Part 5: Accessing TEI texts. Untangling the XML markup and extracting what you need, with particular reference to the TEI XSL stylesheet family and its localization features.

March 8, 2008

Contents

1 XPath

XPath is the basis of most other XML querying and transformation languages. It is just a way of locating nodes in an XML document

1.1 Accessing your TEI document

So you've created some TEI XML documents, what now?

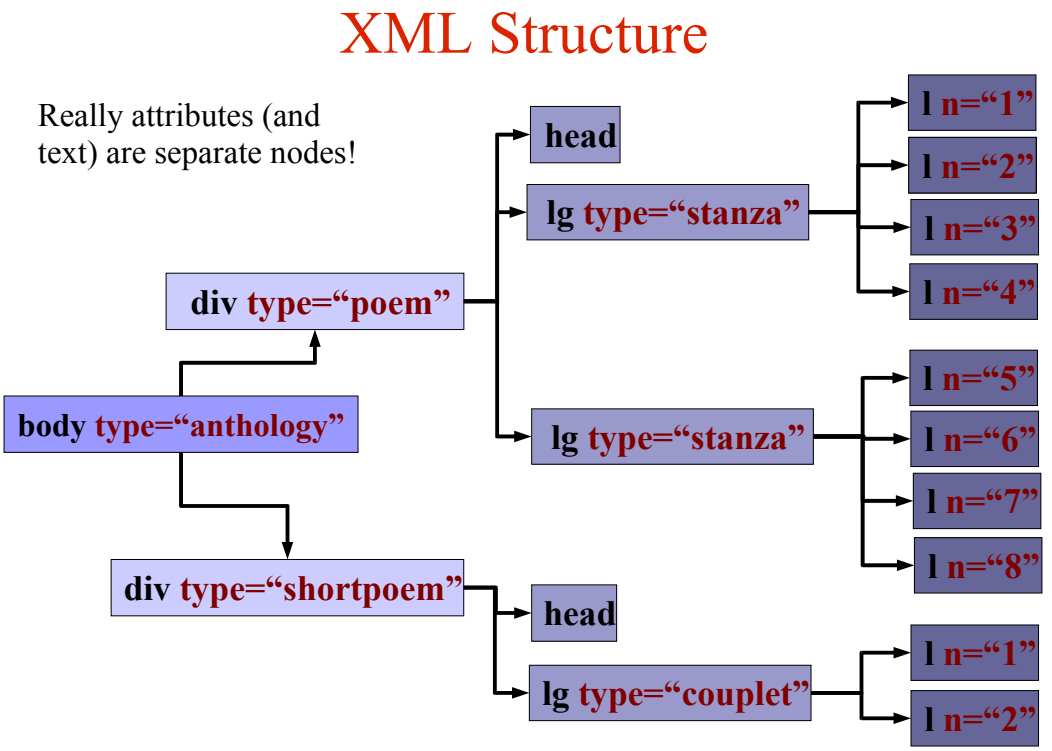
- XPath
- XML Query (XQuery)
- XSLT Tranformation to another format (HTML, PDF, RTF, CSV, etc.)
- Custom Applications (Xaira, TEIPubisher, Philologic etc.)

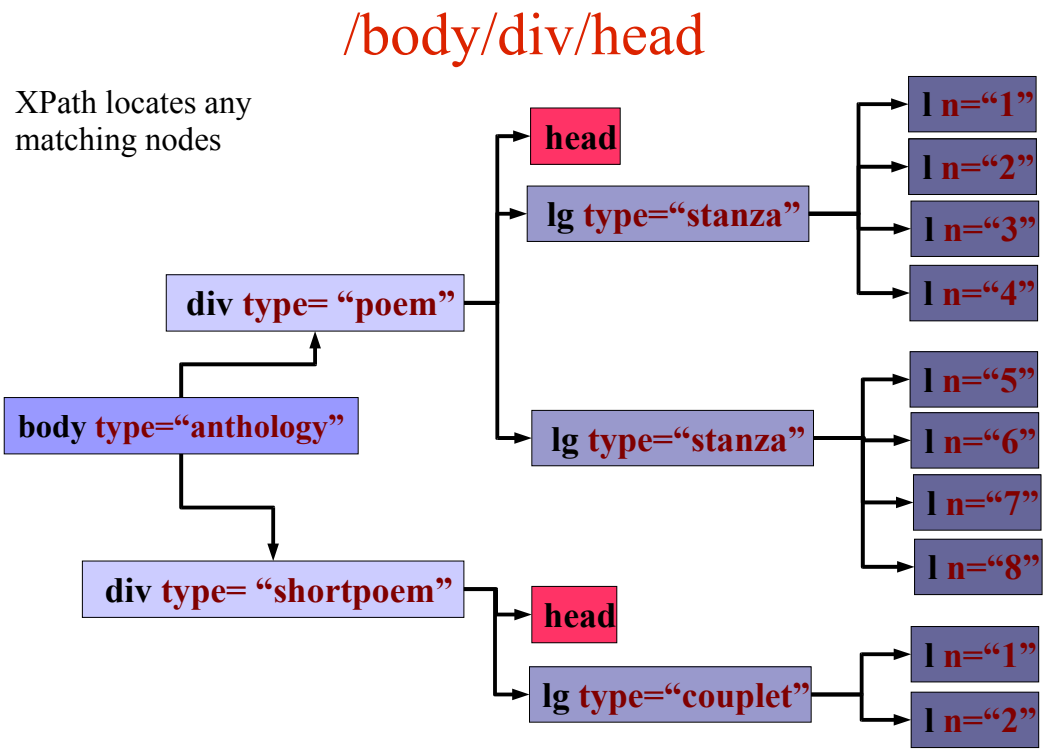
1.2 What is XPath?

