## Chapter 9: XSLT Extensible Stylesheet Language/Transformations

#### **References:**

- James Clark (Editor): XSL Transformations (XSLT), Version 1.0 W3C Recommendation, 16 November 1999 [https://www.w3.org/TR/xslt]
- Michael Kay (Editor): XSL Transformations (XSLT), Version 2.0 W3C Recommendation, 23 January 2007 [http://www.w3.org/TR/xslt20/]
- Michael Kay (Editor): XSL Transformations (XSLT), Version 3.0 W3C Candidate Recommendation, 19 November 2015 [http://www.w3.org/TR/xslt-30/]
- Michael Kay: XSLT 2.0 and XPath 2.0 Programmer's Reference (Programmer to Programmer) Wiley, 4th Ed. (June 3, 2008), ISBN-10: 0470192747, 1376 pages.
- Wikipedia (English): XSLT [https://en.wikipedia.org/wiki/XSLT]
- Robert Tolksdorf: Vorlesung XML-Technologien (Web Data and Interoperability), Kapitel 6: XSLT: Transformation von XML-Dokumenten. Freie Universität Berlin, AG Netzbasierte Informationssysteme, 2015. [http://blog.ag-nbi.de/wp-content/uploads/2015/05/06\_XSLT.pdf]
- w3schools: XSLT ELement Reference. [http://www.w3schools.com/xml/xsl\_elementref.asp]

# Objectives

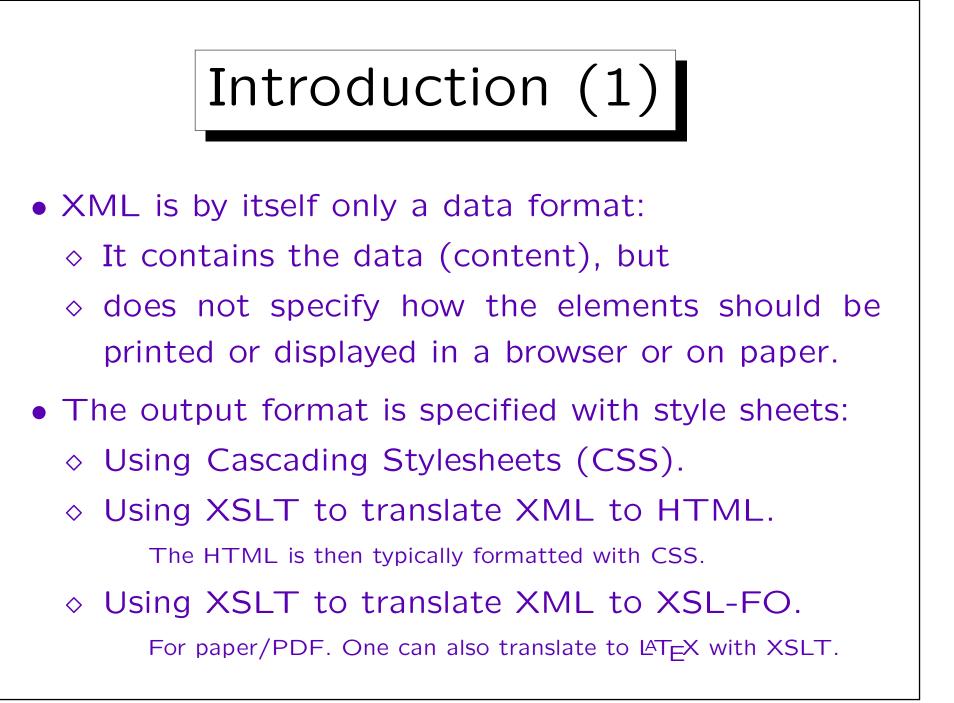
After completing this chapter, you should be able to:

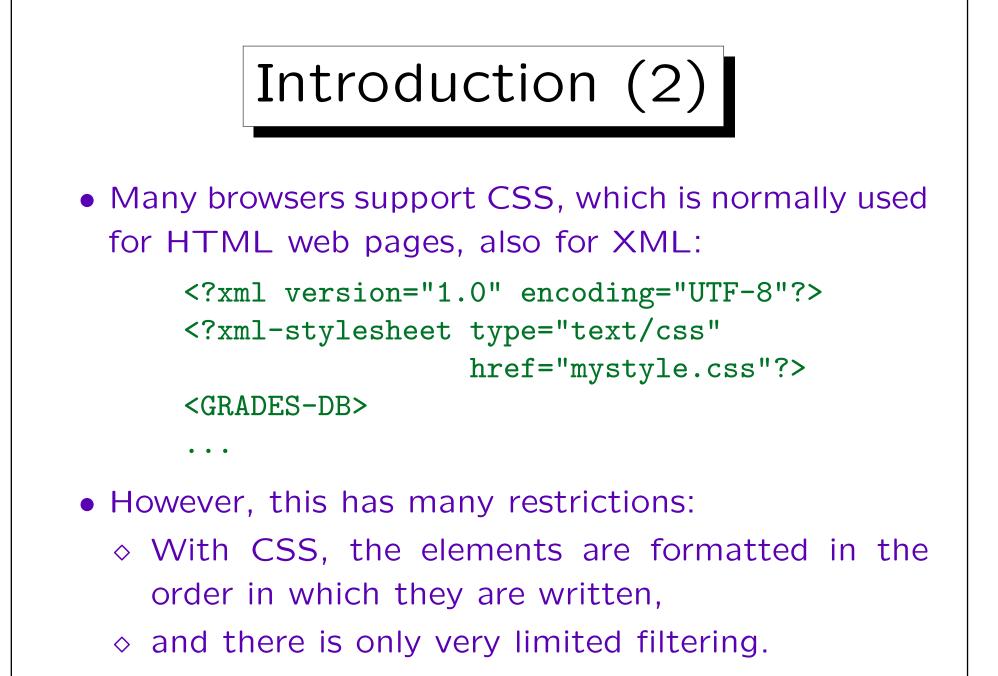
• write transformations from XML to XML, or from XML to HTML as an XSLT stylesheet.

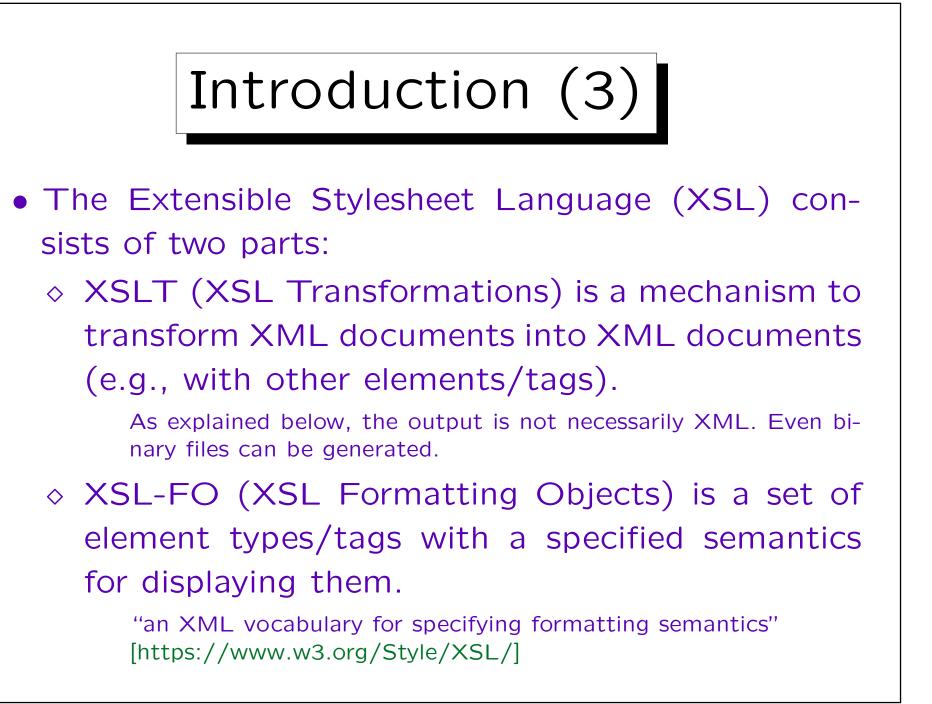
This chapter also explains how a transformation from XML to  $\mbox{LT}_{E}\!X$  is done with XSLT.

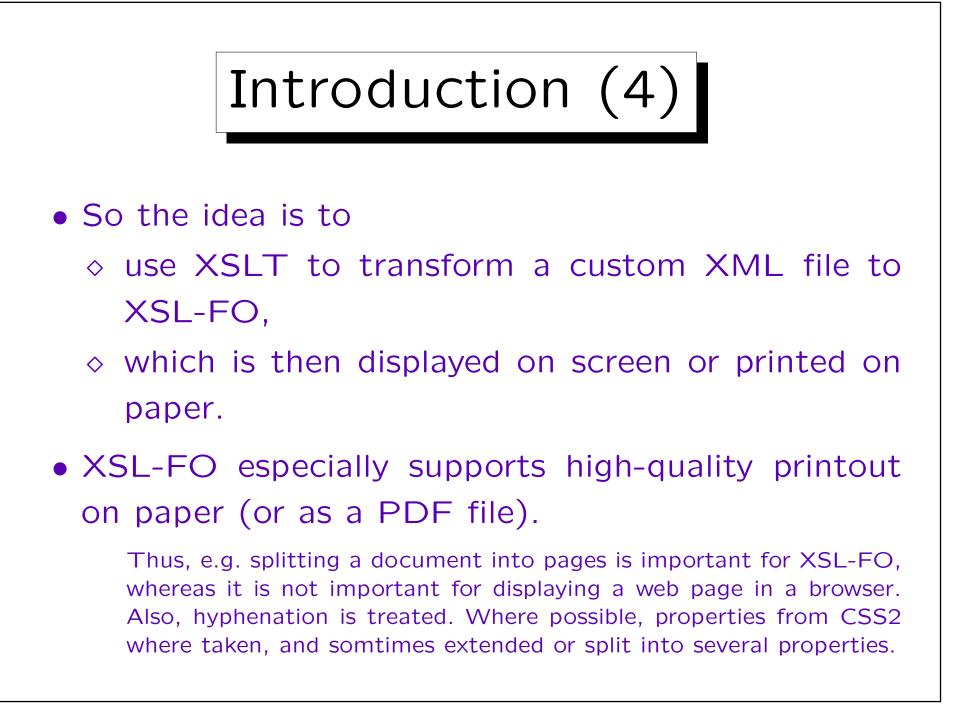
• read and understand given XSLT stylesheets.













- XSL has its roots in DSSSL, the Document Style Semantics and Specification Language (for SGML).
- XSLT 1.0 became a W3C recommendation (official standard) on November 16, 1999.

See [https://www.w3.org/TR/xslt]. The current version is XSLT 2.0 from Januar 23, 2007. [https://www.w3.org/TR/xslt20/].

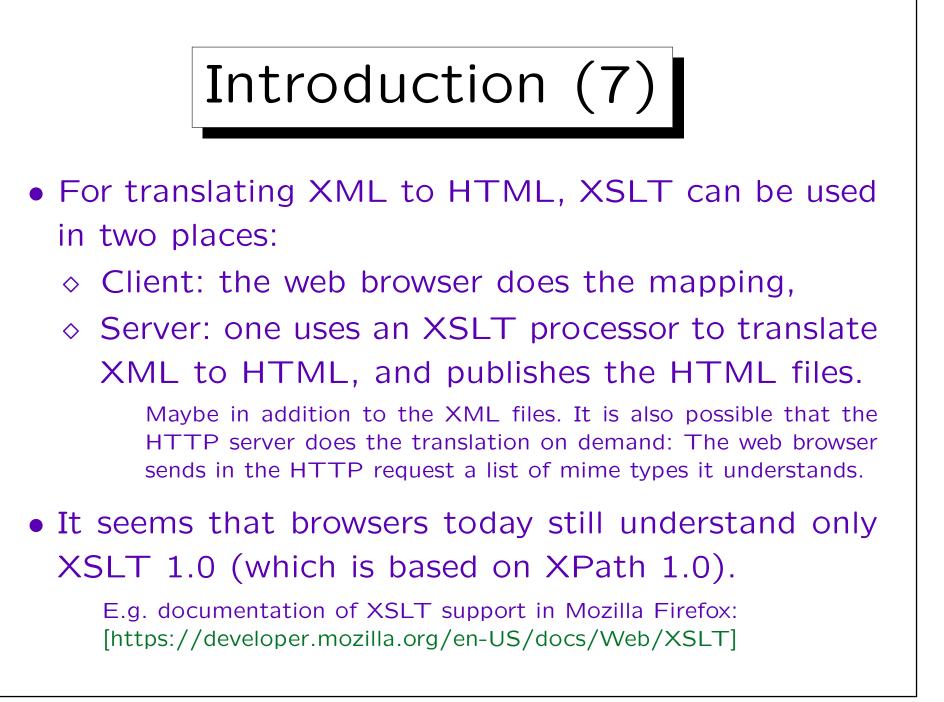
• XSL 1.0 (which specifies XSL-FO) became a W3C recommendation on October 15, 2001.

See [https://www.w3.org/TR/2001/REC-xsl-20011015/] Current ver.: XSL 1.1 (Dec. 5, 2006) [https://www.w3.org/TR/xsl/] Draft: XSL 2.0 (Jan. 17, 2012) [https://www.w3.org/TR/xslfo20/]



- Quite often, XSLT is used without XSL-FO:
  - ♦ For instance, XML is transformed to HTML to be displayed in a browser.
  - ◊ Or XSLT is used to transform a given XML document into a differently structured XML document (with different element types/tags).

In this way, one can adapt an XML file from a business partner to one's own XML structure. Or one can integrate XML files from different sources to a common XML vocabulary.





- Doing the XML to HTML mapping on Client or Server, continued:
  - ◇ If one does the translation in an intranet only for the employees of the company, one can at least rely on the knowledge which browser is used.
  - On the global internet, it might be that potential customers use old browsers which do not support XSLT or support it in incompatible ways.

One can still put the XML file on the server in addition to the HTML file, in order to support semantic web applications (like price comparision services).





