**ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ**
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ**

# Εισαγωγή στην Επιστήμη και Τεχνολογία των Υπηρεσιών

## Ενότητα 6: XML, XSLT and XPATH - 2

**Χρήστος Νικολάου**
**Τμήμα Επιστήμης Υπολογιστών**





Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται στην άδεια χρήσης **Creative Commons** και ειδικότερα

***Αναφορά – Μη εμπορική Χρήση – Όχι Παράγωγο Έργο v. 3.0***

***(Attribution – Non Commercial – Non-derivatives )***


- Εξαιρείται από την ως άνω άδεια υλικό που περιλαμβάνεται στις διαφάνειες του μαθήματος, και υπόκειται σε άλλου τύπου άδεια χρήσης. Η άδεια χρήσης στην οποία υπόκειται το υλικό αυτό αναφέρεται ρητώς.

# Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.




Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

# XML
# XSLT and XPATH (Part II)
# 605.444 / 635.444

David Silberberg

Lecture 13

# XPath

- Path of document elements and attributes is an important aspect of XSLT
- XPath provides a mechanism for pointing at parts of the source XML document
- XPath expressions specify **location path** in a document
  - Step-by-step instructions about how to get to a place in the source document
  - You need to know where you are to get to where you are going
  - Thus, there is implicit **context node** in XPath.
- Primarily important for XSLT
- Also important for other XML tools such as XQuery, XPointer, etc.

# XML Documents as Trees

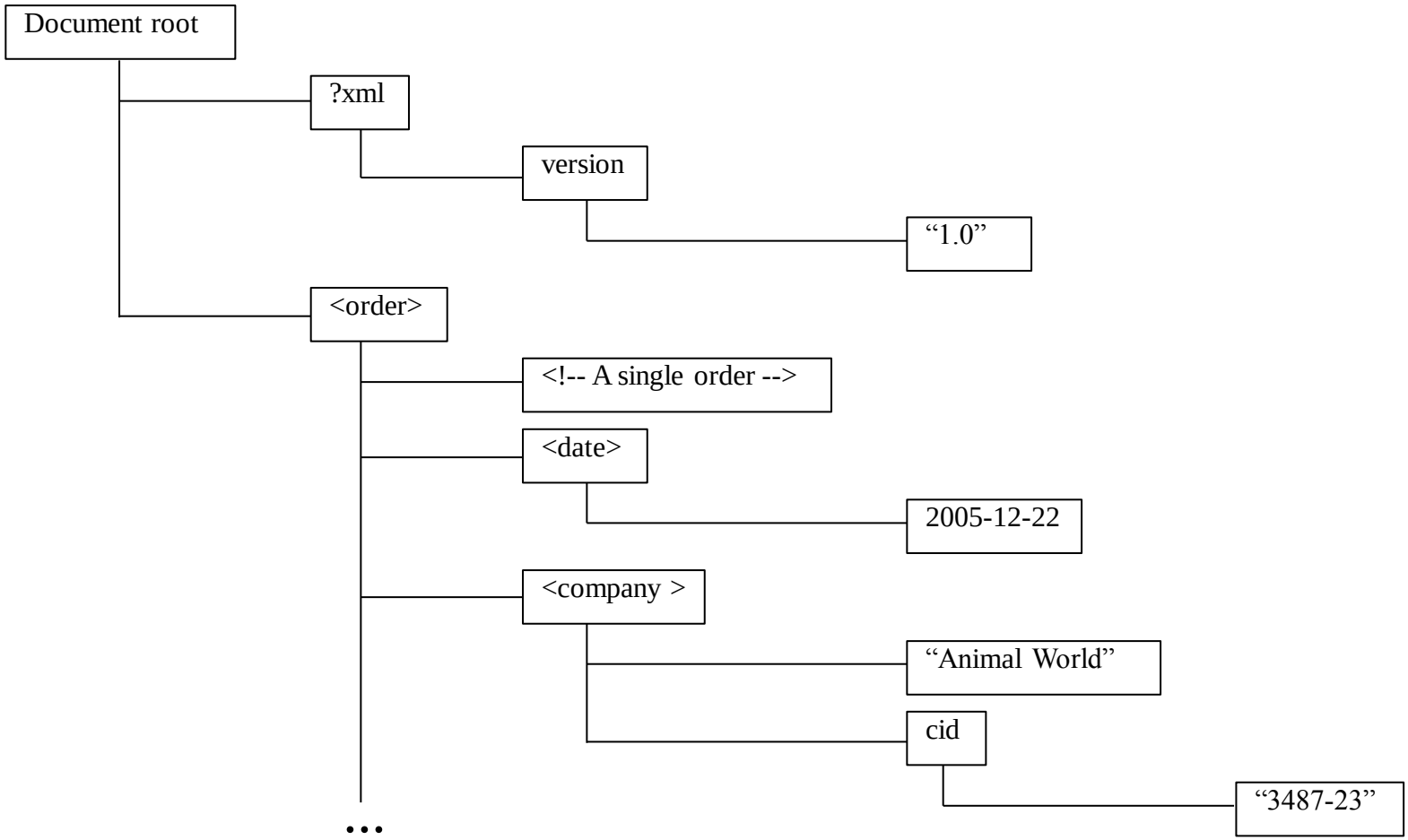
- Augmented *supplier.xml* document

```
<?xml version="1.0"?>
<order>
    <!-- A single order -->
    <date>2005-12-22</date>
    <company cid="3487-23">Animal World</company>
    <item>
        <code version="2.3">CC-10456</code>
        <description>Cat Chow</description>
        <qty>2</qty>
    </item>
    <for>
        <name>Frank Thomas</name>
        <street>10 Maple Street</street>
        <city country="US">Columbia, MD  22222</city>
    </for>
</order>
```