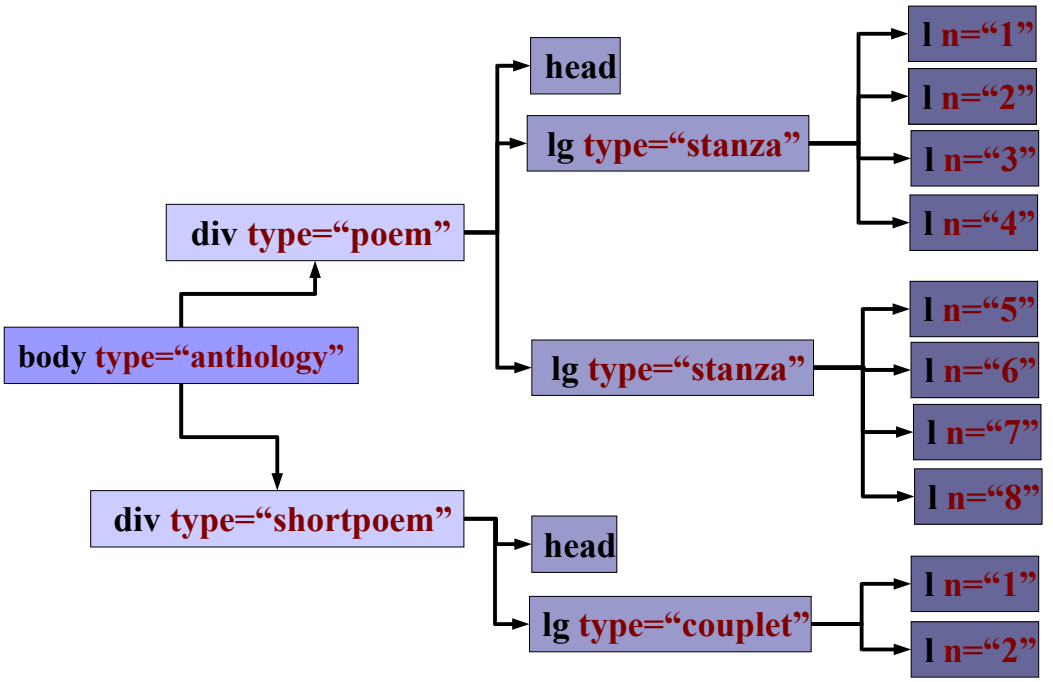
- It is a syntax for accessing parts of an XML document
- It uses a path structure to define XML elements
- It has a library of standard functions
- It is a W3C Standard
- It is one of the main components of XQuery and XSLT