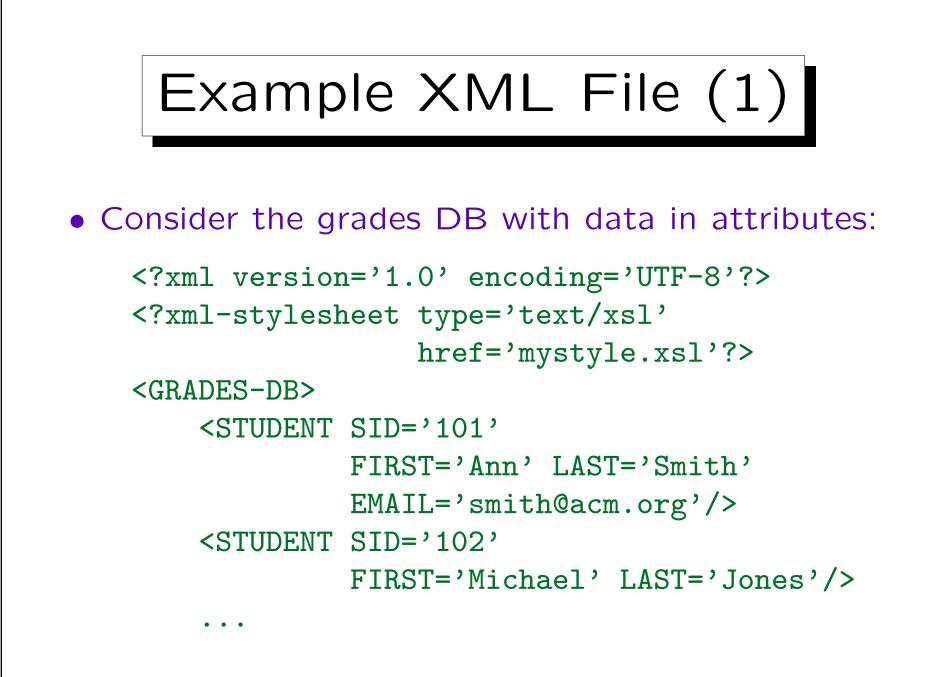
1. Introduction

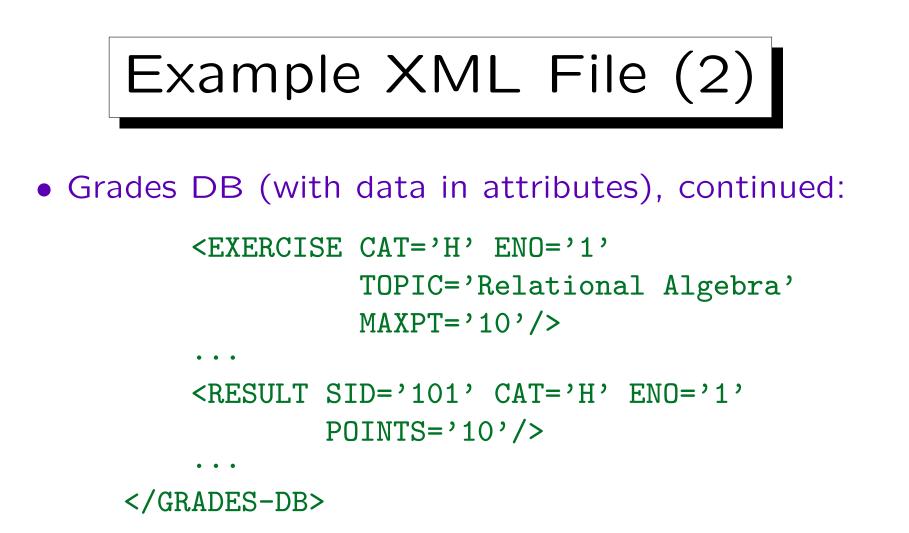
2. Example XSLT Stylesheet

3. Template Rules: Details

4. Restrictions in XPath 1.0

5. More XSLT Constructs





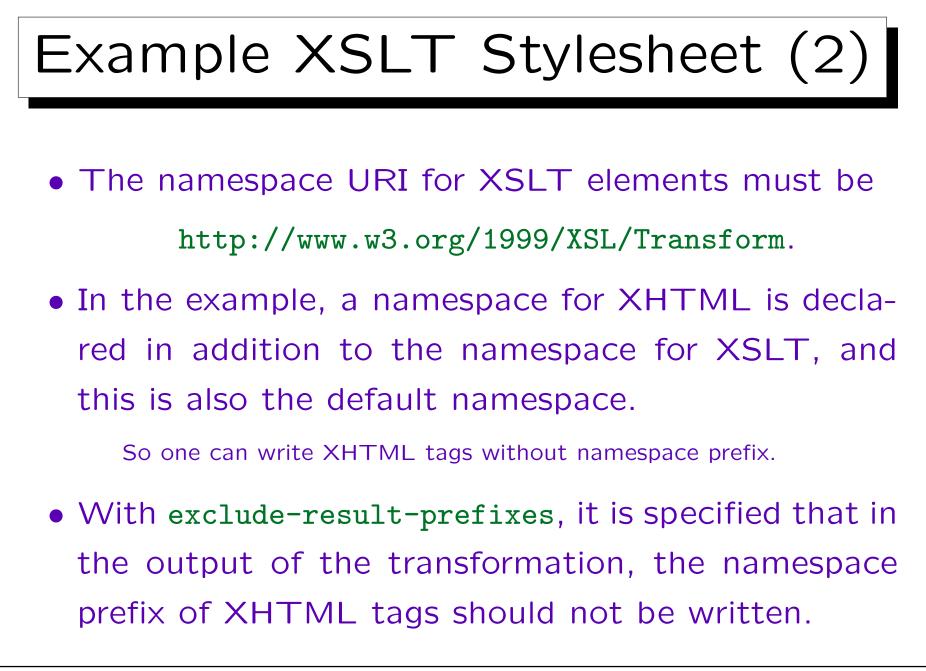
Note: If there is a typing error in the name of the stylesheet, many browsers (e.g., Firefox 43) silently apply the built-in templates (see Slide 9-56), which means that the output will be empty if the data is stored in attributes. Textual element content is shown.

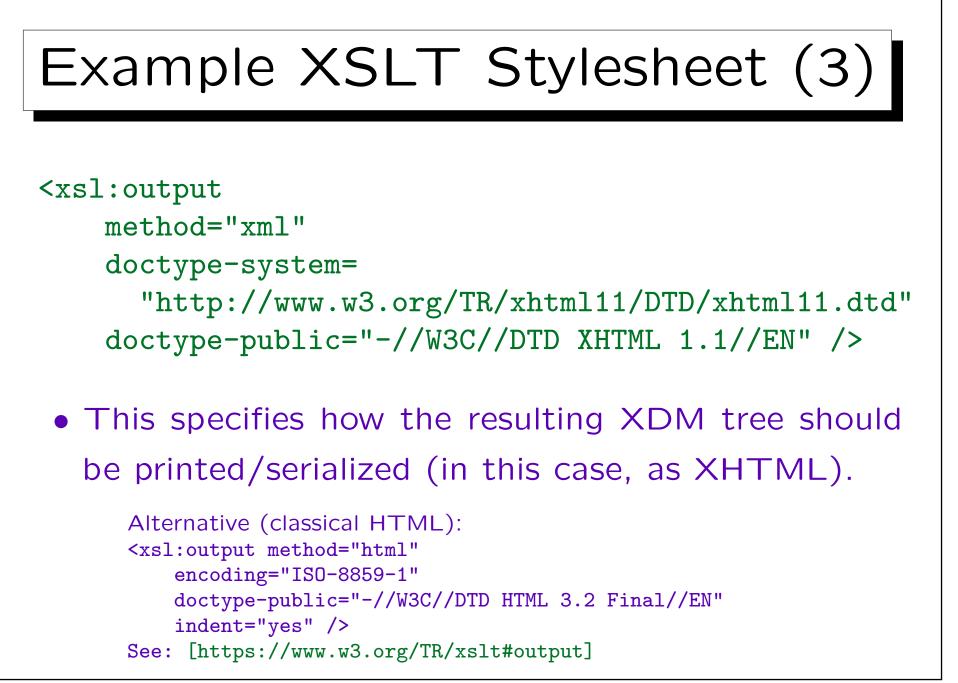


```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
   version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:html="http://www.w3.org/1999/xhtml"
   xmlns="http://www.w3.org/1999/xhtml"
   exclude-result-prefixes="html">
```

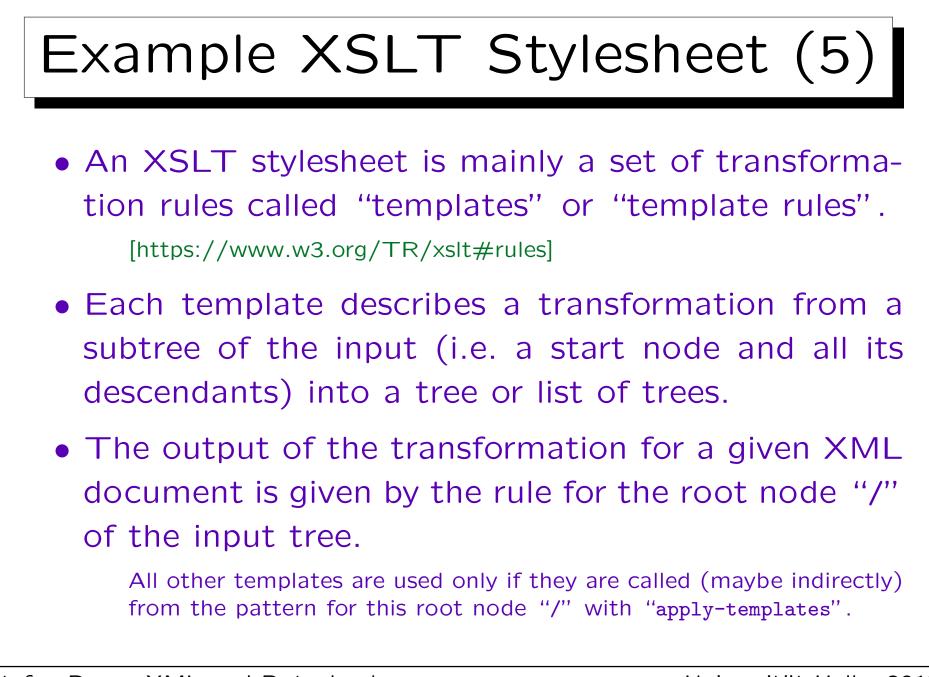
• XSLT stylesheets are written in XML syntax, using the outermost element stylesheet.

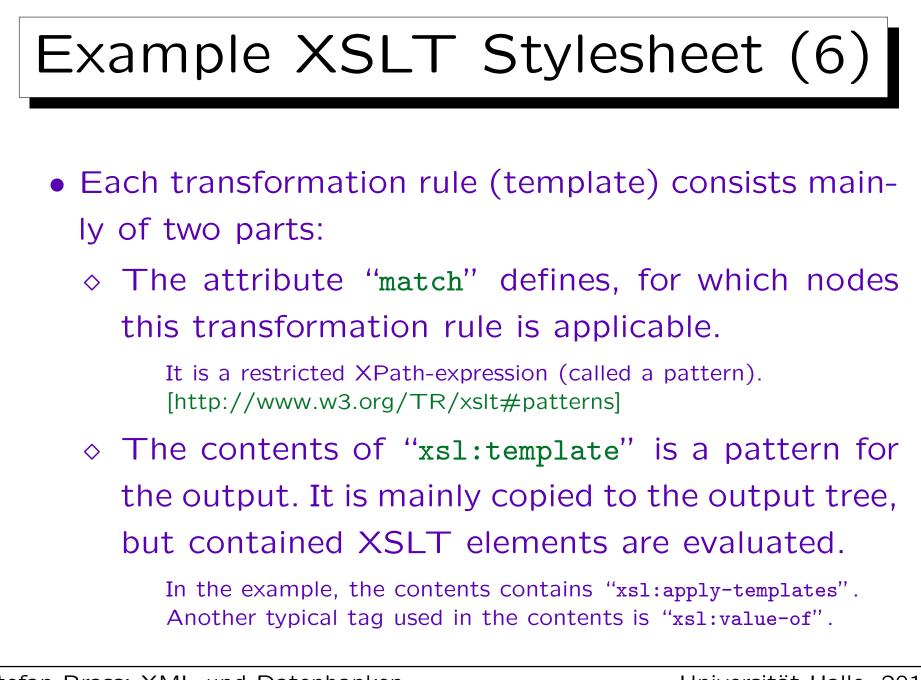
transform is allowed as a synonym. The version number is mandatory.

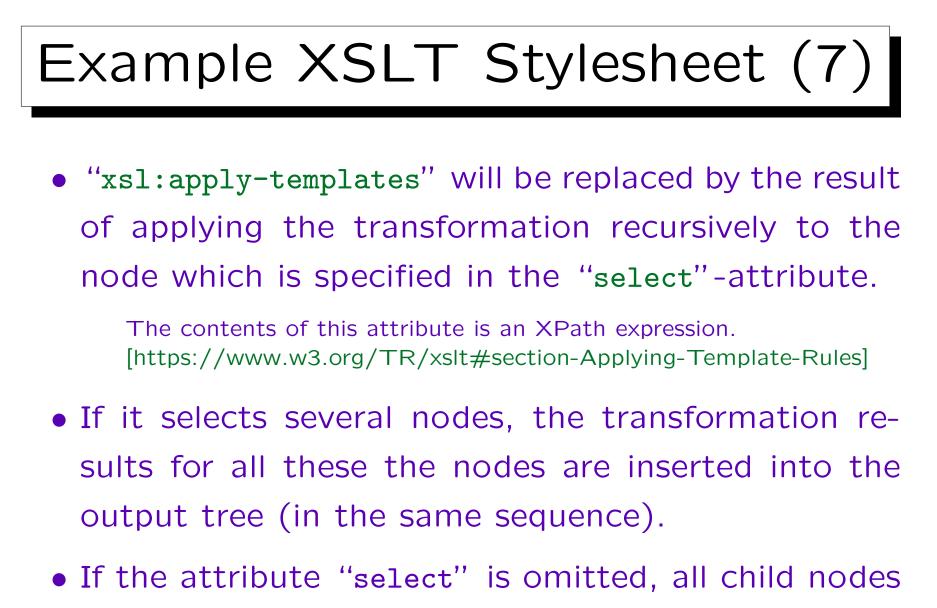


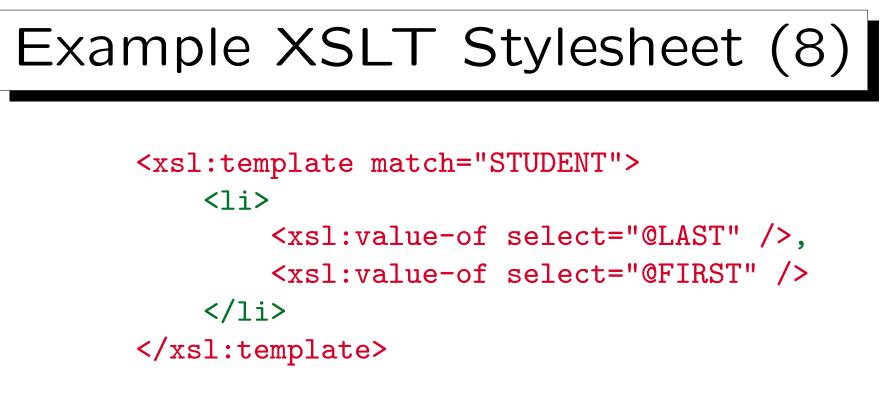


```
Example XSLT Stylesheet (4)
<rsl:template match="/">
   <html>
       <head><title>Students</title></head>
       <body>
          <h1>Student List</h1>
          <rsl:apply-templates
                   select="/GRADES-DB/STUDENT"/>
          </body>
   </html>
</rsl:template>
```



