 Kontextnoden

Hur ett XPath-uttryck skrivs beror till stor del var uttrycket placeras i ett XSLT-dokument. Av den anledningen används begreppet kontextnod. Kontextnoden är den nod i nodträdet vi just nu befinner oss i. Det är från denna nod vi utgår ifrån inuti till exempel en mall. Tänk dig följande mall för elementet `namn`:

```
<xsl:template match="namn">
  <p>Förnamnet är <xsl:value-of select="förnamn" /></p>
</xsl:template>
```

Som du ser behöver vi här inte ange hela sökvägen (en absolut sökväg) till elementet `förnamn` i mallen, något som även skulle kunna bli fel (återkommer till detta längre fram).

Anledningen till att vi endast behöver skriva `förnamn` är för att i mallen är `namn` satt som kontextnod, vilket framgår av attributet `match` i mallen. I en mall är det kontextnoden nya XPath-uttryck utgår från.

Är elementet `person` satt som kontextnod måste vi använda sökvägen `namn/förnamn` för att få tag i `förnamn`, medan vi endast behöver skriva `förnamn` om `namn` är kontextnod.

Nedan listas den information vi kan få utifrån den kontext vi befinner oss i:

- Kontextnod – den nod som analyseras för ögonblicket.
- Kontextstorlek – antal noder i det nodset som analyseras.
- Kontextposition – kontextnodens position i det nodset som analyseras.
- Namnrymd – de namnrymder som existerar inom aktuellt definitionsområde (eng. scope).

Vi kommer enbart att använda namnrymden för XSLT, men givetvis är det möjligt att använda flera namnrymder.

- Funktioner – funktioner som finns inom aktuellt definitionsområde.

Vi ska nu se på ett exempel med några av begreppen i listan ovan. I XML-dokumentet över personer ser vi att det för personen `Karin` är registrerat två telefonnummer. För att hämta ut alla `telefonnr` kan vi använda följande två mallar i ett XSLT-dokument:

```
<xsl:template match="/">
  <xsl:apply-templates select="personregister/person/telefon" />
</xsl:template>
<xsl:template match="telefon">
  <p>Telefonnumret är <xsl:value-of select="." /></p>
</xsl:template>
```

Den första mallen matchar roten (match="/"). Genom att här använda elementet `xsl:apply-templates` anropas de mallar som matchar de noder i nodsetet som XPathuttrycket

i attributet `select` returnerar. I just detta fall returnerar XPath-uttrycket ett nodset med de två `telefon`-elementen som fanns i XML-dokumentet:

```
<telefon typ="hem">11111111</telefon>
<telefon typ="mobil">95959595</telefon>
```

Detta innebär att kontextstorleken här kommer att vara 2. Den andra mallen kommer att utföras på vart och ett av de två `telefon`-elementen. Först utförs den på hemnumret (vilket är det första `telefon`-elementet) varpå följande text skrivs ut:

Telefonnumret är 11111111

Nu kommer detta `telefon`-element att vara kontextnod. Vi ser att i mallen används elementet `xsl:value-of` där värdet på attributet `select` är en punkt. Punkten representerar aktuell kontextnod vilket innebär att det värde som hämtas är 11111111.

1.6 XPath-uttryck för att hämta element och attribut

Vi kommer nu att titta på några vanliga XPath-uttryck. Några av dess kommer att vara repetition från förra dokumentet, men vi tar upp det här ändå så vi får allt samlat.

Som du har sett använder vi ofta tecknet `/` i den första mallen i XSLT-dokumentet. Denna mall kommer då att representera hela dokumentet, prologen inkluderad. Den tar med andra ord inte utgångspunkt från rotelementet.