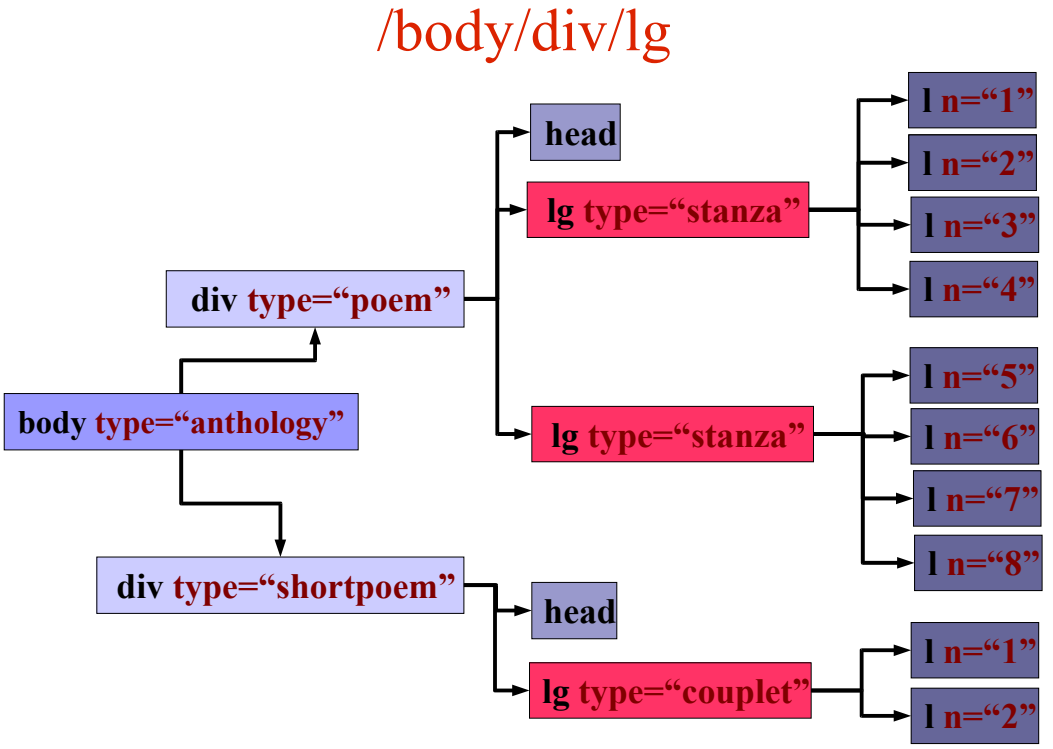
1.3 Example text

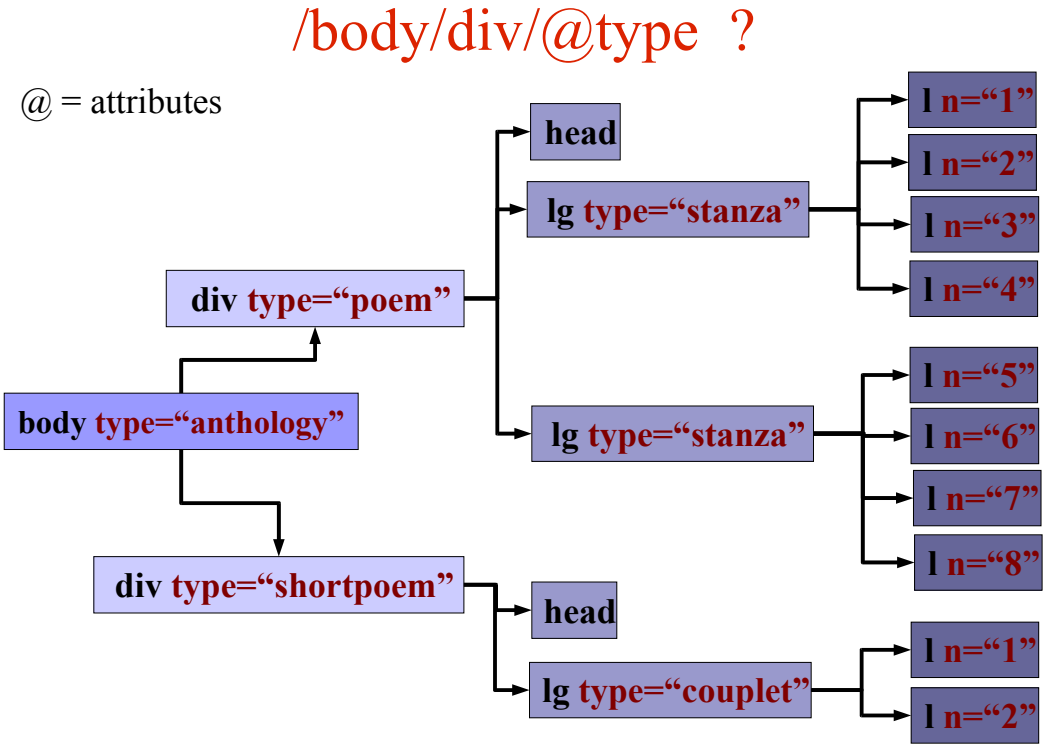
```
<body type="anthology"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:tei="http://www.tei-c.org/ns/1.0">
 <div type="poem">
  <head>The SICK ROSE </head>
  <lg type="stanza">
   <l n="1">O Rose thou art sick.</l>
   <l n="2">The invisible worm,</l>
   <l n="3">That flies in the night </l>
   <l n="4">In the howling storm:</l>
  </lg>
  <lg type="stanza">
   <l n="5">Has found out thy bed </l>
   <l n="6">Of crimson joy:</l>
   <l n="7">And his dark secret love </l>
   <l n="8">Does thy life destroy.</l>
  </lg>
 </div>
</body>
```