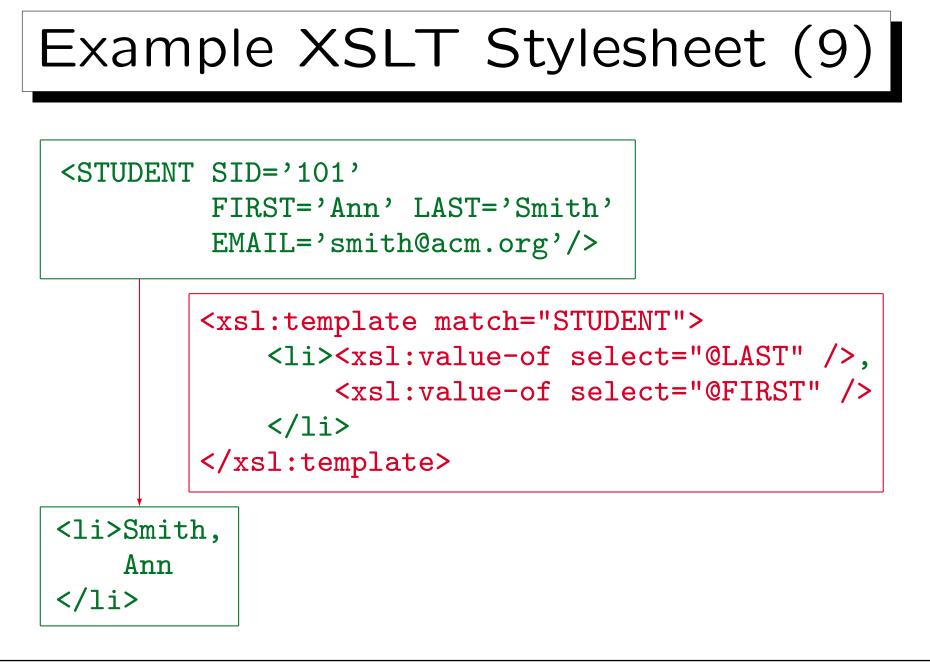






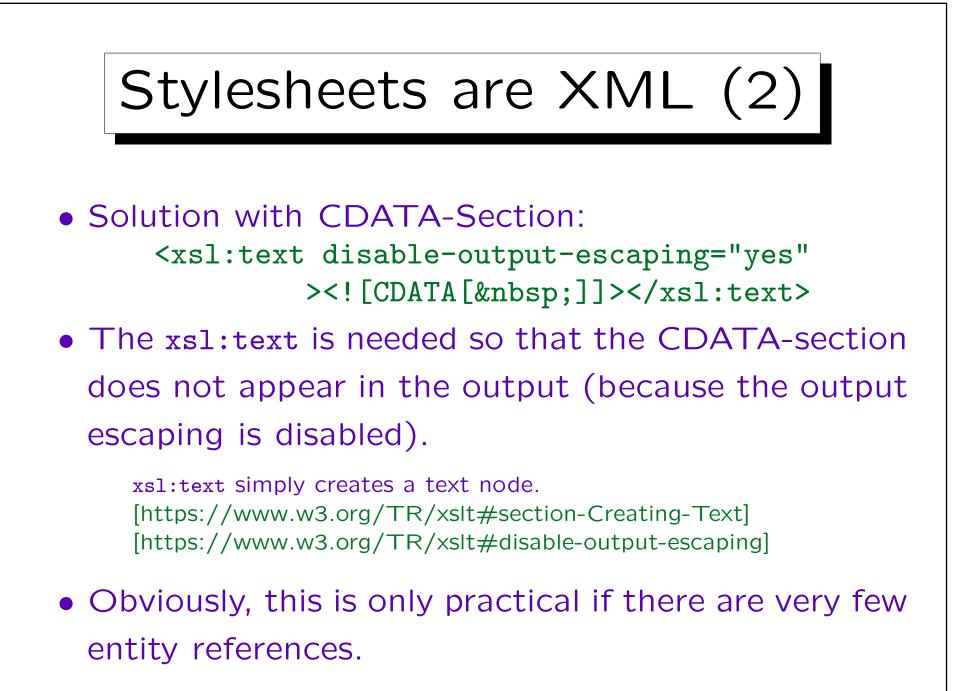
- </rsl:stylesheet>
- The result of the stylesheet is an HTML page which contains the student names, e.g. "Smith, Ann" as an unordered list.

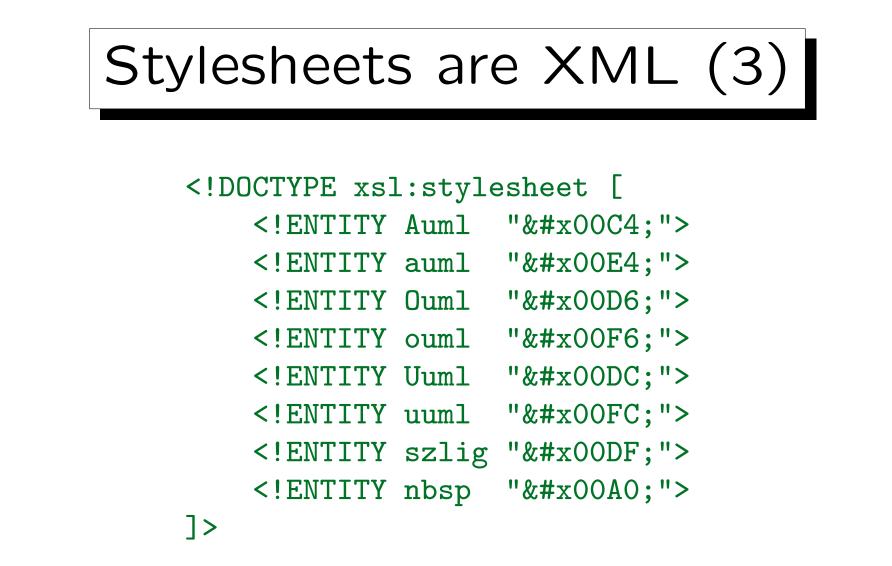
value-of adds the value of the XPath-expression converted to a string.



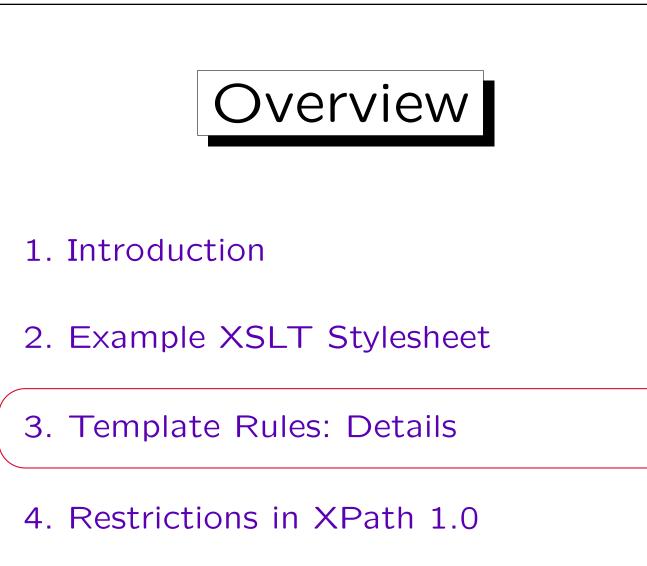
## Stylesheets are XML (1)

- Note that XSLT stylesheets must be well-formed XML. Thus, even if HTML is generated, one must e.g. write "<br />" and not simply "<br>".
- XML has only the five predefined entities "<", ">", "'", "&quote;", "&".
- To use other HTML entities (e.g. " "):
  - ♦ declare them in a local DTD part in the DOC-TYPE declaration (see below), or
  - $\diamond\,$  put them into a CDATA section (see below), or
  - ◊ write a character reference: for .

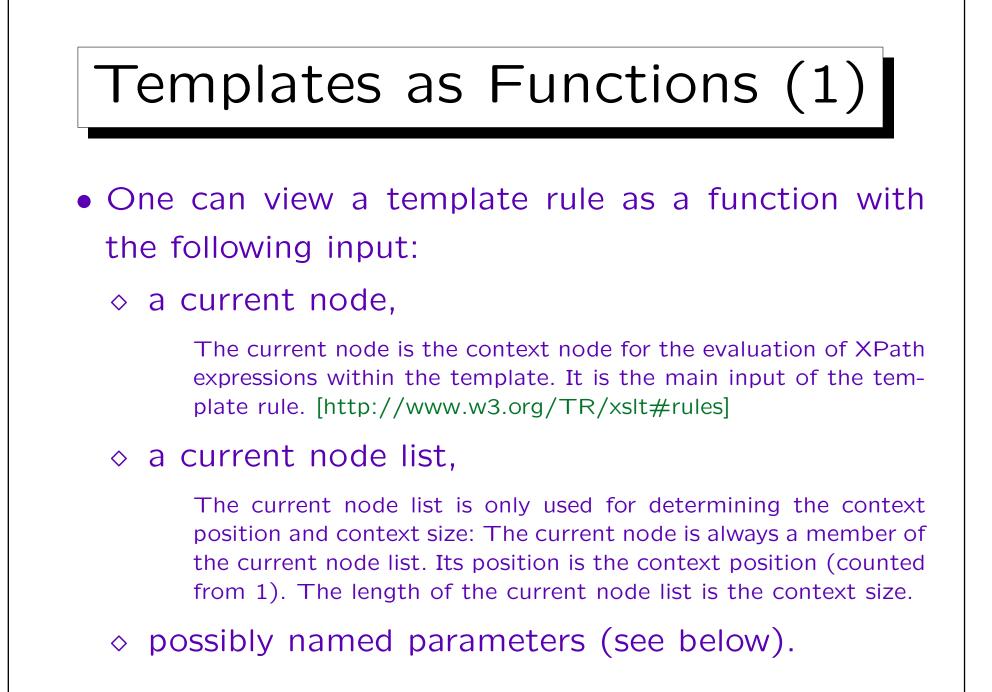




The numbers can be taken from the HTML DTD or the Unicode standard or [http://www.w3.org/2003/entities/2007/w3centities-f.ent].