# Document Representation

- Parts of a document are called **nodes**
  - Elements
  - Attributes
  - Comments
  - Processing Instructions
- Nodes are just parts of a document tree
- **node-sets** are sets of nodes
  - If XPath specifies a location path for elements, several nodes may be returned
  - The set of these nodes is called, not surprisingly, a **node-set**

# Tree Representation of supplier.xml



# Document Root

- Matching the document root
  - /
  - <xsl:template match="/"> … </xsl:template>
- Document root serves as a reference for all other nodes
- To move to the *order* node
  - Absolute path
    - /order
    - <xsl:template match="/order">
  - Relative path
    - Assuming that the root node (/) is the current **context node**
    - order (without the absolute path from the root node)
    - <xsl:template match="order">

# Recursive Descent

- Descend the entire document looking for matches
- For example, look for all names in *customer.xml*
  - //name
  - The node *name* can be at any level
- Attributes are accessible with the @ symbol
  - To find the xml version
  - /?xml/@version
  - To find all versions in the document
  - //@version
    - Picks up /?xml/@version
    - Picks up /order/item/code/@version

# Recursive Descent (cont.)

- Recursive descent is very powerful.
- It is inefficient because entire document must be read.
- Use it sparingly!
- It can significantly slow your XSLT processor.

# Specific Location Paths

To access specific nodes:

- Match can be based on having sub-elements
- Match can be based on having sub-attributes
- Search for nodes with specific data values
- Use the [] notation
  - Filters the nodes selected
  - /order[date]
    - Matches all tags <order> that are direct children of the document root (/) and that also have a date element (<date>) child
  - order[date]
    - Matches all tags <order> that are direct children of current context and that also have a date element (<date>) child

# Specific Location Paths (cont.)

- company[@cid]
  - Matches all tags <company> that are direct children of the current context and that also have a cid attribute
- company[@cid = ‘3487-23’]
  - Matches all tags <company> that are direct children of the current context and that also have a cid attribute with value ‘3487-23’
  - Notice that single quotes are used.
  - This is because the XPath expression will be inside of double quotes.
    - <xsl:template match=“company[@cid=‘3487-23’]”>
  - This could be reversed - just need to keep it straight
    - <xsl:template match='company[@cid="3487-23"]'>

# Specific Location Paths (cont.)

- company/@cid
  - Matches all attributes *cid* that are attributes of elements <company> with respect to the current context
- order[company = ‘Animal World’]
  - Matches all tags <order> that are direct children of the current context and that also have a company element with a value ‘Animal World’
- company[. = ‘Animal World’]
  - Matches all tags <company> that are direct children of the current context and that also have a value ‘Animal World’

# Specific Location Paths (cont.)

- Location paths can be arbitrarily complex
- /order//code[@version = ‘2.3’]
  - Matches all tags <code> that are direct or indirect descendents of the <order> tag, which is a direct child of the document root, and that has an attribute *version* with a value of ‘2.3’.
- //order[company/@cid = ‘3487-23’]
  - Matches all tags <order> that are direct or indirect descendents of the document root and that have a child element <company> that has an attribute *cid* with a value of ‘3487-23’

# Reading Location Paths

- /order[date = ‘2005-12-22’]/item/code[@version = ‘2.3’]
- First, break it down into the path
  - /order/item/code
- Then, select the most specific elements
  - /order/item/code[@version = ‘2.3’]
- Finally, select only those paths with certain dates
  - /order[date = ‘2005-12-22’]/item/code[@version = ‘2.3’]
- With a little practice, it should become easy to read.