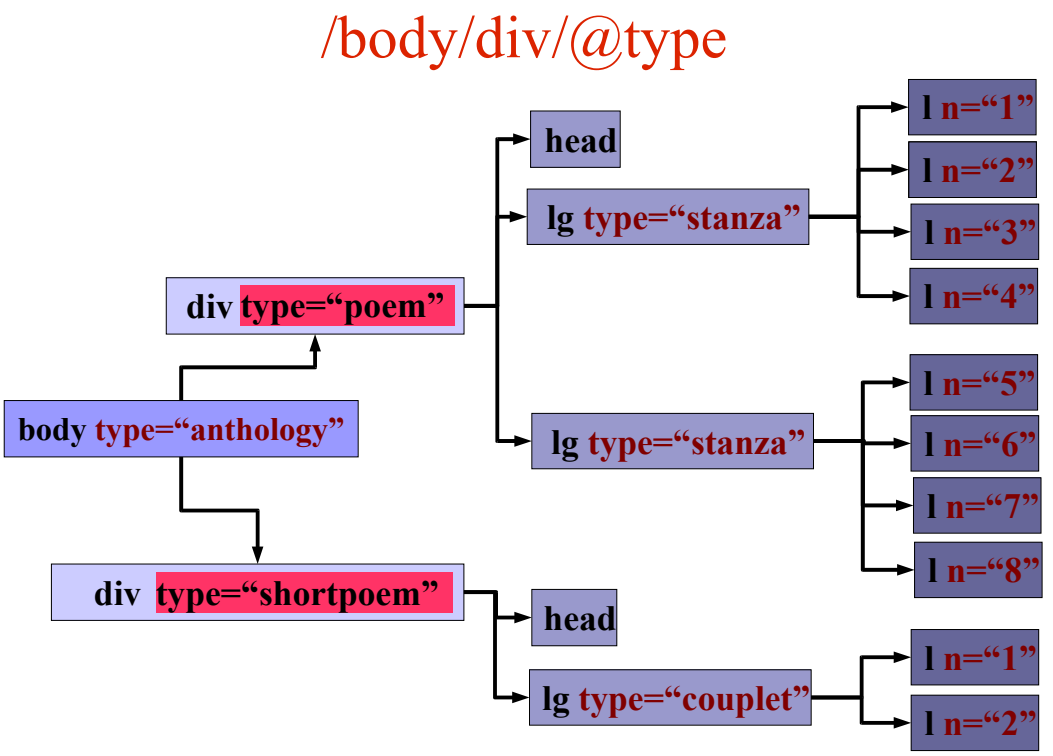


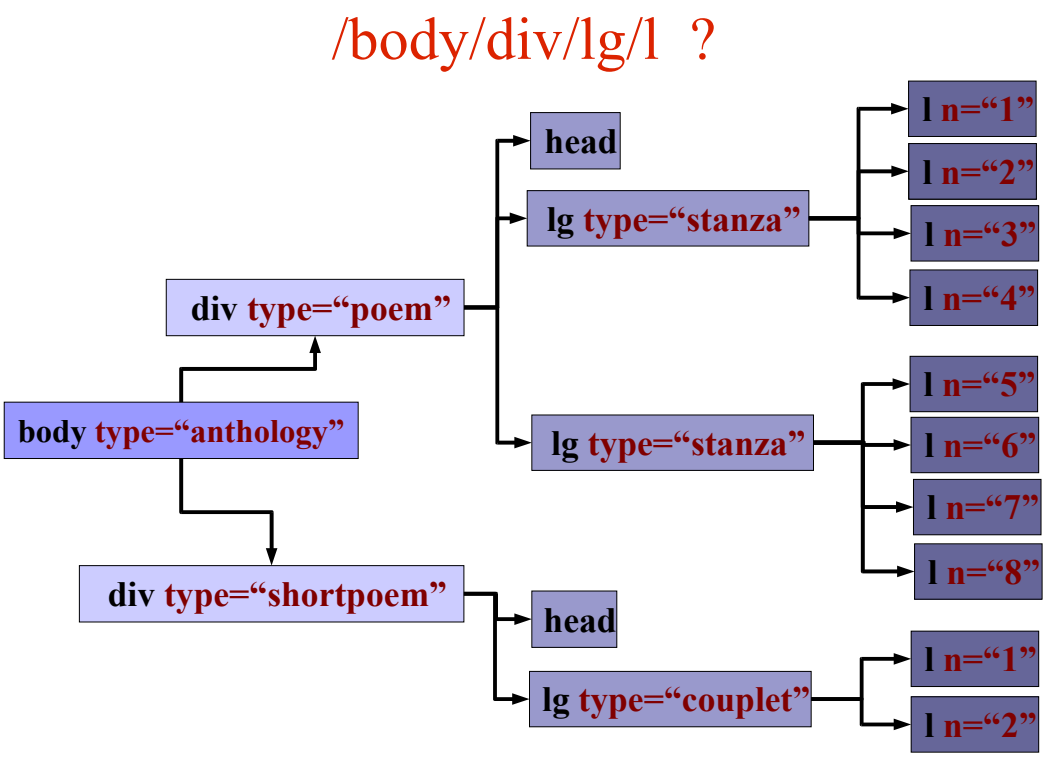

/body/div/head
XPath locates any matching nodes
body type="anthology"
div type= "poem"
div type= "shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"


/body/div/lg ?
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"