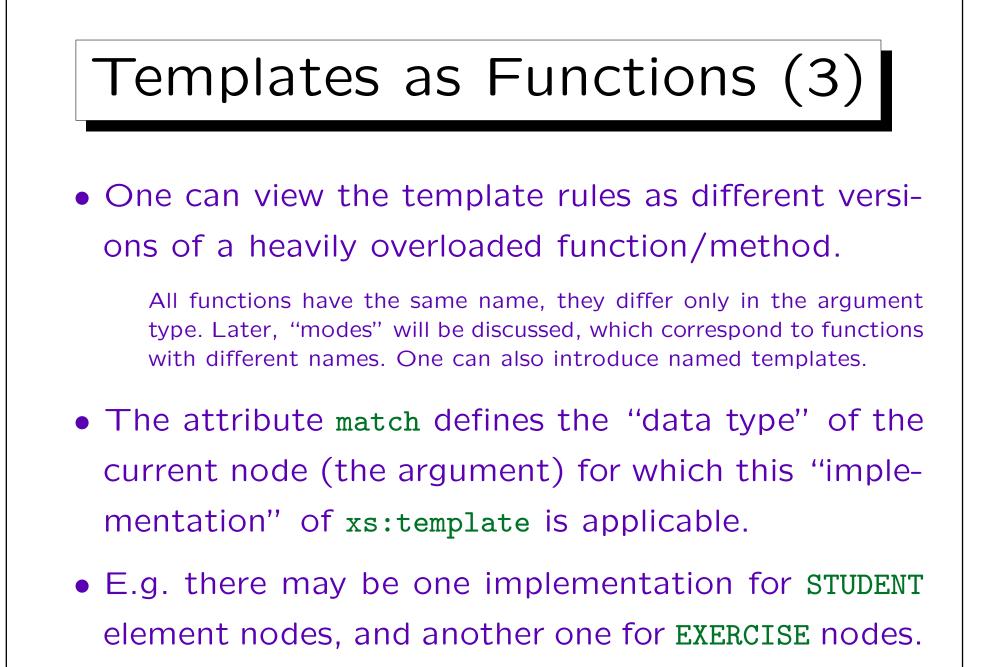
5. More XSLT Constructs

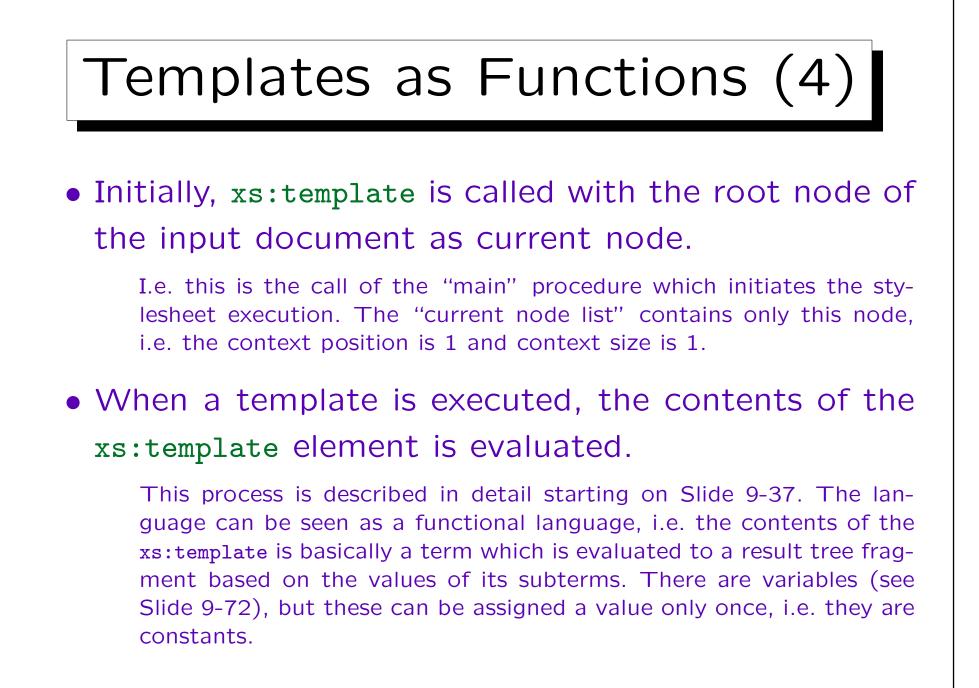


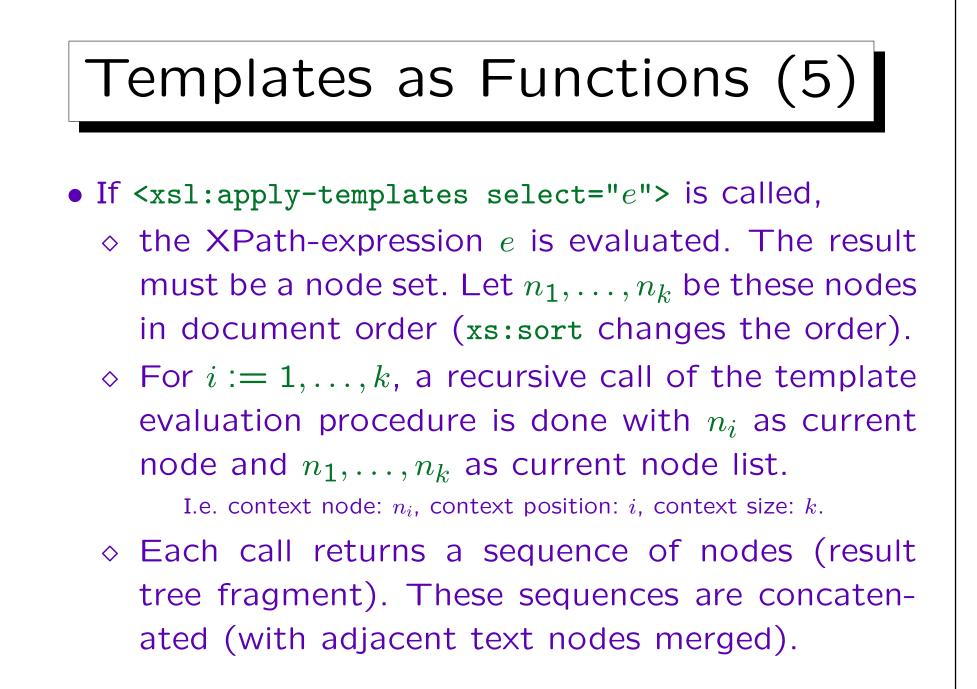
### Templates as Functions (2)

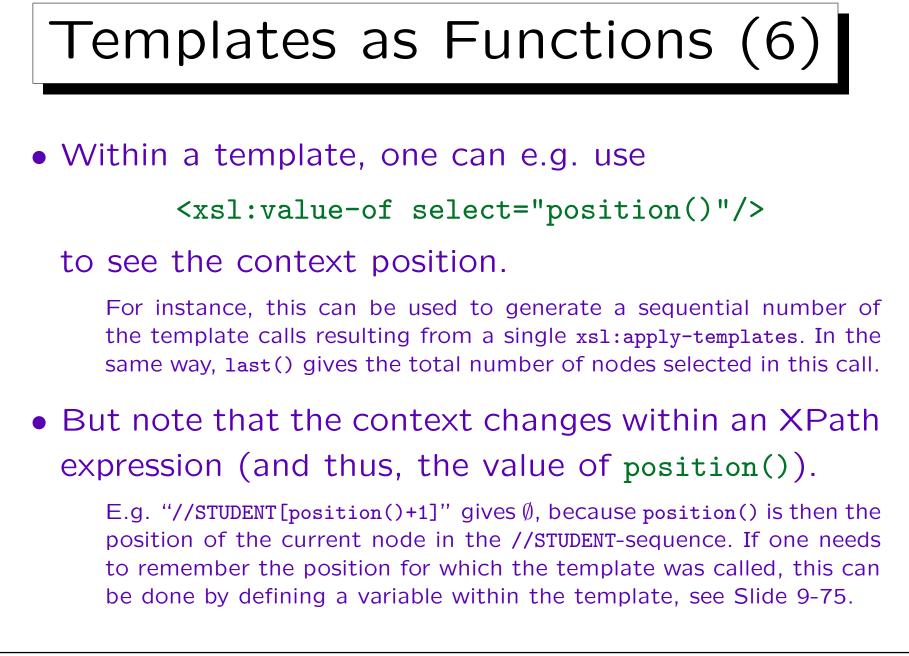
#### • The output is a "result tree fragment".

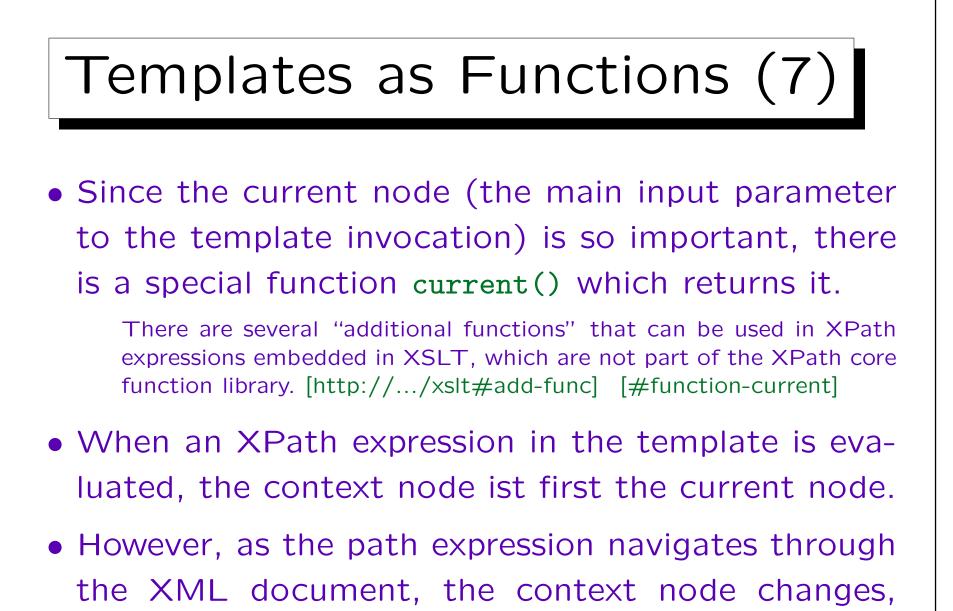
This is a root node with children. The root node is not important, it is normally removed when the "result tree fragment" is inserted into another "result tree fragment". I.e. it could also be simply formalized as a list of nodes. However, because a root node cannot have attribute nodes as children, a result tree fragment cannot contain attribute nodes (except within element nodes). The special type "result tree fragment" as opposed to node set is also needed because one is not allowed to use /, // and [...] on result tree fragments (only the string value can be used). It is possible to define variables that have result tree fragments as values, (see Slide 9-72), but one cannot do much with it except inserting it into the output document. In particular, one cannot apply templates to the result of other templates. [http://www.w3.org/TR/xslt#section-Result-Tree-Fragments]