# XPath Functions

- Sometimes, pure path expressions are not powerful enough to retrieve what is desired
- Beyond parent/child and element/attribute relationships
- Can work with strings and numbers
- Can format values for output
- Syntax:
  - function(parameters)
  - function()

# Node Functions - name()

- Node functions work with nodes
- name()
  - Returns the name of the node
  - Can be called with a parameter: name(.)
    - Name of context node
    - Same as name()
  - To return a sorted list of names of nodes that are children of <order>:

```
<xsl:template match="/order">
    <xsl:for-each select="*">
        <p><xsl:sort select="name()"/></p>
    </xsl:for-each>
</xsl:template>
```

# Node Functions - node()

- node() returns the node itself
- Same as .
- Not really necessary

# Node Functions - processing-instruction()

- Returns processing instructions
- This:

```
<xsl:template match="processing-instruction()">
    <xsl:value-of select="."/>
</xsl:template>
```

- Returns all processing instructions
- This:

```
<xsl:template match="processing-instruction('xml')">
    <xsl:value-of select="."/>
</xsl:template>
```

- Returns ?xml processing instructions: <?xml version="1.0"?>
  - Not all parser return the <?xml …?> element

# Node Functions - comment()

- Matches comments in the XML document
- This:

```
<xsl:template match="//comment()">
    <xsl:value-of select="."/>
</xsl:template>
```

- Returns all the comments in the XML document:

```
<!-- A single order -->
```

- Parsers are not required to return comments so your XSLT may never get a comment to process.

# Node Functions - text()

- Text lets you select the PCDATA content of a node
- When you match a node, you generally get back the node and all of its descendents.
- Example XML fragment:

```
<outerlayer> This is text for the outer layer
    <middlelayer> This is text for the middle layer
        <innerlayer> This is text for the inner layer
          </innerlayer>
    </middlelayer>
</outerlayer>
```

# Node Functions - text() - (cont.)

- Applying this:

```
<xsl:output method="text“/>
<xsl:template match=“outerlayer”>
    <xsl:value-of select=“.”/>
</xsl:template >
```

- Yields all outerlayer, middlelayer, and innerlayer text

```
This is text for the outer layer
    This is text for the middle layer
        This is text for the inner layer
```

# Node Functions - text() - (cont.)

- To get just the outerlayer text, apply this:

```
<xsl:output method="text“/>
<xsl:template match=“outerlayer”>
      <xsl:value-of select=“text()”/>
</xsl:template >
```

- This yields only:

```
This is text for the outer layer
```

# Positional Functions - position()

- When you want the position of a node within a node-set
- Using customer.xml, get the position of William Tell

```
<xsl:template match="//name[. = 'Tell, William']">
    <xsl:value-of select="position()"/>
</xsl:template >
<xsl:template match="//name"/>
```

- Yields:

  2

- Because:

  Node set is: text - <name> - text - <name> - text

  "Tell, William" is the first element of <bad> customers

# Positional Functions - position() - (cont.)

- When you want to search for a node at a specific position
- To look for the second name, apply the template:

```
<xsl:template match="//name[position() = 2]">
    <xsl:value-of select="."/>
</xsl:template >
<xsl:template match="//name"/>
```

- Which yields:

```
Li, Sue
Carr, Sam
```

- Shorthand notation

```
<xsl:template match= "//name[2]" >
    <xsl:value-of select="."/>
</xsl:template >
```

# Positional Functions - last()

- Use last() to find the last position in a node set.

```
<xsl:template match="//name[position() = last()]">
    <xsl:value-of select="."/>
</xsl:template >
<xsl:template match="//name"/>
```