Vidare är det möjligt att navigera längre ner i XML-dokumentet. Vi kan till exempel utgå från kontextnoden, som vi kan anta är `personregister`, och skriva:

`person/namn`

Här använder vi en relativ sökväg till elementet `namn`. Det är även möjligt att ange en absolut sökväg till `namn` genom att börja sökvägen med `/`

`/personregister/person/namn`

Här ser du att vi börjar med `/` som representerar hela dokumentet och anger sen sökvägen till elementet `namn` utifrån det. Det är viktigt att tänka på hur en absolut sökväg fungerar, något som exemplet nedan visar. Tänk dig att vi har följande två mallar:

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

```
  <html>
```

```
    <body>
```

```
      <h1>Lista över personer</h1>
```

```
      <xsl:apply-templates select="personregister/person/telefon" />
```

```
    </body>
```

```
  </html>
```

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

```
<xsl:template match="telefon">
```

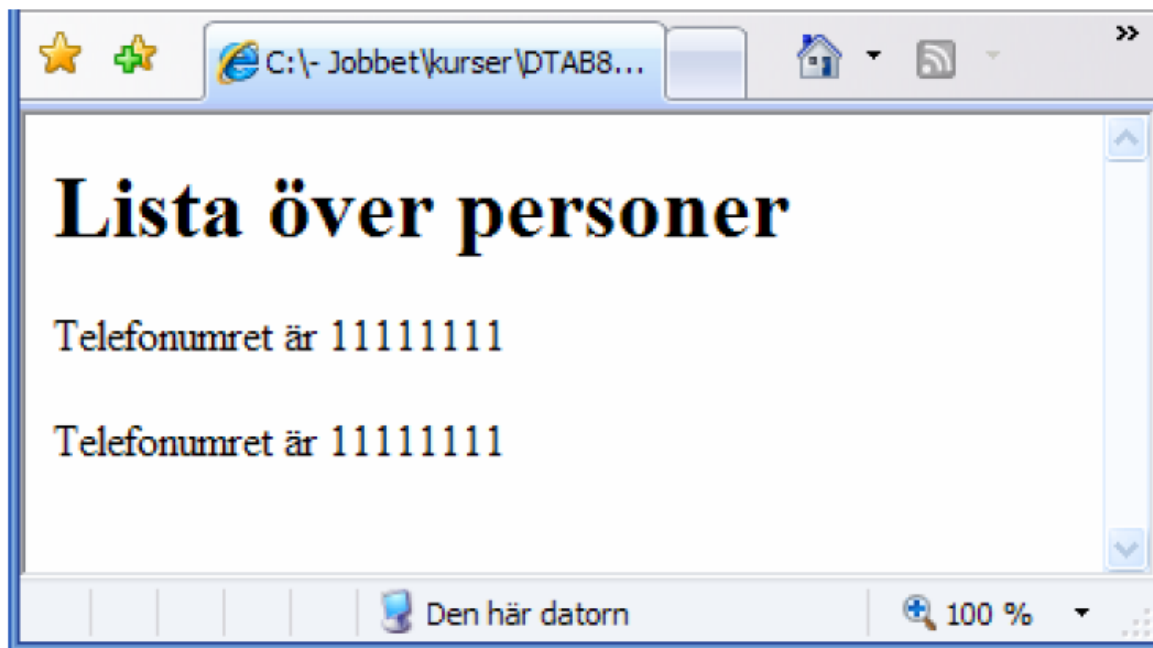
```
  <p>Telefonumret är <xsl:value-of select="/personregister/person/telefon"
```

```
  />
```

```
</p>
```

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

Markerat med fet stil ser du att vi, i andra mallen, använder en absolut sökväg till elementet `telefon` i stället för att utgå från kontextnoden som är just `telefon`. Om vi öppnar XML-dokumentet, som använder mallarna ovan i ett XSLT-dokument, i en webbläsare kommer vi att få följande utskrifter: (se nästa sida)



Som du ser visas rätt antal telefonnummer, med det är samma telefonnummer som visas på båda raderna. Anledningen är den absoluta sökvägen som användes i andra mallen. Denna väljer alltid ut första `telefon`-elementet i XML-dokumentet, som den absoluta sökvägen faktiskt pekar på. Detta visar att en absolut sökväg inte utgår från kontextnoden. För att utgå från kontextnoden måste som sagt en relativ sökväg användas.

Vi kan även använda ett XPath-uttryck som väljer ut alla `telefon`-element i dokumentet:
`//telefon`

Vi använder här `//` för att ange att alla noder av angiven typ (i detta fall `telefon`) ska väljas ut. Detta kan i många situationer vara bra att använda i den första mallen (där attributet `match` har värdet `/`) för att välja alla element av en viss typ i XML-dokumentet.