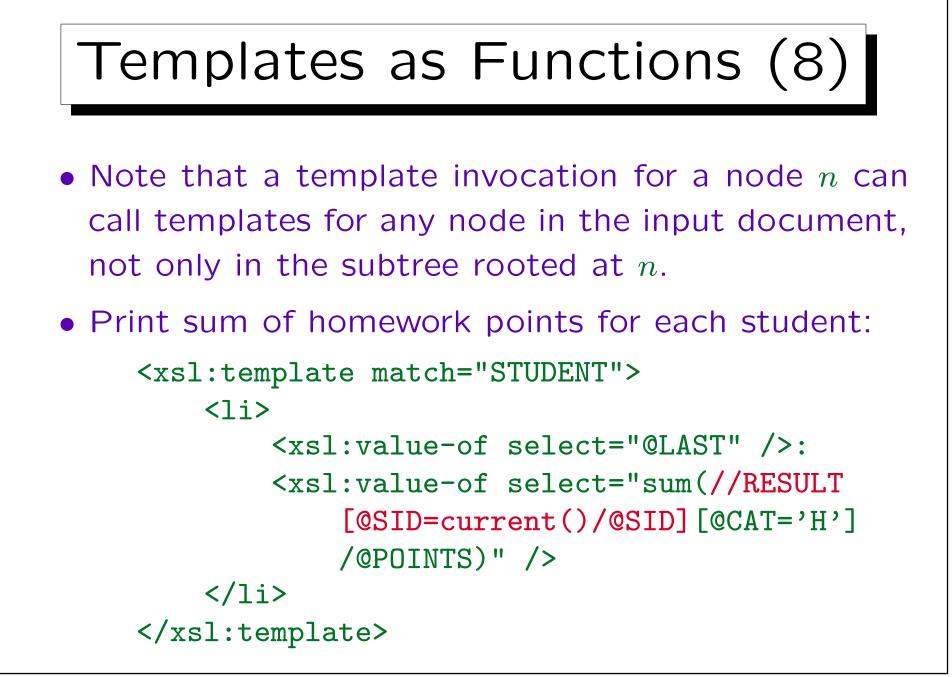


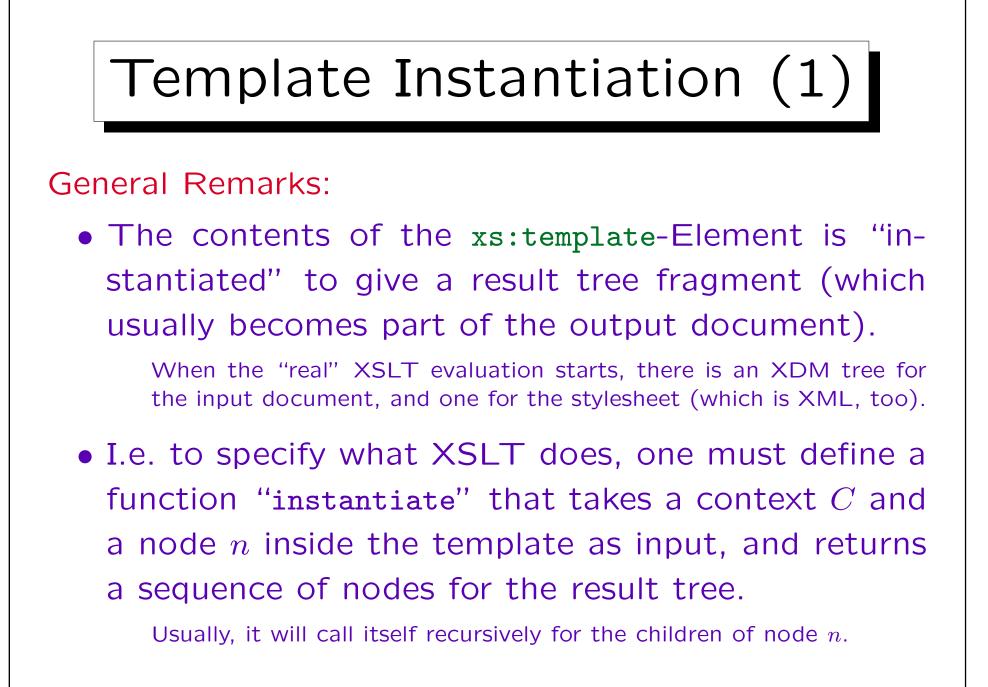


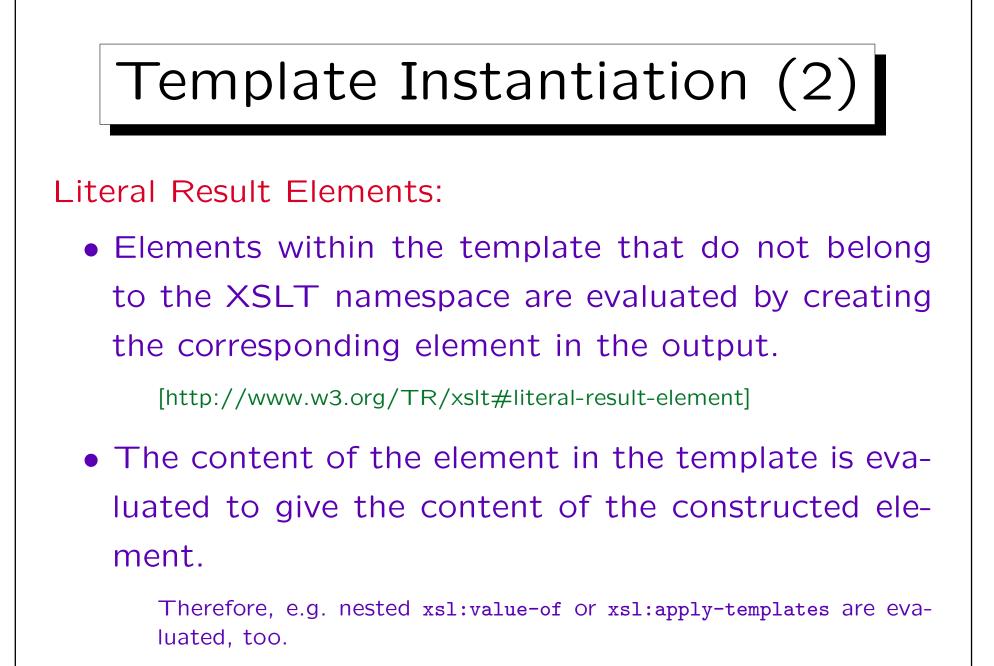


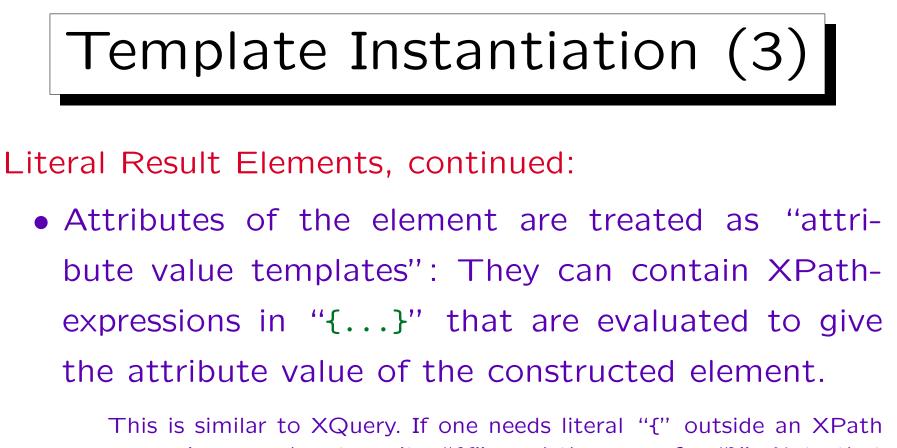




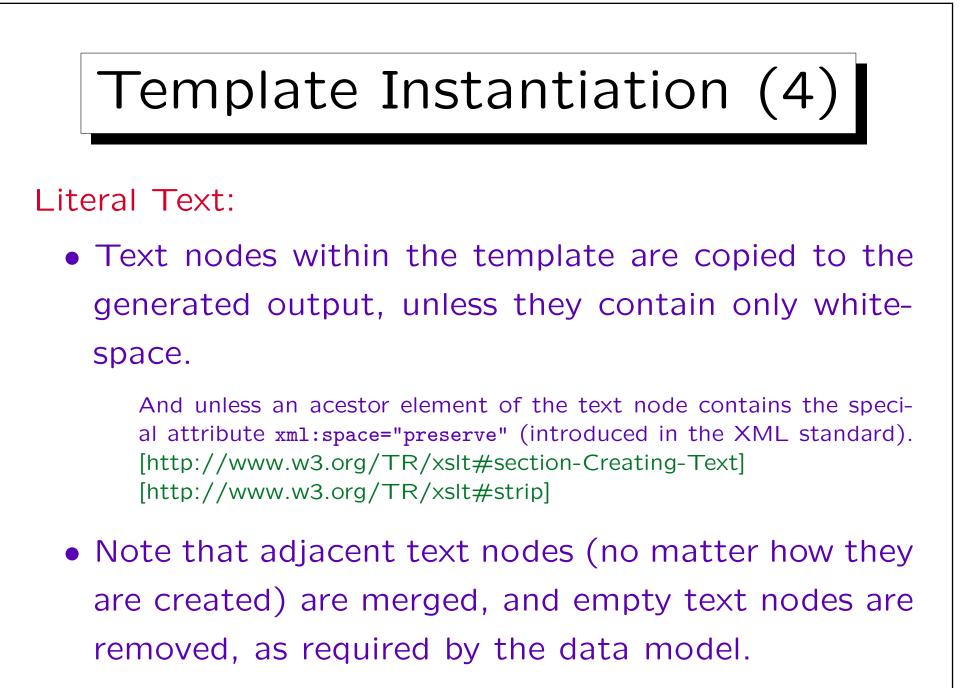


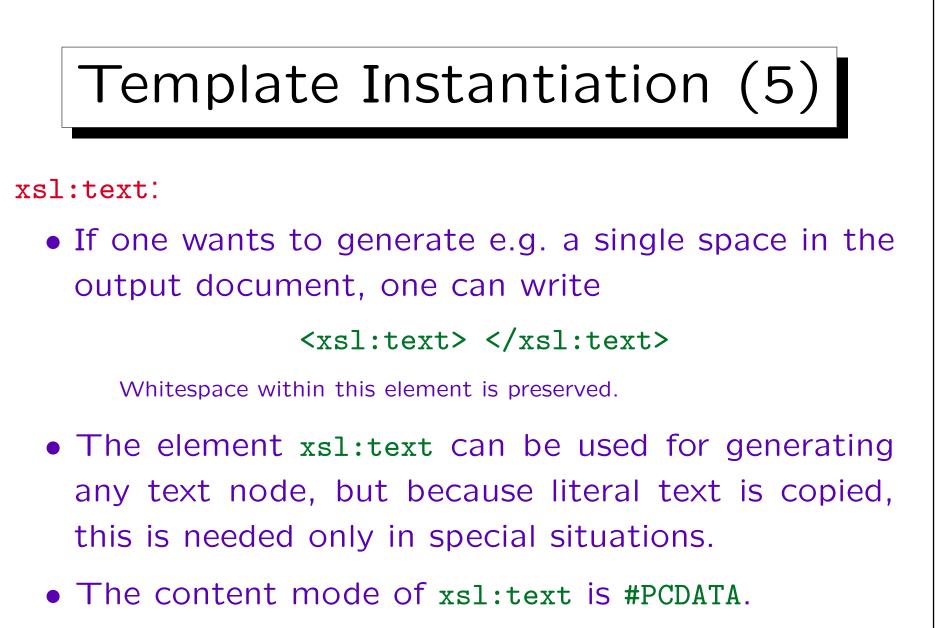




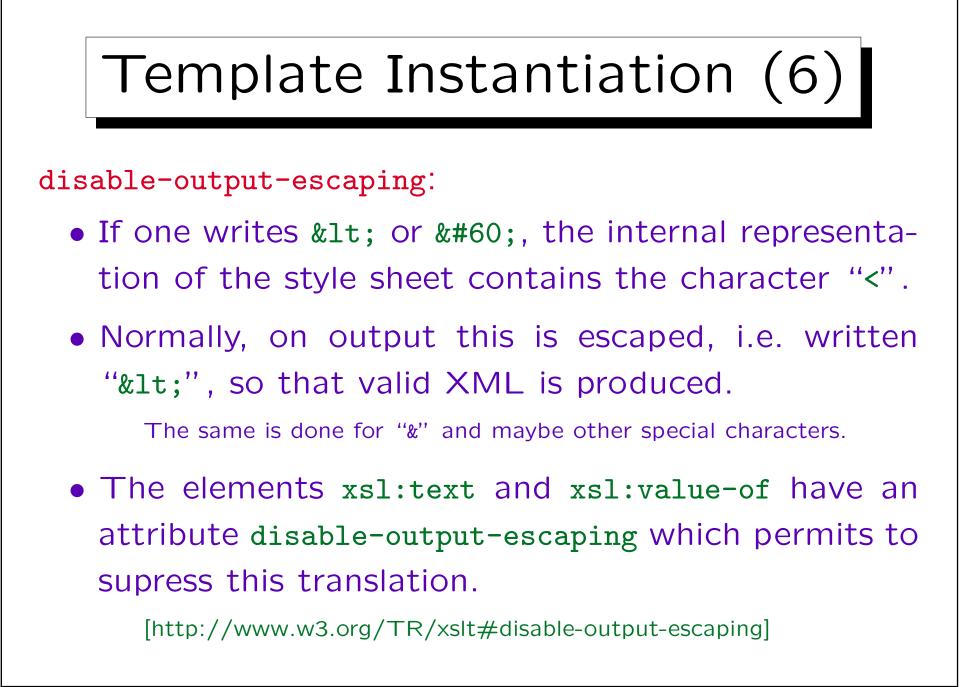


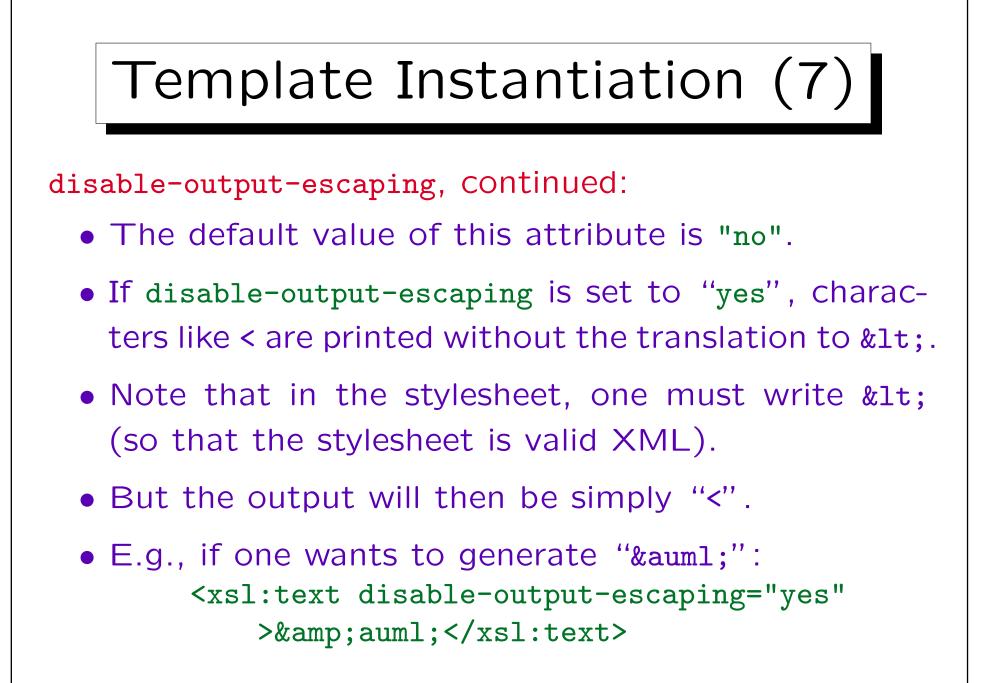
expression, one has to write " $\{\{", and the same for "\}".$  Note that this special interpretation of " $\{...\}$ " happens only in attribute values of literal result elements or certain attributes of some XSLT elements. Within element content, " $\{...\}$ " is not interpreted. [http://www.w3.org/TR/xslt#dt-attribute-value-template]

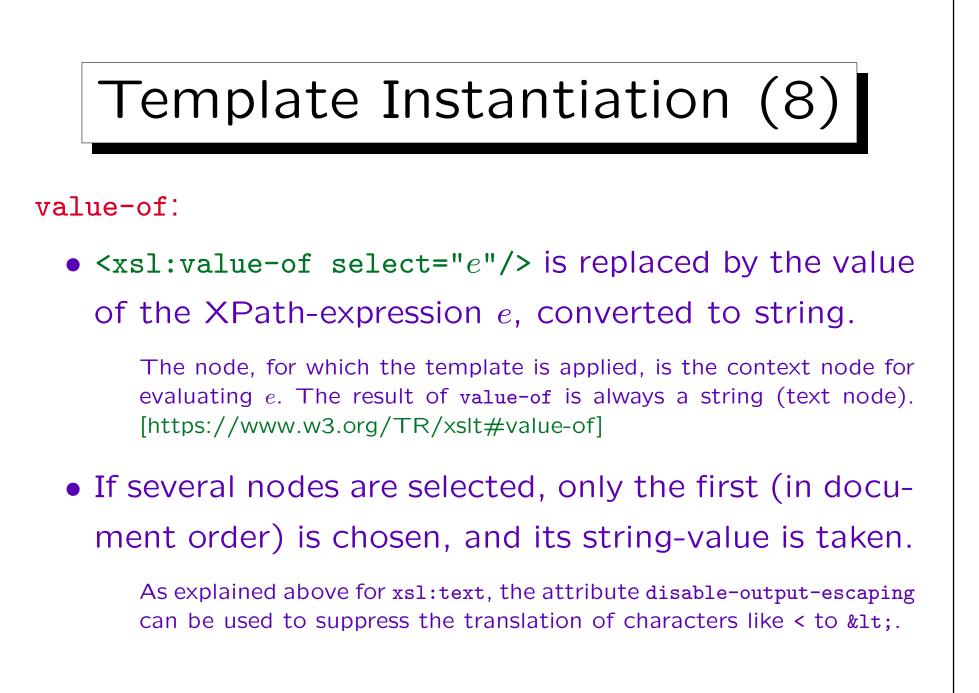




I.e. pure text without nested elements.







### Template Instantiation (9)

#### apply-templates:

- <xsl:apply-templates select="e"/> is replaced by the result of doing the "template rule" transformation recursively for all nodes that in the result of the XPath-expression e. See Slide 9-33.
- If select is missing, all child nodes are selected.

xsl:apply-templates can contain xsl:sort  $\rightarrow$ 9-80 and xsl:with-param  $\rightarrow$ 9-83. Furthermore, it has an attribute mode  $\rightarrow$ 9-82. [http://www.w3.org/TR/xslt#section-Applying-Template-Rules]

• Of course, a template can contain any number of calls to xsl:apply-templates, not only one.

### Template Instantiation (10) xsl:copy: • <xsl:copy>C < /xsl:copy> copies the current node (the node for which the template was called) and replaces its children by the result of evaluating C. I.e. the node is copied, but not its children or attributes. The copied node can be any kind, it does not have to be an element node. [http://www.w3.org/TR/xslt#copying] • The identity transformation can be specified as: <rsl:template match="@\*|node()"> <rsl:copy> <xsl:apply-templates select="@\*|node()"/> </r> </rsl:template>

### Template Instantiation (11)

#### xsl:copy-of:

 <xsl:copy-of select="e"/> copies the result of evaluating e into the result of the template, including all descendant nodes.

[http://www.w3.org/TR/xslt#copy-of]

• I.e. xsl:copy-of can be used to copy portions of the input XDM tree into the output.

The input XDM tree is subject to the removal of pure whitespace text nodes, as defined by <xsl:strip-space elements="A B C"/> (whitespace text nodes that are children of the named elements A, B and C will be removed), see [http://www.w3.org/TR/xslt#strip].

### Template Instantiation (12)

#### xsl:element:

• <xsl:element name="n">C</xsl:element> generates an element node with name n and content that is the result of evaluating C.

[http://.../xslt#section-Creating-Elements-with-xsl:element]

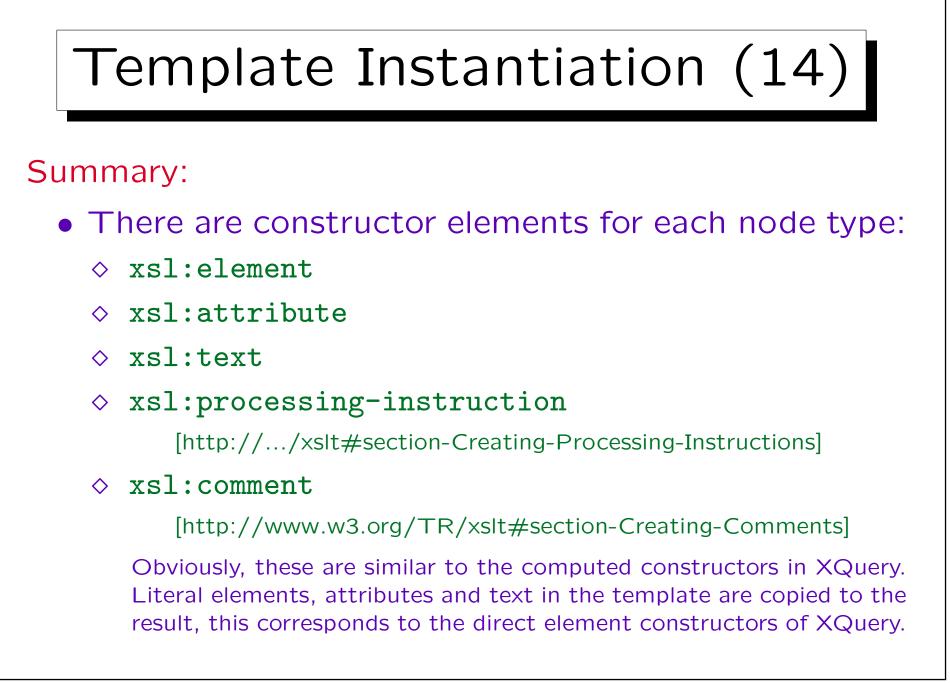
• One can use {e} in the attribute value of name to include XPath-expressions that are evaluated and replaced by the result (converted to a string).