- Which yields:

```
Carnot, John
Carr, Sam
```

# Positional Functions - last()-1

- Use last() to find the next to last position in a node set.

```
<xsl:template match="//name[position() = last()-1]">
    <xsl:value-of select="."/>
</xsl:template >
<xsl:template match="//name"/>
```

- Which yields:

Li, Sue
Tell, William

# Positional Functions - count()

- Use count() to find the number of elements in a node set.

```
<xsl:template match="/">
    <xsl:value-of select="count(//name)"/>
</xsl:template >
```

- Which yields:

```
5
```

# Numeric Functions - number()

- The number function converts PCDATA from a character string to a number
- To get a numeric count of the number of items ordered:

```
<xsl:template match="/">
    <xsl:value-of select="number(/order/item/qty)"/>
</xsl:template >
```

- This yields the number:

```
2
```

# Numeric Functions - sum()

- Produces the sum of a node-set
- Given an XML document:

```
<employee id="111-11-1111">
    <hours day="1">7</hours>
    <hours day="2">10</hours>
    <hours day="3">7</hours>
    <hours day="4">8</hours>
    <hours day="5">8</hours>
</employee>
```

- Apply to get the total hours worked for the week (40):

```
<xsl:template match="/">
    <xsl:value-of select="sum(employee/hours)"/>
</xsl:template>
```

# Boolean Functions - boolean()

- Used for boolean math
- boolean()
  - Evaluates to either *true* or *false*
  - If value is number, it is *false* if value is 0 and *true* otherwise
  - If value is string, it is *true* if the length > 0 and *false* otherwise
  - If value is node-set, it is *false* if empty and *true* otherwise
- Applied to customers.xml

```
<xsl:template match="/">
    <xsl:if test="boolean(//name)">
        <p>We have some customers :-)</p>
    </xsl:if>
</xsl:template>
```

# Boolean Functions - not()

- Test the opposite condition of boolean statement
- Applied to customers.xml

```
<xsl:template match="/">
    <xsl:if test="not(boolean(//name))">
        <p>We do not have any customers :-(</p>
    </xsl:if>
</xsl:template>
```

# Boolean Functions - true() and false()

- true() and false() always evaluate to *true* and *false*, respectively.
- Useful for debugging

```
<xsl:template match="/">
    <xsl:if test="true()">
        <p>Always print this line.</p>
    </xsl:if>
</xsl:template>
```

# String Functions - string()

- Converts any value to a string
- Useful for converting from string to numbers, performing calculations, and converting back to string.

```
<xsl:template match="/">
    <p>
    The number of hours worked this week is:
    <xsl:value-of select="string(sum(employee/hours))"/>
    </p>
</xsl:template>
```

# String Functions - string-length()

- Finds the length of the string
- To length of each customer name

```
<xsl:template match="//name">
    <xsl:value-of select="string-length()"/>
</xsl:template >
```

# String Functions - contains()

- Determines if a substring is contained in a string
- Prints all customers with the string ‘William’ in their names

```
<xsl:template match=“//name”>
    <xsl:if test=“contains(., ‘William’)”>
      <xsl:value-of select=“.”/>
    </xsl:if>
</xsl:template >
```

- This yields:

  Tell, William

- contains() is case-sensitive, like all string functions.

# String Functions - starts-with()

- Determines if a string starts with a certain string.
- Prints all customers with last names that start with ‘L’

```
<xsl:template match=“//name”>
    <xsl:if test=“starts-with(text(), ‘L’)”>
      <xsl:value-of select=“.”/>
    </xsl:if>
</xsl:template >
```

- Yields:

  Li, Sue

# String Functions - substring()

- Finds a substring of a string
- Applying to customer.xml:

```
<xsl:template match="//name[1]">
    <xsl:value-of select="substring(., 8)"/>
</xsl:template >
<xsl:template match="//name"/>
```

- Yields:

```
Fred
illiam
```

- This works as well:

```
<xsl:template match="//name[1]">
    <xsl:value-of select="substring(.,  8, 4)"/>
</xsl:template >
<xsl:template match="//name"/>
```

# String Functions - substring-after()

- Finds substring after a string
- Applying to customer.xml:

```
<xsl:template match="//name[1]">
        <xsl:value-of select="substring-after(. , ',')"/>
</xsl:template >
<xsl:template match="//name"/>
```