1.6.1 Wildcard

Det är möjligt att använda wildcards (`*`) för att hämta okända element i ett XML-dokumentet. Vi kan till exempel hämta alla barnelement till elementet `person` genom följande uttryck:

```
/personregister/person/*
```

Det går bra att även använda wildcard i ett steg mitt inne i ett XPath-uttryck och inte enbart i sista steget. För att ge exempel på ett användningsområde utgår vi från ett annat exempel än personregistret. Tänk dig att vi har ett XML-dokument över olika produkter. På flera ställen i dokumentet används elementet `pris` för att ange priset på dessa produkter. De olika `pris`-elementen är barnelement för andra element. Vi har ett element med namnet `köttprodukt` med ett barnelement `pris`, och vi har även ett element med namnet `mjölprodukt` som även det har ett barnelement som heter `pris`. Om vi nu önskar att beräkna det totala priset kan vi använda wildcards för att hämta alla `pris`-element för alla produkter oavsett om det är en

```
köttprodukt eller mjölprodukt:  
/produkter/*/pris
```

Här kan `*` representera både element av typen `köttprodukt` eller `mjölprodukt`. Uttrycket ovan innebär att vi väljer alla `pris`-element som är barnbarn till elementet `produkter`.

1.6.2 Attribut

XPath-uttryck kan också användas till att välja ut attribut i ett XML-dokument. Vi måste då i ett steg i sökvägen använda axeln `attribute`. I vårt exempel med personregistret finns två typer av attribut; `personnr` i elementet `person` samt `typ` i elementet `telefon`. Om kontextnoden är satt till `person` kan vi välja attributet `personnr` genom att skriva:

```
attribute::personnr
```

Om kontextnoden fortfarande är `person` kan uttrycket ovan användas i samband med elementet `xsl:value-of` enligt nedan för att hämta personens personnummer:

```
<xsl:template match="person">
  <p>Personnummer: <xsl:value-of select="attribute::personnr" />
  ...
</p>
</xsl:template>
```

I stället för att skriva `attribute::` kan förkortningen `@` användas. Om vi till exempel vill välja alla `personnr`-attribut i ett XML-dokument kan vi skriva:

```
//@personnr
```

1.7 Nodtester och filter



Nodtestet i ett steg i en sökväg har vi sett avgör vilken eller vilka noder som väljs ut. Detta tillsammans med en axel räcker väldigt långt. Om vi däremot vill välja ut element som har ett bestämt attribut eller ett bestämt barnelement, eller om vi vill välja ut element/attribut som har ett bestämt värde räcker inte enbart axlar och nodtest. Vi kan då använda oss av ett eller flera filter. Filtret anges innanför `[]` efter det nodtest vi vill att filtret ska gälla för:

```
namn[förnamn]
```

Detta uttryck väljer alla `namn`-element som har ett `förnamn`-element som barnelement. Om vi vill välja alla personer som har ett `personnr`-attribut skriver vi:

```
//person[@personnr]
```

Vi kan även välja alla `person`-element som har ett `civilstatus`-element med värdet `Gift`:

```
//person[civilstatus = 'Gift']
```

På samma sätt kan vi välja ut element vars attribut har ett bestämt värde:

```
telefon[@typ = 'hem']
```

Det är inte enbart `=` som kan användas i ett filter. Vi kan även använda `<` (mindre än), `>` (större än), `<=` (mindre än eller lika med), `>=` (större än eller lika med) och `!=` (skiljt från).