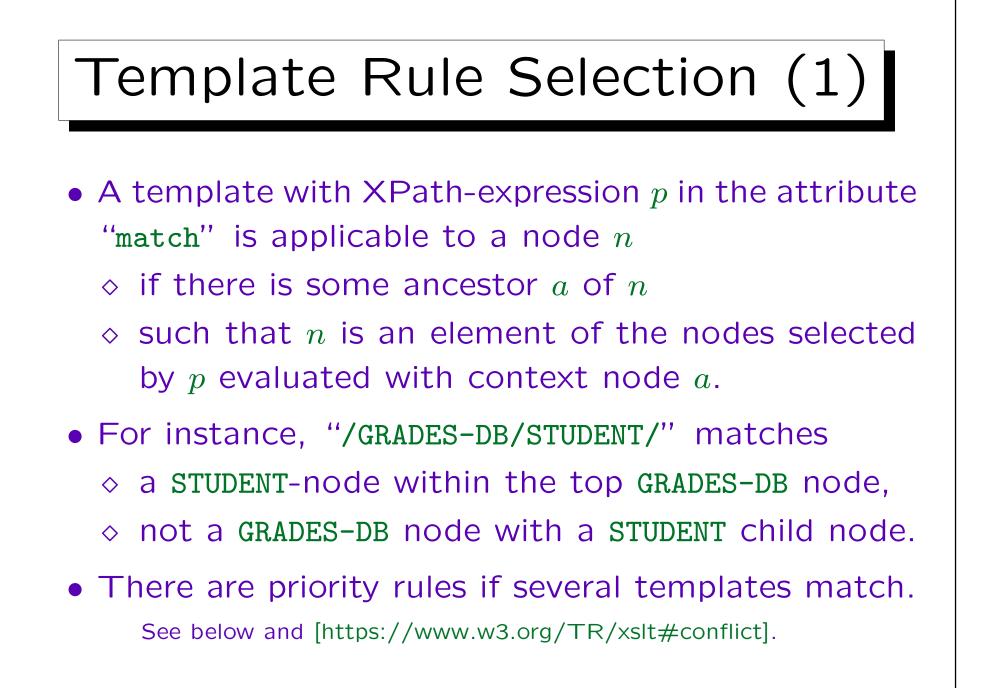
I.e. the attribute value is interpreted as "attribute value template". This possibility to compute the element name is probably the reason for using xsl:element, because otherwise one could have written simply the element itself.

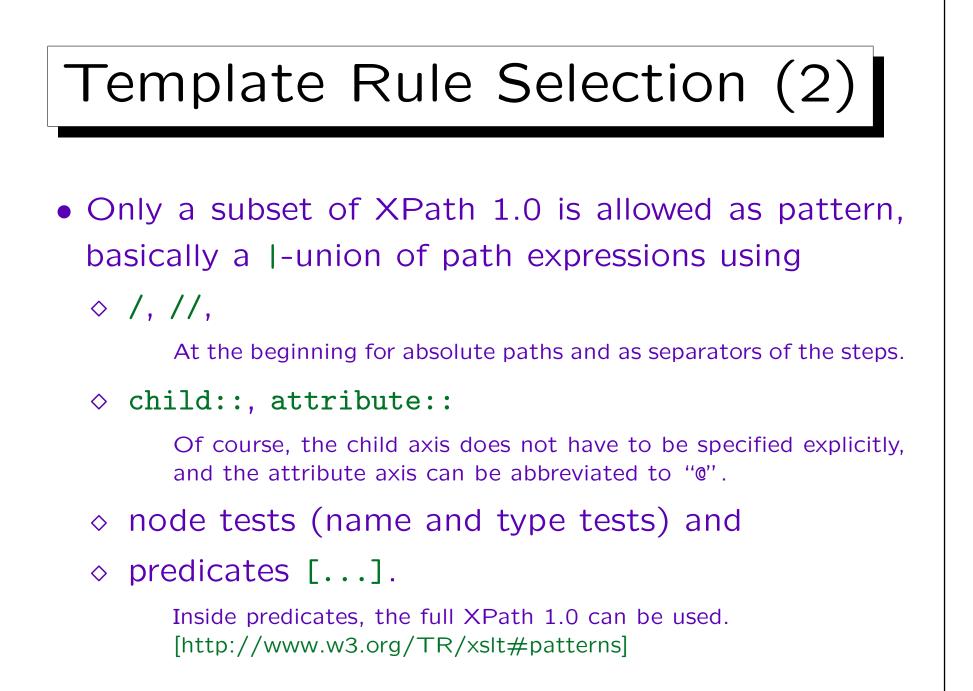


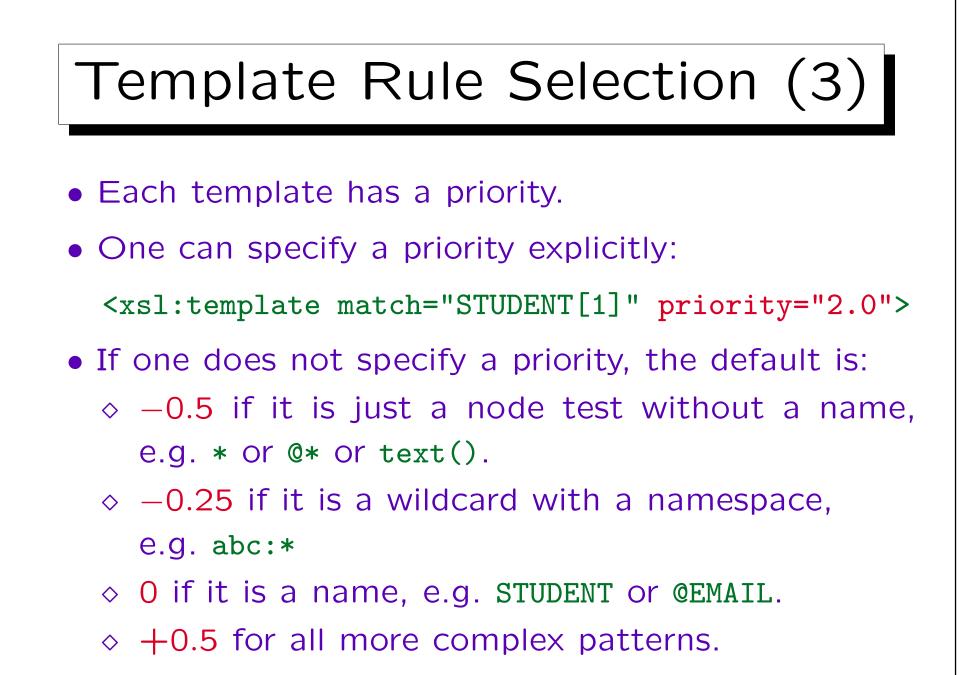
#### xsl:attribute:

- <xsl:attribute name="n">C</xsl:attribute>
  generates an attribute node with name n and the
  result of evaluating C as value.
  - One can use {e} in the attribute value of name (see xsl:element above). The result of evaluating C must be a text node (e.g., element nodes cannot become an attribute value). [http://www.w3.org/TR/xslt#creating-attributes]
- The generated attribute node is assigned to the enclosing element.
  - Attribute nodes must come first in the content, they cannot be added, e.g., after a text node.



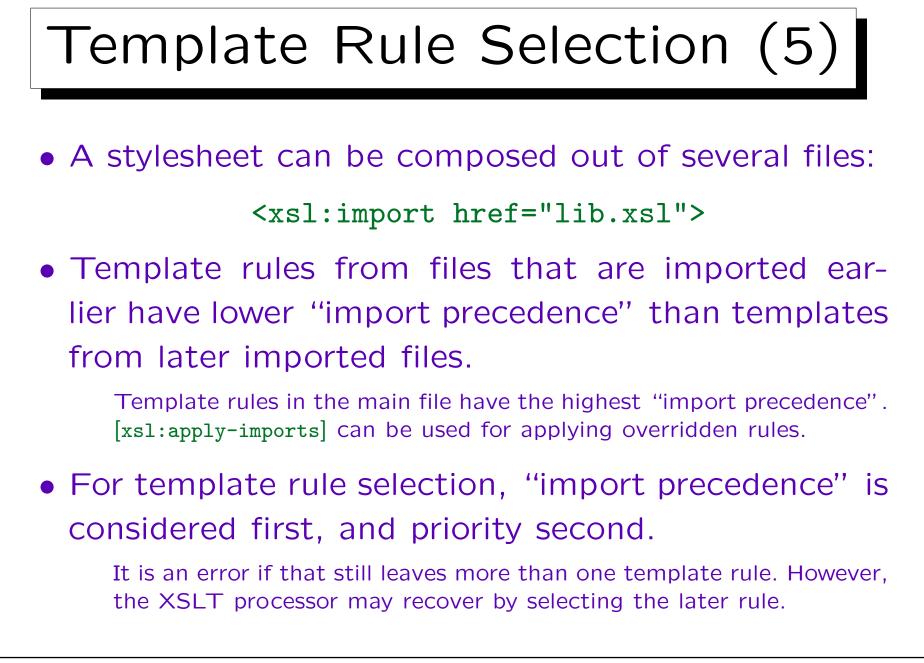


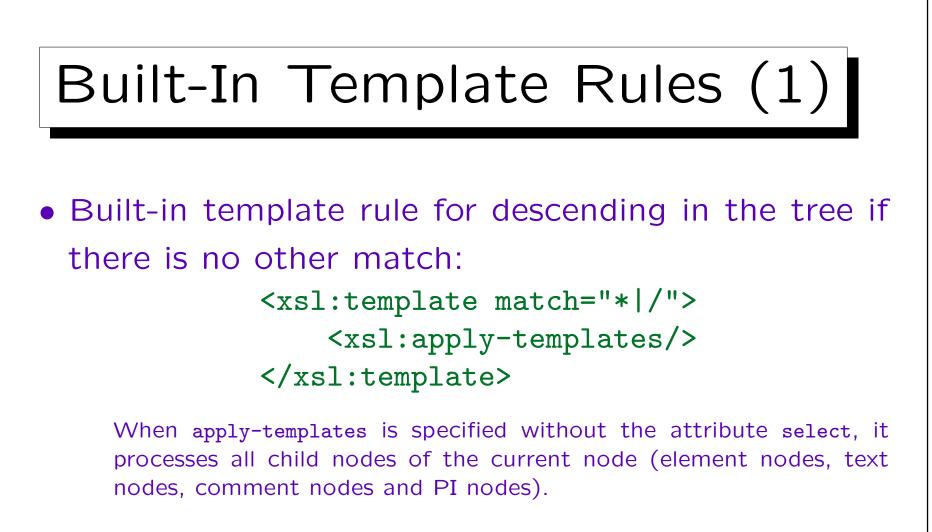




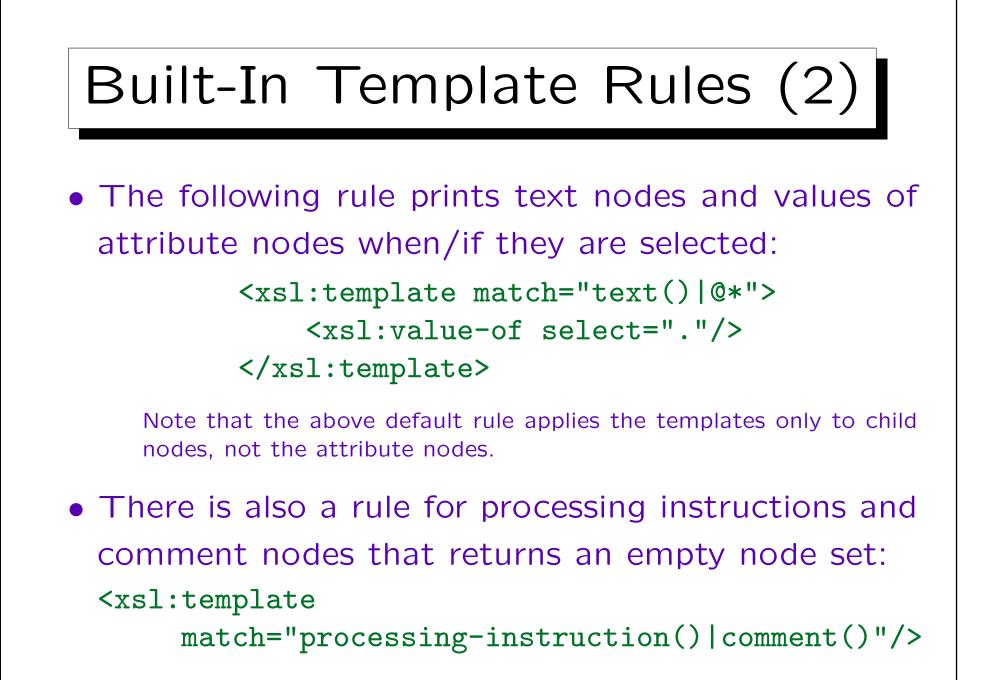
# Template Rule Selection (4)

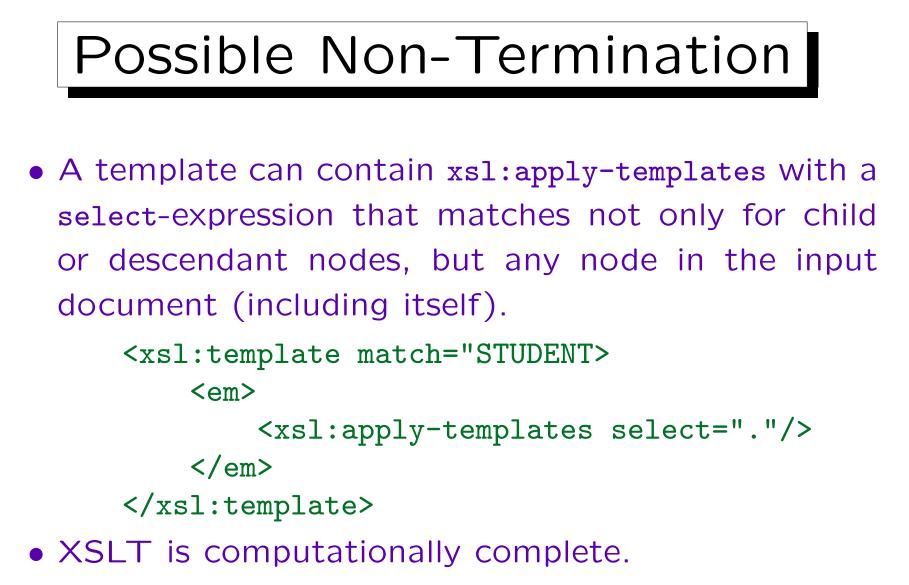
- If a pattern is composed with |, then the template is treated like several templates, one for each alternative.
- E.g. if the pattern is "STUDENT|EXERCISE[1]", the template rule has priority
  - $\diamond$  **0**, if applied to a STUDENT-element, and
  - ◊ 0.5, if applied to the first EXERCISE-child of its parent.