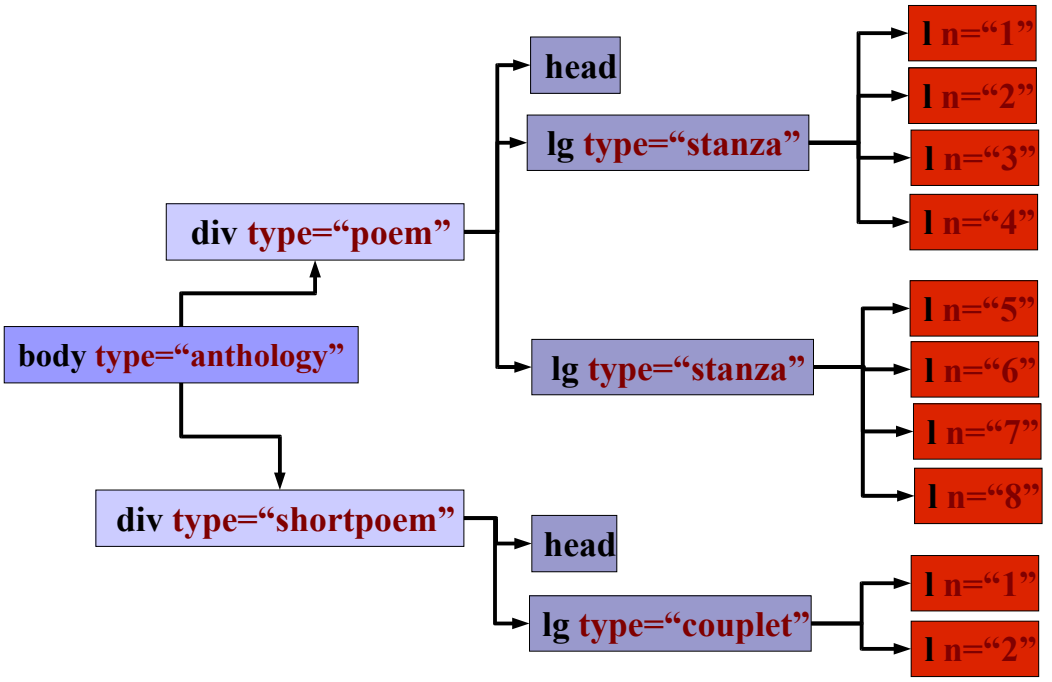

/body/div/lg
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"


/body/div/@type ?
@ = attributes
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"


/body/div/@type
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"

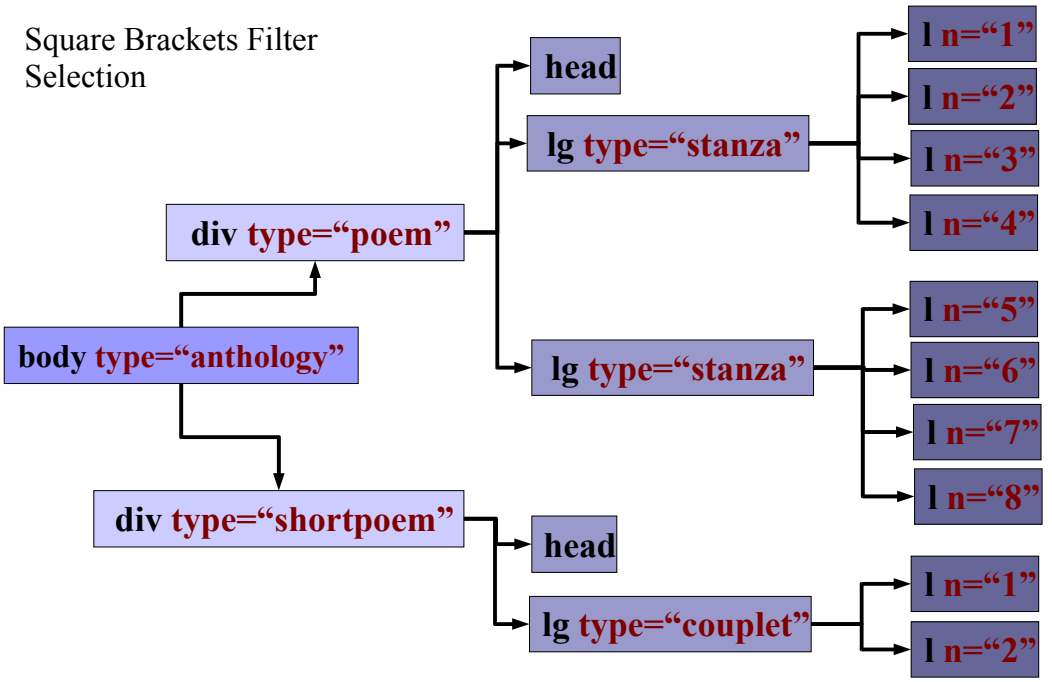

/body/div/lg/l ?
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"