- Yields:

```
Fred
William
```

# String Functions - substring-before()

- Finds substring before a string
- Applying to customer.xml:

```
<xsl:template match="//name[1]">
    <xsl:value-of select="substring-before(. , ',')"/>
</xsl:template >
<xsl:template match="//name">
```

- Yields:

```
Jones
Tell
```

# String Functions - concat()

- Puts multiple strings together
- Applying to customer.xml:

```
<xsl:template match="//name[1]">
    <xsl:value-of select="concat(substring-after(. , ','), ' ',
      substring-before(. , ','))"/>
</xsl:template >
<xsl:template match="//name">
```

- Yields:

Fred Jones

William Tell

# String Functions - translate()

- Translates one character set to another
- translate(string-to-translate,  match-string, translate-string)
- Applying to customer.xml:

```
<xsl:template match="//name">
    <xsl:value-of select="translate(. , 'abcdefghijklmnopqrstuvwxyz',
        'bcdefghijklmnopqrstuvwxyza')"/>
</xsl:template >
```

- Yields

Jpoft, Fsfe

Lf, Svf

Cbsopu, Jpio

Temm, Wjmmjbn

Cbss, Sbn

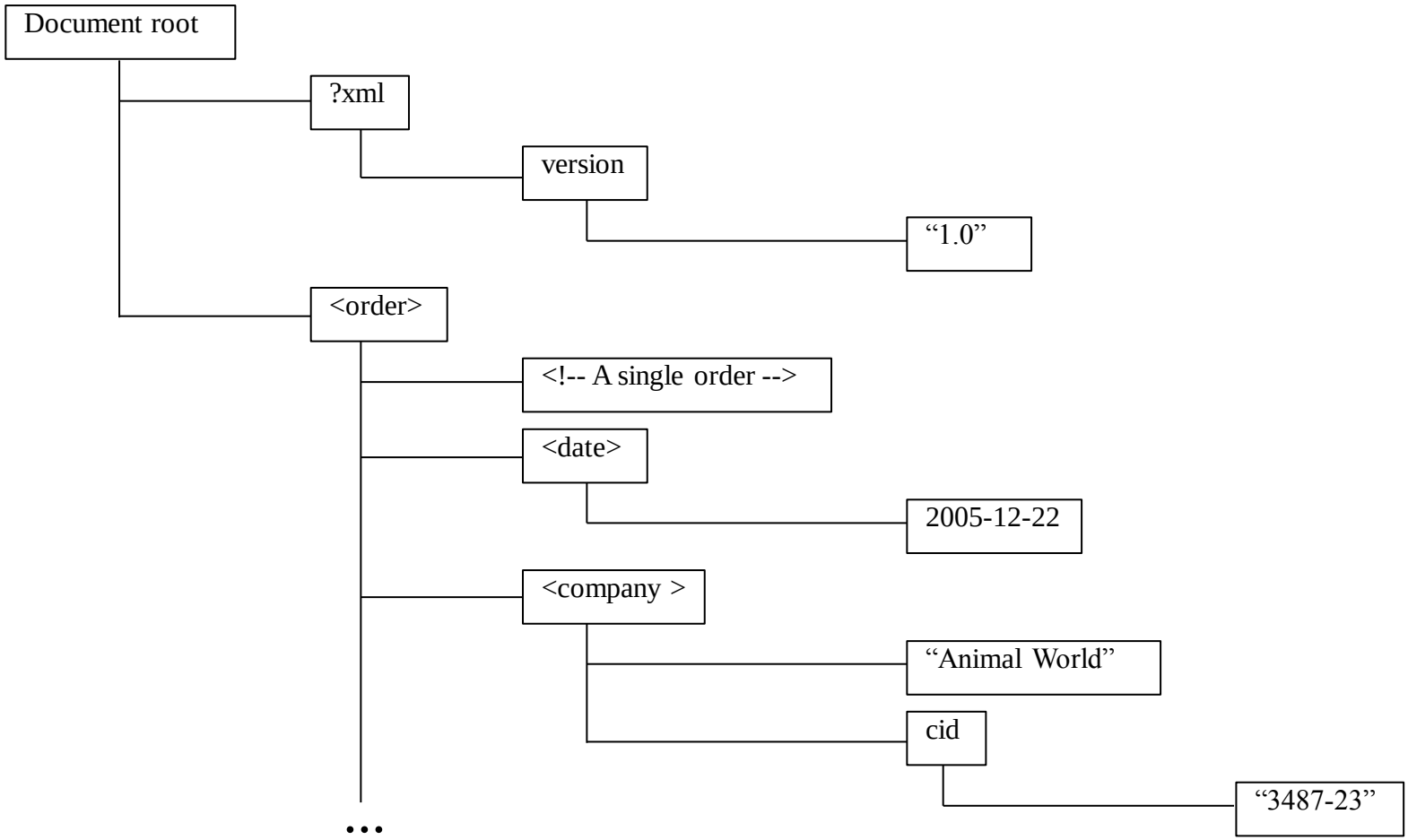
# String Functions - translate() - (cont.)