Var uppmärksam på att vi för `<` måste använda entiteten `<`. Gör vi inte det kommer `<` att uppfattas som början på ett nytt element och vi får ett felmeddelande. Vi kan nu med hjälp av elementet `xsl:apply-templates` välja ut alla personer som har ett `postnr` större än 80000:

```
<xsl:apply-templates select="//person[adress/postnr > 80000]"/>
```

Uttrycket ovan kommer inte att resultera i något eftersom den enda personen som är inlagd har ett postnummer mindre än 80000. Notera att om vi vill att testet ska göras på ett element

som ligger djupare ner i hierarkin än det element vi väljer ut (ovan väljer vi `person`, men testar på `postnr`), måste hele sökvägen till elementet som testet gäller för anges innanför `[]`. Denna sökväg är relativ till nodtestet (det som står till vänster om `[]`). Det är möjligt att använda flera tester i ett filter genom att använda operatorerna `or` (eller) och `and` (och). Vi kan till exempel ange ett filter som testar om en person har både elementen `förnamn` och `efternamn` registrerat:

```
//person[namn/förnamn and namn/efternamn]
```

Filtret behöver inte enbart finnas i sista steget i en sökväg, utan kan användas i vilket steg som helst. Vi kan välja ut alla efternamn för de personer som har ett personnummer:

```
//person[@personnr]/namn/efternamn
```

1.8 Funktioner i XPath

XPath innehåller en hel del funktioner som kan användas i framför allt filter, men även fristående. Funktionerna kan bland annat användas i samband med elementet `xsl:value-of` för att välja ut värden till en nod i XML-dokumentet. Det finns flera typer av funktioner. Några opererar på ett nodset, medan andra opererar på text och tal. Vi kommer inte att nämna alla funktioner här, utan tar några exempel så finns det mer i kursboken.

1.8.1 Funktioner som opererar på ett nodset

Tidigare beskrev vi ett nodset som en samling av flera noder. När vi använder koden:

```
<xsl:apply-templates select="/personer/person/telefon"/>
```

väljs båda `telefon`-elementen i XML-dokumentet. Med andra ord kommer XPath-uttrycket ovan att returnera ett nodset innehållandes två noder och `xsl:apply-templates` går igenom detta nodset och skickar varje nod till en matchande mall. Det finns flera funktioner som kan användas med ett nodset, varav fyra av de vanligaste listas nedan:

<i>Funktion</i>	<i>Beskrivning</i>
<code>name()</code> <code>name(argument)</code>	Returnerar namnet på aktuell kontextnod eller på nodsetet som anges som argument.
<code>position()</code>	Returnerar kontextnodens position i nodsetet.
<code>last()</code>	Returnerar kontextpositionen för sista noden i nodsetet.
<code>count(argument)</code>	Returnerar antal noder i nodsetet som anges som argument.

Tidigare beskrev vi ett nodset som en samling av flera noder. När vi använder koden:

```
<xsl:apply-templates select="/personer/person/telefon"/>
```

väljs båda `telefon`-elementen i XML-dokumentet. Med andra ord kommer XPath-uttrycket ovan att returnera ett nodset innehållandes två noder och `xsl:apply-templates` går igenom detta nodset och skickar varje nod till en matchande mall. Det finns flera funktioner som kan användas med ett nodset, varav fyra vanliga lista enligt:

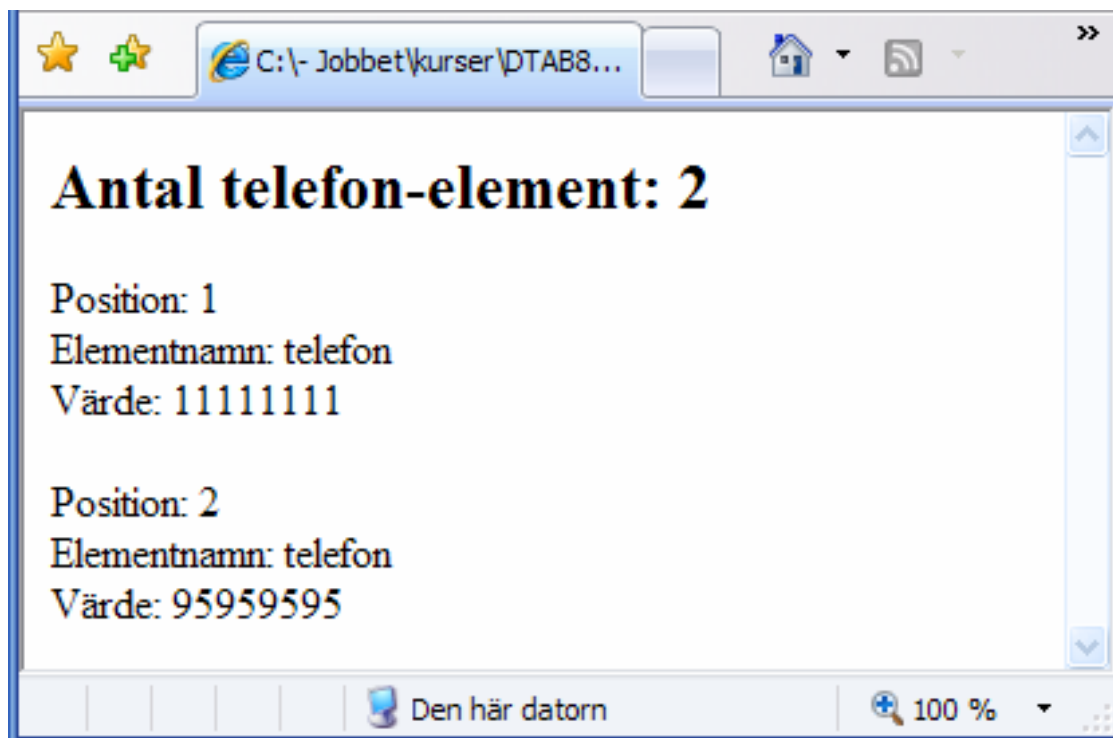

```

<xsl:template match="/">
  <html>
    <body>
      <h2>Antal telefon-element:
        <xsl:value-of select="count(//telefon)" />
      </h2>
      <xsl:apply-templates select="personregister/person/telefon"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="telefon">
  <p>Position: <xsl:value-of select="position()" />
    <br/>
    Elementnamn: <xsl:value-of select="name()" />
    <br/>
    Värde: <xsl:value-of select="." />
  </p>
</xsl:template>

```

Vi börjar med att skriva ut totala antalet `telefon`-element i XML-dokumentet genom att använda funktionen `count()`. Därefter går vi igenom alla `telefon`-element. I mallen som matchar `telefon` börjar vi med att skriva ut den kontextposition aktuell `telefon`-nod har i nodsetet genom att använda funktionen `position()`. Genom att använda funktionen `name()` skriver vi därefter ut elementnamnet på aktuell nod. Till sist skriver vi ut nodens värde genom att använda XPath-uttrycket `.` (punkt som representerar kontextnoden). I en webbläsare får vi nu följande resultat:



Vill vi välja ut det sista telefonnumret för varje person kan vi i första mallen skriva:

```
<xsl:apply-templates select="/personregister/person/telefon[position() = last()]" />
```

Är vi i stället intresserad av enbart det sista telefonnumret totalt sett skriver vi:

```
<xsl:apply-templates select="/descendant::telefon[position() = last()]" />
```

1.8.2 Strängfunktioner

Det finns ett flertal funktioner som på något sätt kan användas på eller har att göra med textsträngar. Nedan listas några vanliga strängfunktioner:

<i>Funktion</i>	<i>Beskrivning</i>	<i>Exempel</i>
<code>concat(arg, arg, arg*)</code>	Funktionen sätter samman flera textsträngar till en. Funktionen kan ta ett obegränsat antal argument och dessa sätts samman till en enda lång sträng.	<code>concat('H', 'jälp')</code> → 'Hjälp' <code>concat('X', 'M', 'L')</code> → 'XML'
<code>contains(string, substr)</code>	Funktionen kontrollerar om <code>string</code> innehåller <code>substr</code> . Om så är fallet returneras <code>true</code> , annars returneras <code>false</code> .	<code>contains('XML', 'X')</code> → <code>true</code>
<code>substring(arg, num1, num2)</code>	Funktionen returnerar en delsträng av <code>arg</code> som börjar på position <code>num1</code> och är <code>num2</code> antal tecken lång. Första tecknet i <code>arg</code> har position 1. Om <code>num2</code> utesluts fortsätter delsträngen till slutet av <code>arg</code> .	<code>substring('XPath', 2)</code> → <code>Path</code> <code>substring('XPath', 2, 3)</code> → <code>Pat</code>
<code>starts-with(arg1, arg2)</code>	Funktionen kontrollerar om strängen i <code>arg1</code> börjar med <code>arg2</code> och returnerar <code>true</code> om så är fallet.	<code>starts-with('XML', 'X')</code> → <code>true</code>
<code>string-length(arg)</code>	Funktionen returnerar antalet tecken som strängen <code>arg</code> består av. Utesluts argumentet returneras antal tecken innehållet i kontextnoden har.	<code>string-length('XPath')</code> → 5