/body/div/lg/l

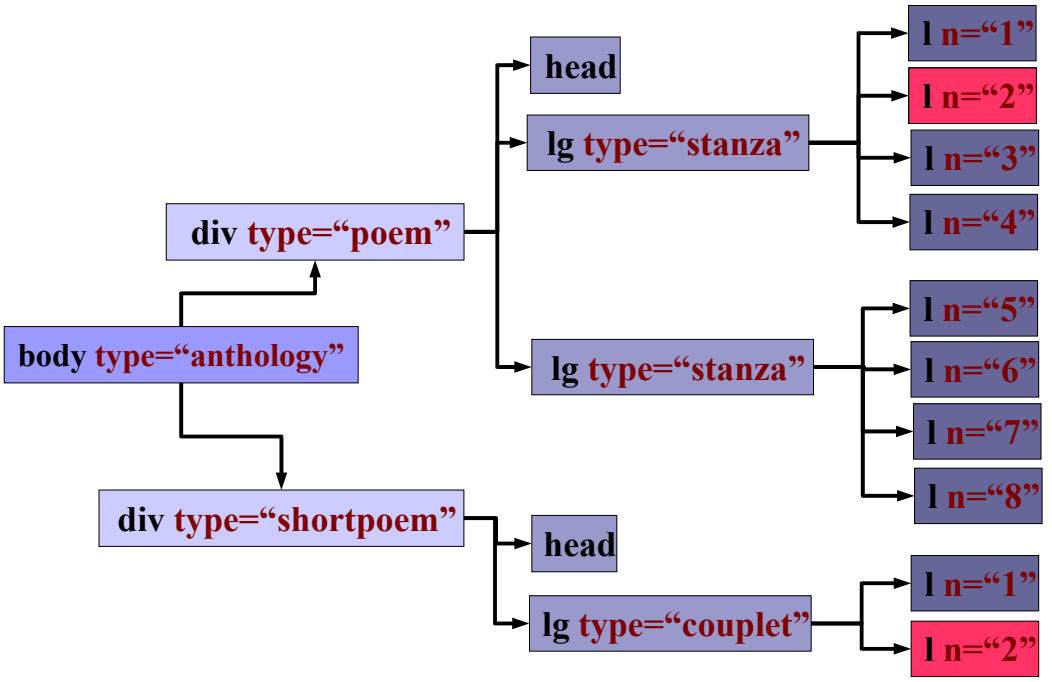


/body/div/lg/l[@n="2"] ?

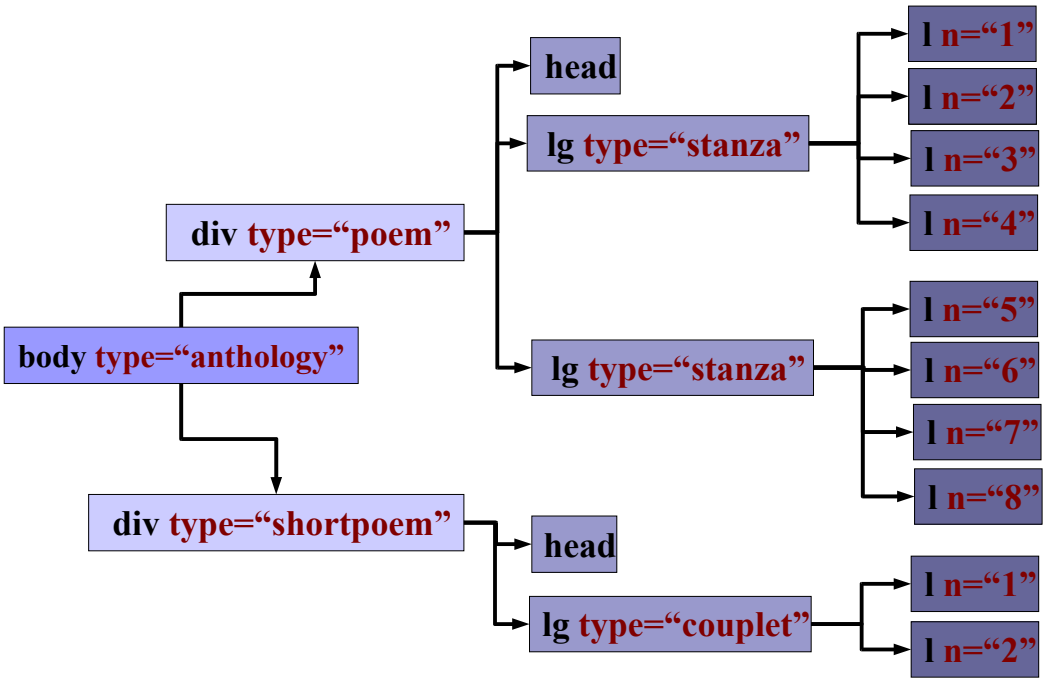
Square Brackets Filter Selection



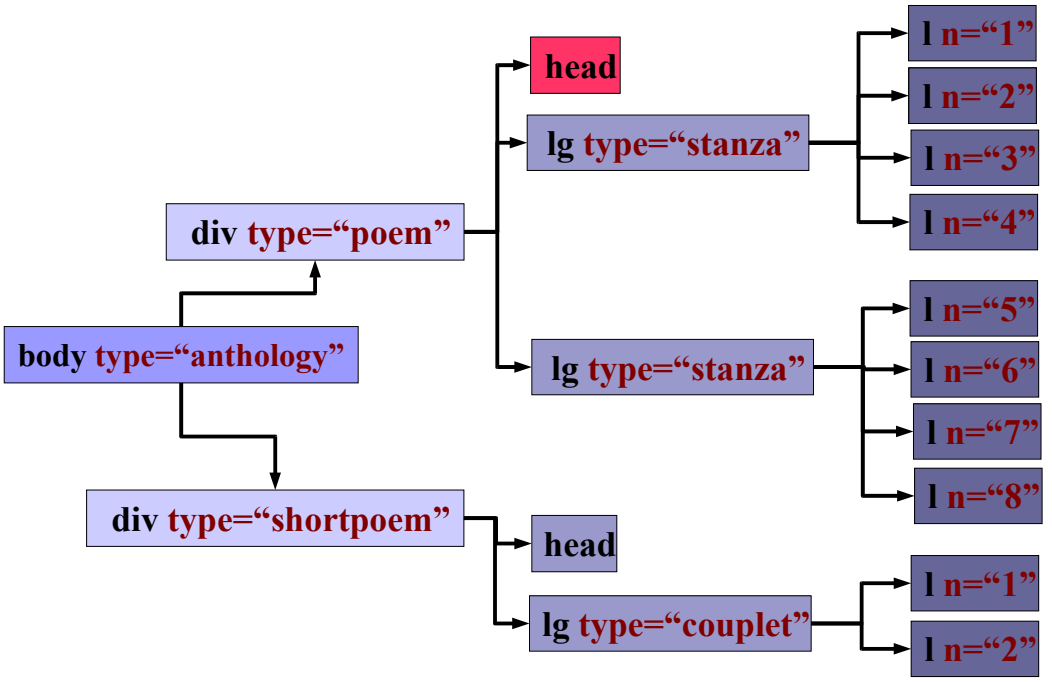
/body/div/lg/l[@n="2"]



/body/div[@type="poem"]/head ?