- Remember that translate is case-sensitive
- If the second string is longer than the third string, the extra character is stripped out of the result
- translate(‘abcd’, ‘bcd’, ‘BC’)
- Yields: aBC

# Axis Names

- Axis names are a way to specify any node of a document with respect to any other node (or itself)
- Up until now, we dealt with parents and children.
- Now, we can use axis names to deal with other node relationships.

# Remember supplier.xml?



# Some Axis Commands

- **self**
  - The self axis refers to the node at the current context
  - “.” is equal to “self::node()”
  - <xsl:for-each select=“.”> =
    <xsl:for-each select=“self::node()”>
  - If current context is the <order> tag, then both produce <order>
- **child**
  - Child is used most often so it is the default
  - “date” is equal to “child::date”
  - <xsl:for-each select=“date”> =
    <xsl:for-each select=“child::date”>
  - If our current context is the <order> tag, then both produce <date>

# More on Children

- **child**
  - "child::*" produces all children elements of the current context
  - <xsl:for-each select="*"> =

    <xsl:for-each select="child::*">
  - If current node is <order>, this produces <date>, <company>, etc.

  - "child::@*" produces all children attributes of the current context
  - <xsl:for-each select="@*"> =

    <xsl:for-each select="child::@*"> =

    <xsl:for-each select="attribute::*">
  - If current node is <company>, this produces "cid"

# Descendants

- **descendent**
  - “descendent::date” produces all elements <date> which are children, childrens’ children, etc. elements of the current context
  - <xsl:for-each select=“descendent::date”>
  - If current node is <order> or the document root (/), this produces <date>

- **descendent-or-self**
  - “descendent-or-self::date” produces all elements <date> which are the same, children, childrens’ children, etc. elements of the current context
  - <xsl:for-each select=“descendent-or-self::date”> =

    <xsl:for-each select=“.//date”>
  - If current node is <date>, <order>, or the document root (/), this produces <date>

# Ancestors

- **parent**
  - “parent::order” produces the element <order> which is a parent of the current context
  - <xsl:for-each select=“parent::order”> =

    <xsl:for-each select=“../order”>
  - If current node is <date>, this produces <order>
  - <xsl:for-each select=“date/parent::order”> =

    <xsl:for-each select=“date/../order”>

# Ancestors (cont.)

- **ancestor**
  - “ancestor::order” produces the element <order> which is an ancestor of the current node
  - <xsl:for-each select=“ancestor::order”>
  - If current node is <date>, <company>, “cid”, etc., this produces <order>
- **ancestor-or-self**
  - “ancestor-or-self::order” produces the element <order> which is an ancestor of the current node or the current node itself
  - <xsl:for-each select=“ancestor-or-self::order”>
  - If current node is <date>, <company>, “cid”, etc., this produces <order>

# Siblings

- **following-sibling**
  - “following-sibling::company” produces the next element <company> which is the sibling following the current node
  - <xsl:for-each select=“/order/date/following-sibling::company”>
  - This produces <company>, etc.
  - Does not work for attributes
- **preceding-sibling**
  - “preceding-sibling::date” produces the previous element <date> which is the sibling preceding the current node
  - <xsl:for-each select=“/order/date/company/preceding-sibling::date”>
  - This produces the previous element: <date>
  - Does not work for attributes

# Siblings (cont.)

- **following**
  - “following::*” produces the next elements (including descendents) which follow the current node
  - <xsl:for-each select=“/following::*”> produces all the next elements of the root node: <order>, <date>, <company>, etc.
- **preceding**
  - “preceding::*” produces the preceding elements of the current node
  - <xsl:for-each select=“/order/date/company/preceding::*”>
  - This produces all the previous elements: <date>, <order>, and root

# Τέλος Ενότητας