Nedan följer ett exempel på en mall i XSLT som använder några funktionerna:

```
<xsl:template match="person">
  <p>Personens initialer:
    <xsl:value-of select="concat(substring(namn/förnamn, 1, 1),
                                substring(namn/efternamn, 1, 1))" />

    <br />
    Telefon (<xsl:value-of select="concat(telefon/@typ, '): ', telefon)" />
  </p>
</xsl:template>
```

Här har vi en mall som börjar med att skriva ut en persons initialer. Vi använder då först funktionen `concat()` för att sätta samman första bokstaven i för- och efternamnet. Som argument till denna funktion använder vi funktionen `substring()`. Den första `substring`-funktionen returnerar första bokstaven i förnamnet, medan den andra funktionen returnerar första bokstaven i efternamnet. Till sist sätter vi samman en text av `telefonnr`. Resultatet av detta kommer att bli:



Notera att funktionen `concat()` inte är nödvändig att använda då vi i stället kan sätta samman text genom att använda flera `xsl:value-of`-element.

1.8.3 Numeriska funktioner

<i>Funktion</i>	<i>Beskrivning</i>
<code>number()</code> <code>number(arg)</code>	Funktionen konverterar aktuell kontextnod eller argumentet <code>arg</code> till ett tal och returnerar det. <code>NaN</code> (not a number) returneras om innehållet inte kan konverteras till ett tal.
<code>sum(arg)</code>	Beräknar summan av alla noder i nodsetet <code>arg</code> . Summan beräknas genom att varje nod i <code>arg</code> först konverteras till en sträng (funktionen <code>string()</code>) som sedan konverteras till ett tal (<code>number()</code>) för att till sist adderas.
<code>round(arg)</code>	Avrundar talet i argumentet <code>arg</code> till närmaste heltal. <code>round(3.5) → 4, round(-3.5) → -3, round(9.2) → 9</code>

Vi kan dessutom använda `+`, `-`, `*`, `div` och `mod` i funktionerna ovan samt i beräkningar. `div` används i stället för `/` (division), medan `mod` ger resten i en division (till exempel är `7 mod 3` lika med `1`, eftersom detta är resten efter en division; `7 = 3*2+1`). För att till exempel avrunda en summa (av ett nodset med pris-element) till 2 decimaler kan vi:

```
round(sum(pris) * 100) div 100
```

Här multiplicerar vi först summan med 100 och avrundar det till närmaste heltal, för att till slut dividera med 100 igen.

1.8.4 Booleska funktioner

<i>Funktion</i>	<i>Beskrivning</i>
<code>not(arg)</code>	Denna funktion returnerar det motsatta booleska värdet av <code>arg</code> . Om <code>arg</code> är <code>true</code> kommer funktionen att returnera <code>false</code> och tvärtom. För att till exempel välja ut alla personer som inte är gifta kan vi skriva: <code>personregister/person[not(civilstatus = 'Gift')]</code>
<code>boolean(arg)</code>	Funktionen konverterar argumentet <code>arg</code> till ett booleskt värde. Om <code>arg</code> är ett notset returneras <code>false</code> om nodsetet är tomt, annars returneras <code>true</code> . Om <code>arg</code> är ett numeriskt värde returneras <code>true</code> för alla tal som inte är <code>0</code> eller <code>NaN</code> . Om <code>arg</code> är en sträng returneras <code>false</code> om antalet tecken i strängen är <code>0</code> , annars returneras <code>true</code> .

Utöver dessa två booleska funktioner finns ytterligare ett par som du kan läsa mer om i kursboken.