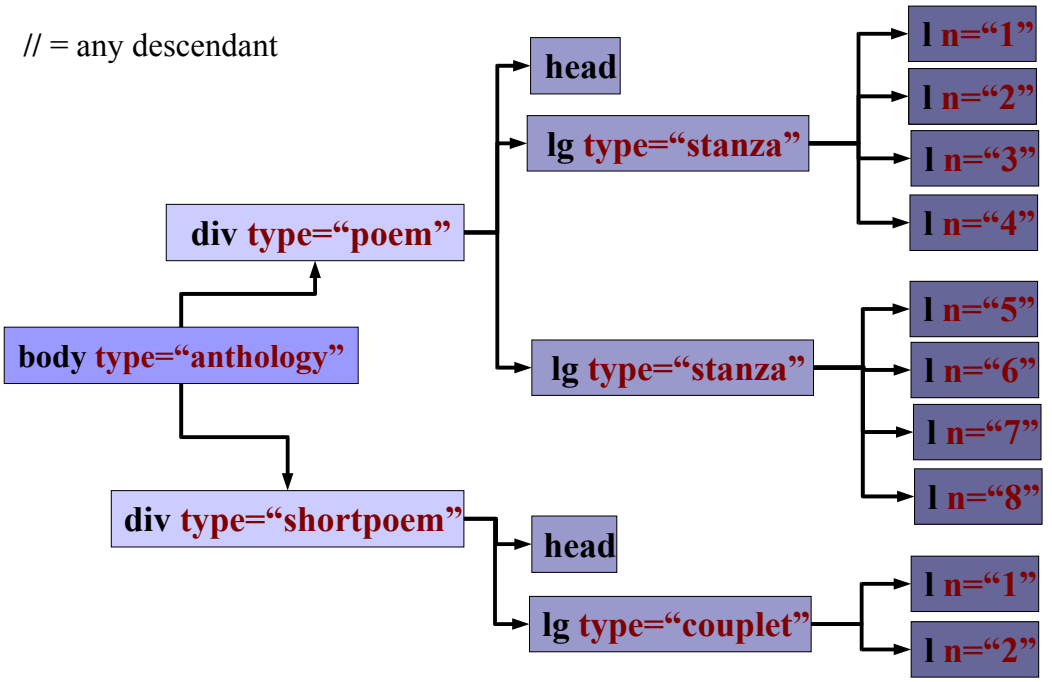


/body/div[@type="poem"]/head

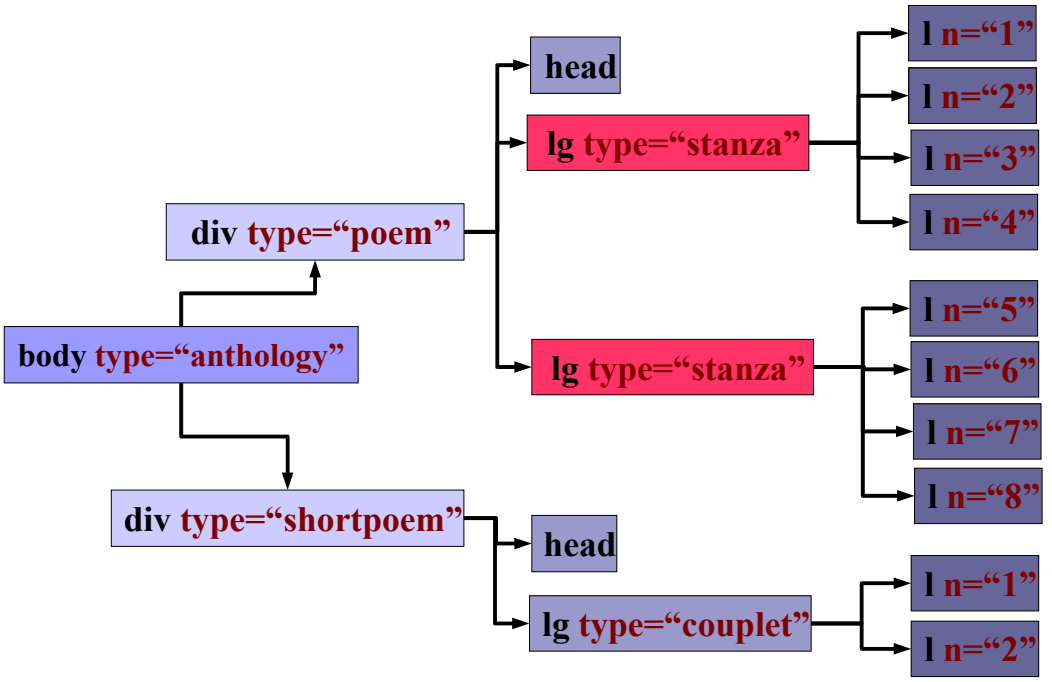


//lg[@type="stanza"] ?