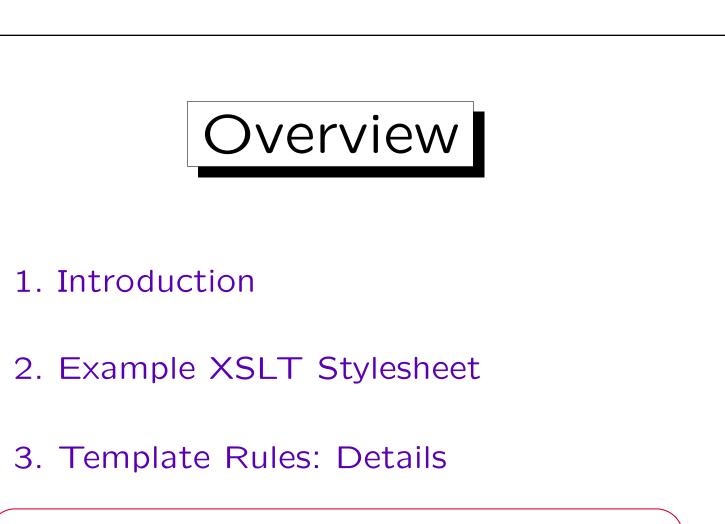


• Built-in rules have the lowest possible import precedence. Thus, if there is another rule matching the node, this other rule is chosen.



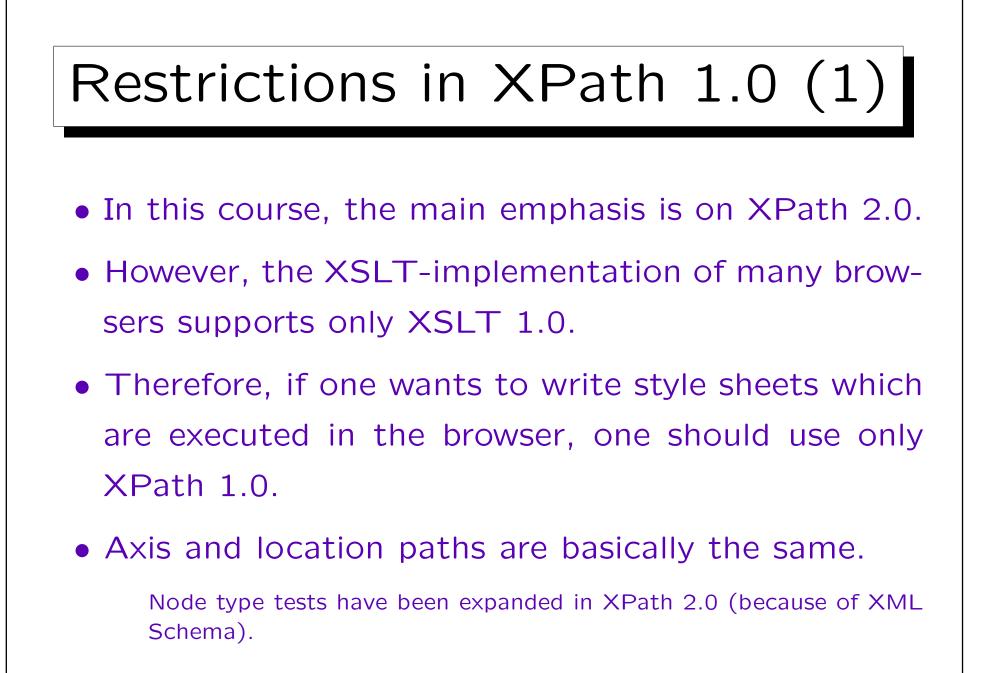


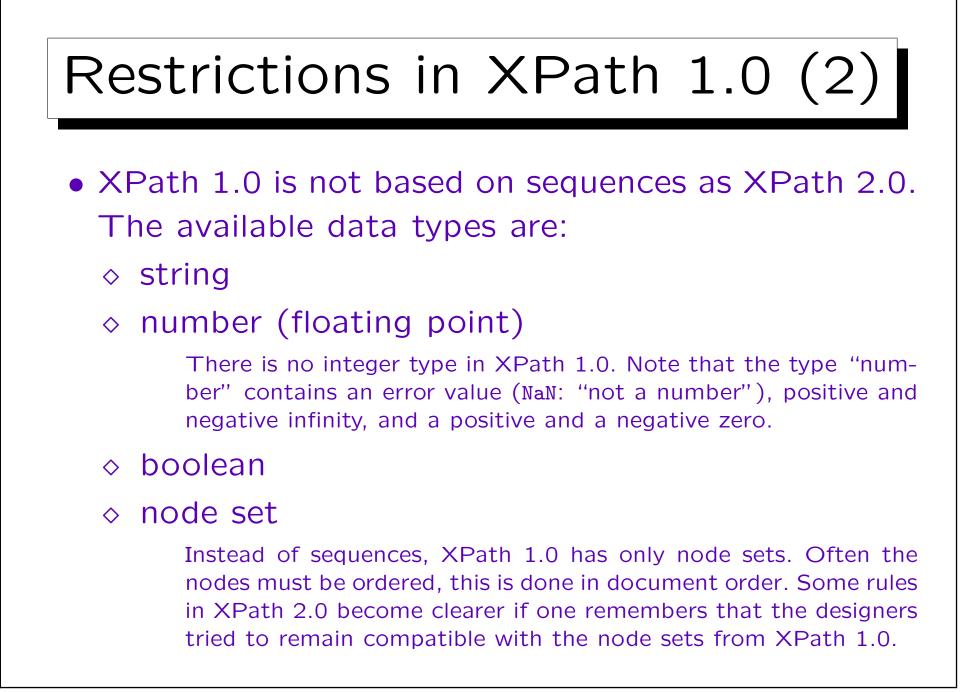
One can simulate any Turing machine (or other computation model) in it. [http://www.unidex.com/turing/utm.htm]

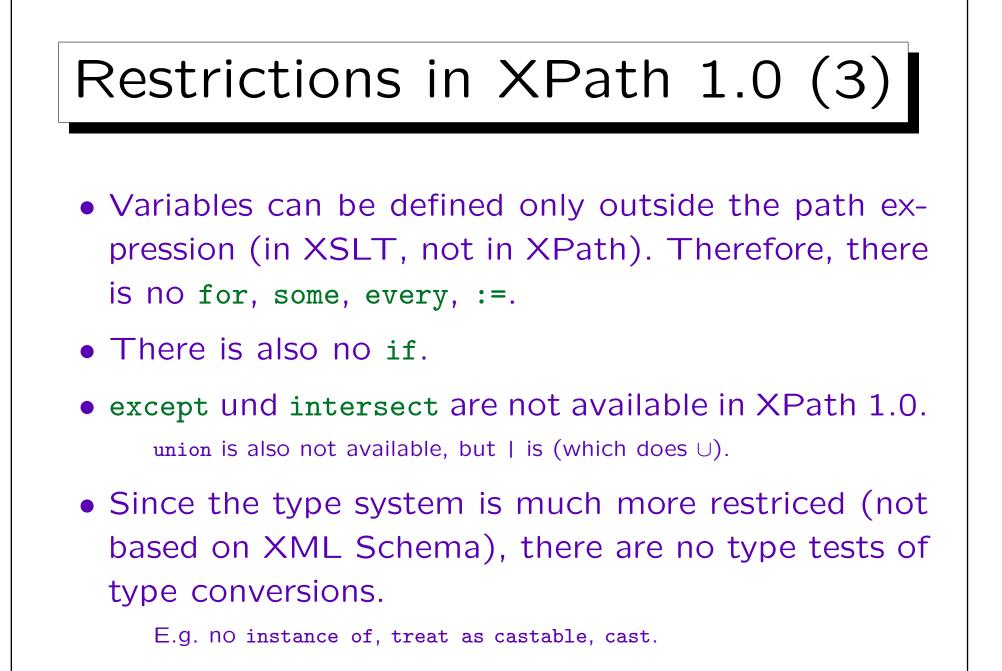


4. Restrictions in XPath 1.0

5. More XSLT Constructs

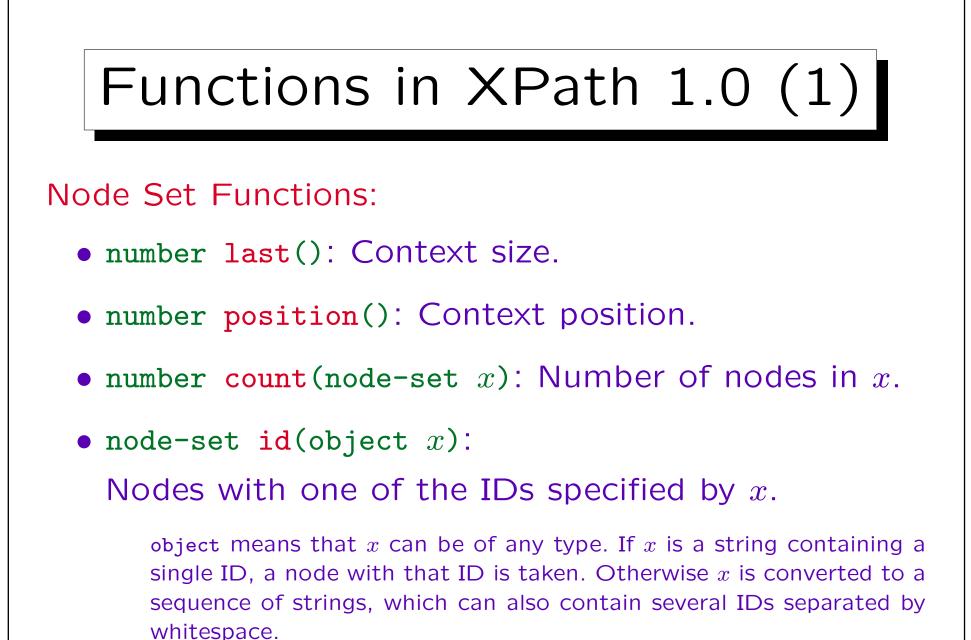


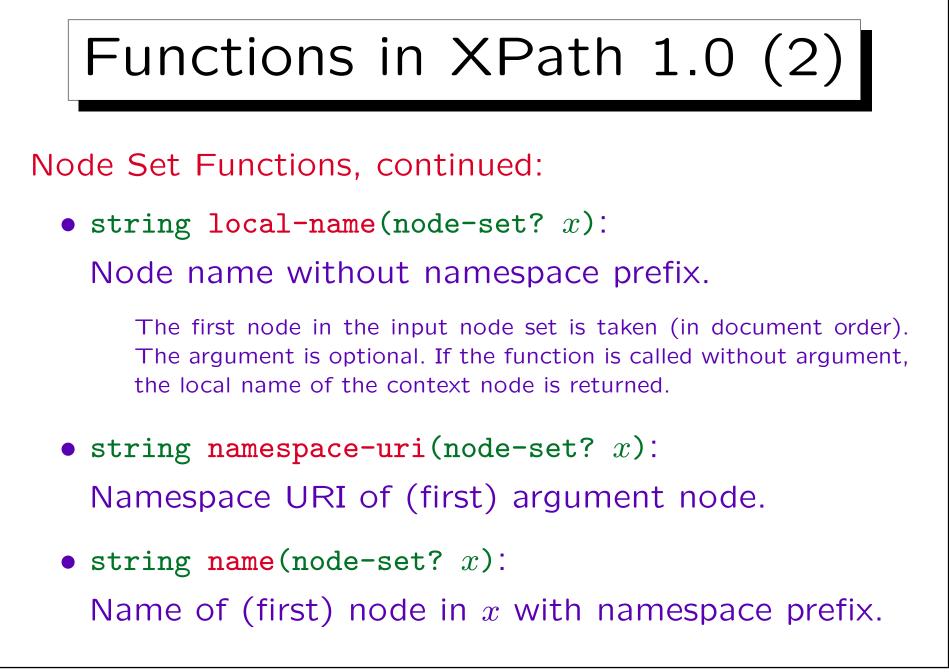


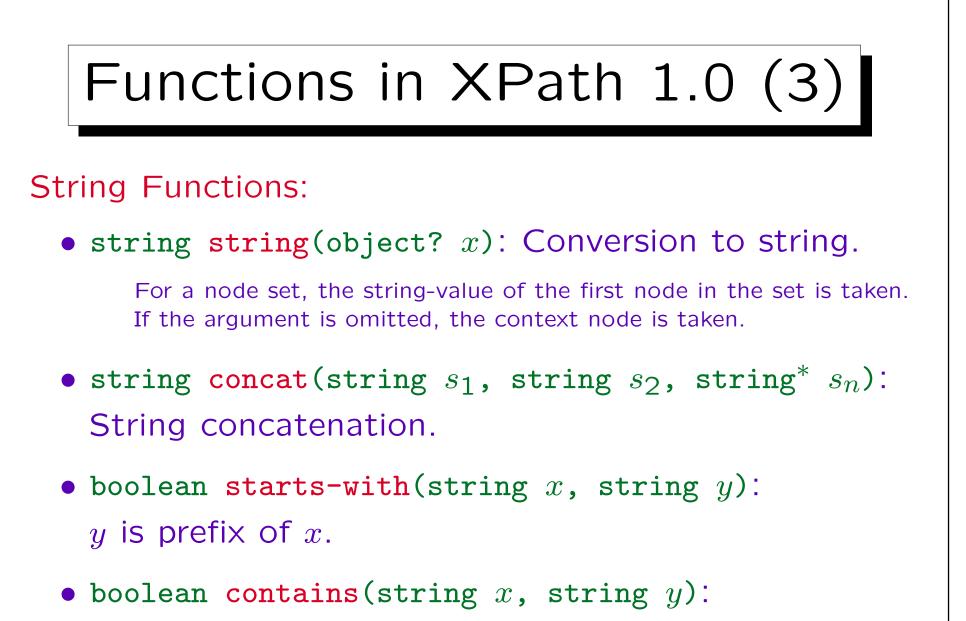


## Operators in XPath 1.0

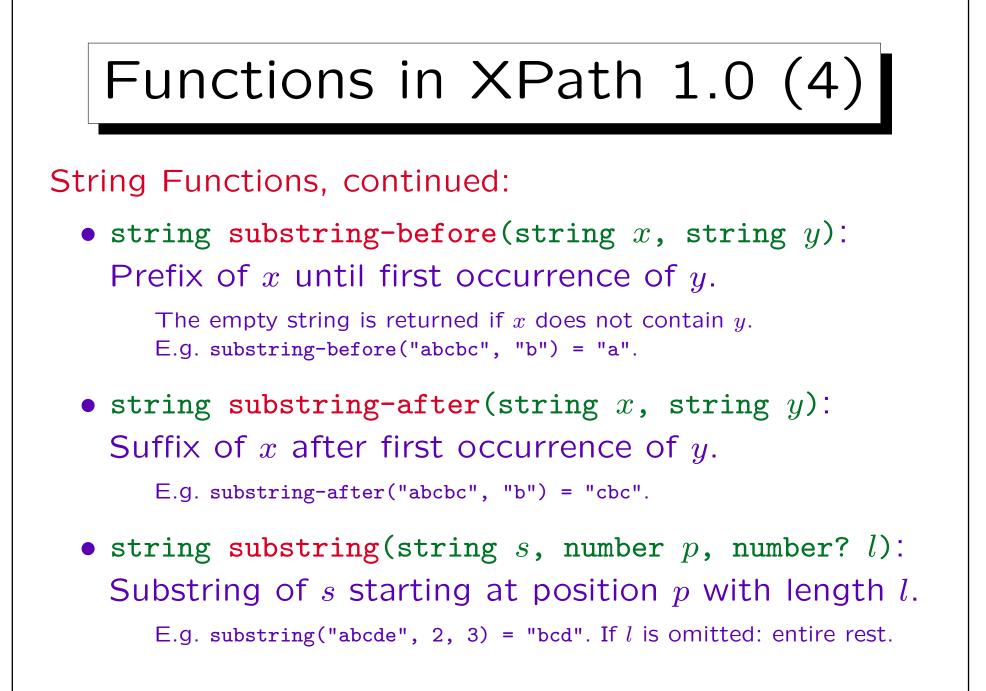
Prio	Operator	Assoc.
1	or	left
2	and	left
3	=, !=	left
4	<, <=, >, >=	left
5	+, -	left
6	*, div, mod	left
7	- (unary)	right
8		left
9	/, //	left
10	[]	left

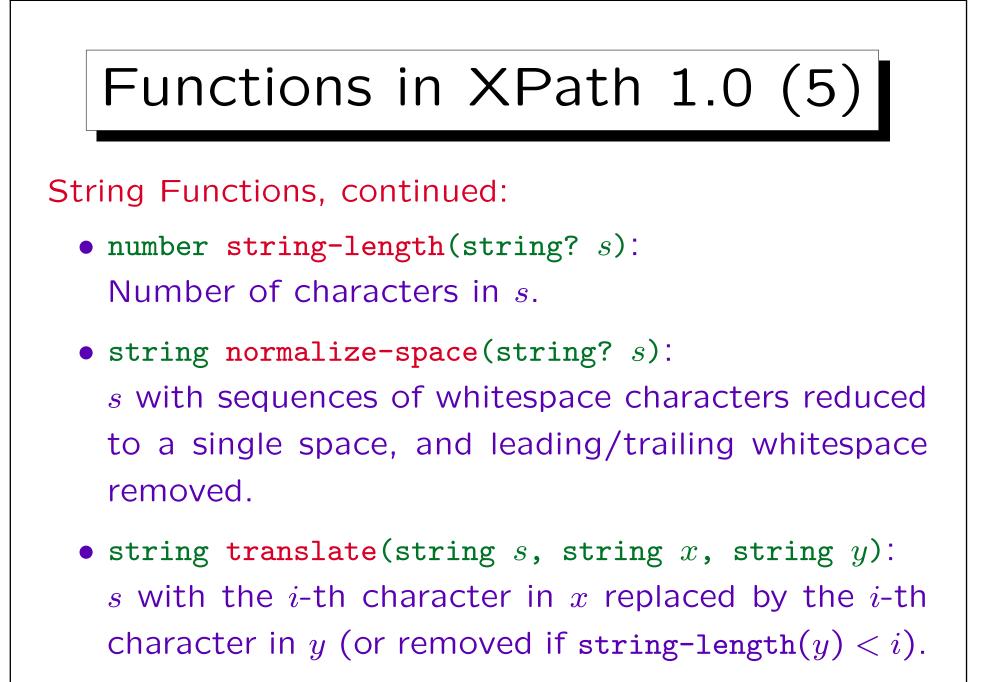






y is substring of x.





## Functions in XPath 1.0 (6)

### **Boolean Functions:**

- boolean boolean(object x): Conversion to boolean.
   See "effective boolean value": E.g. node set is true iff it is not empty.
- boolean **not**(boolean b): Negation.
- boolean true(): Constant value "true".
- boolean false(): Constant value "false".
- boolean lang(string l):

Language of the context node is l.

E.g. if xml:lang = "en-us" is specified in an ancestor node of the context node (and no other language is specified in between), lang("en") is true. If no language is specified, lang returns false.

# Functions in XPath 1.0 (7)

### Number Functions:

• number number(object? x): Conversion to a number.

If a string has no numeric format, it is converted to the special floating point value "NaN" ("not a number", error value).

- number sum(node-set x):
   Sum of the result of converting the string-value of each node in x to a number.
- number floor(number x): Largest integer  $\leq x$ .
- number ceiling(number x): Smallest integer  $\geq x$ .
- number round(number x):
   x rounded to nearest integer (.5 is rounded up).



- 1. Introduction
- 2. Example XSLT Stylesheet
- 3. Template Rules: Details
- 4. Restrictions in XPath 1.0
- 5. More XSLT Constructs



- XPath 1.0 has no constructs that can introduce new variables, or assign a value to a variable.
- But one can use \$x to access the value of a variable x defined in the given context.
- In XSLT 1.0, a variable can be declared and assigned a value as follows:

```
<rsl:variable name="version" select="1"/>
```

• One can use this e.g. with

<rsl:value-of select="\$version"/>

## Variables (2)

- One can use this result tree fragment as follows: <xsl:copy-of select="\$table\_headline"/>
- Note: <xsl:variable name="n" select="2"> is not the same as <xsl:variable name="n">2</xsl:variable>.

In the first case, the value is the number 2, in the second case a node with value 2. E.g. //student[\$n] works in the first case, but in the second, one must write //student[position()=\$n].

Variables (3)

- Variable declarations can appear on the top level,
   i.e. as child of xsl:stylesheet. Then they are global and can be accessed everywhere.
  - If they are defined by a template, this is evaluated with the root node of the input document as context node.
- Variable declarations can also be written inside a template, then they are local, and the variable can be accessed only within that template.

And only after it is defined. This is different from global variables, which are available even before their point of definition (but cycles are forbidden). [http://www.w3.org/TR/xslt#variables]

Variables (4)

 A defined variable cannot be assigned a new value (i.e. it should be understood like a constant).

A global variable can be shadowed by a local variable in a template.

• This permits very different evaluation algorithms, e.g. on parallel hardware.

The language is declarative (functional), not imperative.

• Of course, a variable within a template gets a new value for each template invocation:

<xsl:template match="STUDENT">
 <xsl:variable name="no" select="position()"/>

```
Repetition: for-each (1)
• With the for-each construct, one can embed a tem-
 plate directly into another template:
   <rsl:template match="GRADES-DB">
       <for-each select="STUDENT">
               <1i>
                  <value-of select="@LAST"/>,
                  <value-of select="@FIRST"/>
               </for-each>
       </rsl:template>
    [http://www.w3.org/TR/xslt#for-each]
```

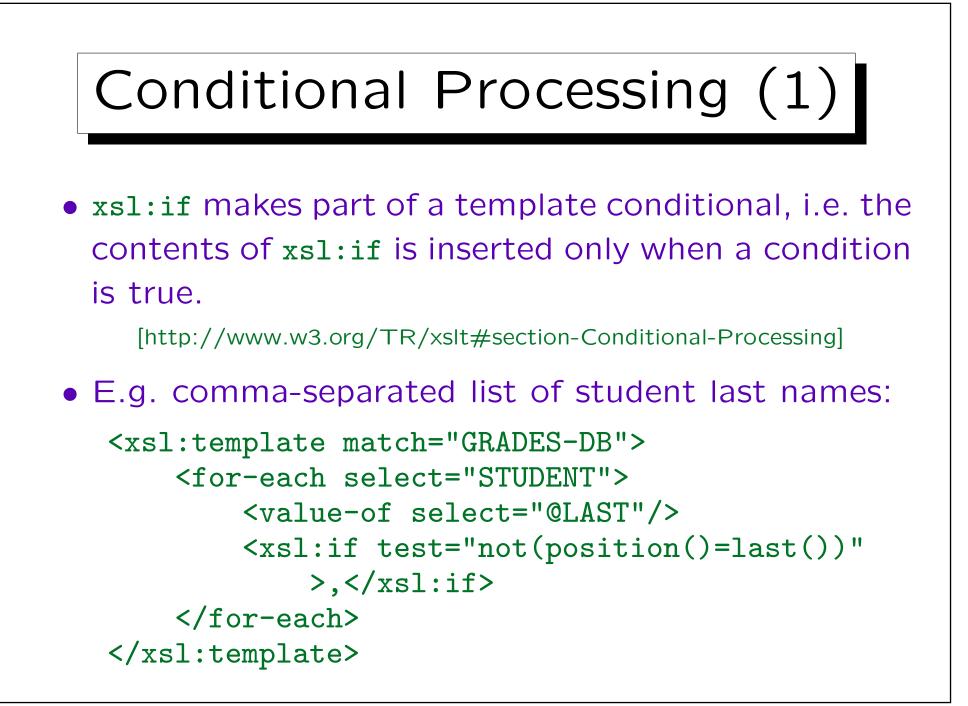
## Repetition: for-each(2)

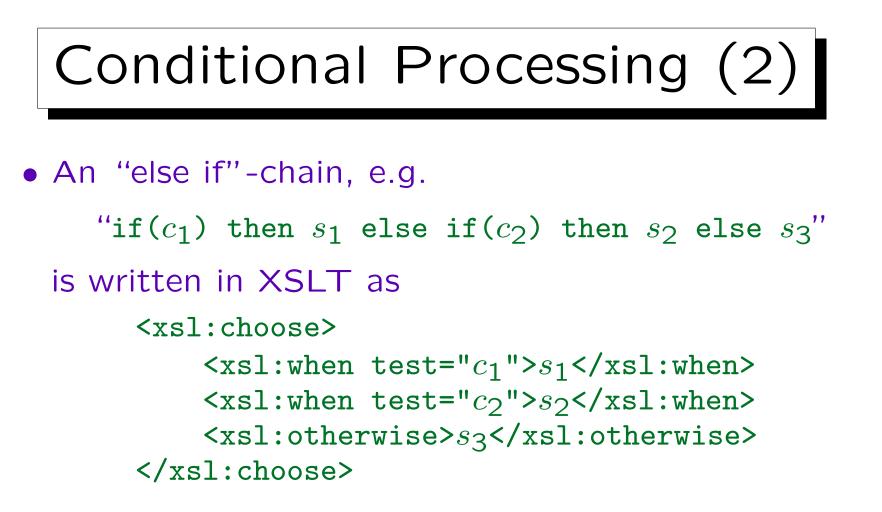
- The XPath-expression in select determines a node set  $n_1, \ldots, n_k$ , just as with apply-templates.
- The contents of the for-each element is instantiated once for each node  $n_i$  as current node, the corresponding node lists (result tree fragments) are concatenated as the value of the for-each.

Note that there is no variable, only the implicit context.

• If one uses apply-templates with a separate template, that becomes a reusable component.

Like a procedure. In contrast, for-each is like putting the procedure body directly into the place where it is called.





The content model of xsl:choose is xsl:when+,xsl:otherwise?. The template within the first xsl:when with a test-condition that evaluates to true is chosen (evaluated). If all are false, the content of xsl:otherwise is evaluated (if missing, it is treated as empty). [http://.../xslt#section-Conditional-Processing-with-xsl:choose]



- One can specify that apply-tempates and for-each should construct the result not in document order of the selected nodes, but in a specific sort order.
- E.g. print students alphabetically sorted by last name, and by first name if last names are equal: <xsl:template match="GRADES-DB">

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

```
Stefan Brass: XML und Datenbanken
```



- Attributes of xsl:sort to modify the sort order:
  - ◊ data-type="number" for numerical order.

The default is data-type="text", which means that the selected values are converted to strings. With data-type="number", they are converted to numbers. All attributes are "attribute value templates": one can use {...}. [http://www.w3.org/TR/xslt#sorting]

- order="descending" means an inverse sort order from large to small values. Default: "ascending".
- ◊ lang="de" select a language-specific sort order.
- ◇ case-order="upper-first" requests a sort order like A a B b .... Alternative: "lower-first".



- xsl:template and xsl:apply-templates have an optional attribute mode which can be set to a name.
   Possibly with namespace prefix (QName). [http://.../xslt#modes]
- xsl:apply-templates applies only templates with a matching mode value, i.e.
  - either both have a mode value specified, and the values are equal, or
  - $\diamond$  both have no mode specified.
- I.e. the mode is something like a function name, and match specifies the argument type.



One can pass a parameter to a called template:
 <xsl:apply-templates select="//RESULT[@CAT='E']">
 <xsl:with-param name="type" select="'Exam'"/>

</r>sl:apply-templates>

The parameter definition is very similar to a variable definition. [http://.../xslt#section-Passing-Parameters-to-Templates]

 A template that uses the parameter value must declare it at the beginning of the content: <xsl:templates match="RESULT> <xsl:param name="type" select="'Unknown'"/>

The defined value is a default value. The parameter can be accessed as a variable \$type in XPath. [http://www.w3.org/TR/xslt#variables]