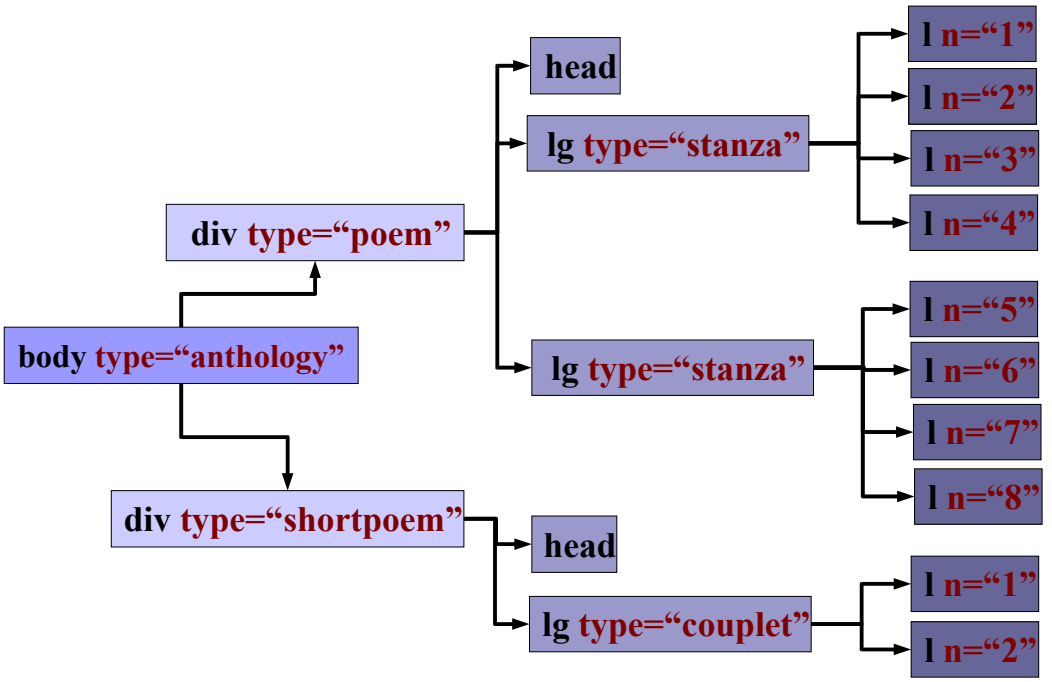
// = any descendant



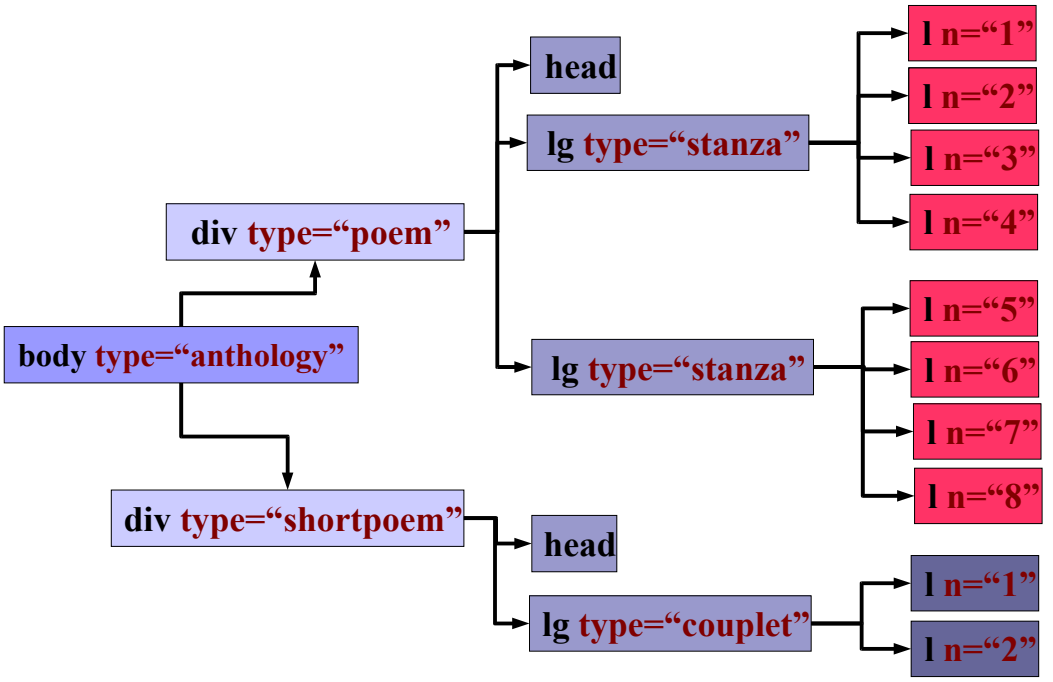
//lg[@type="stanza"]



//div[@type="poem"]//l ?

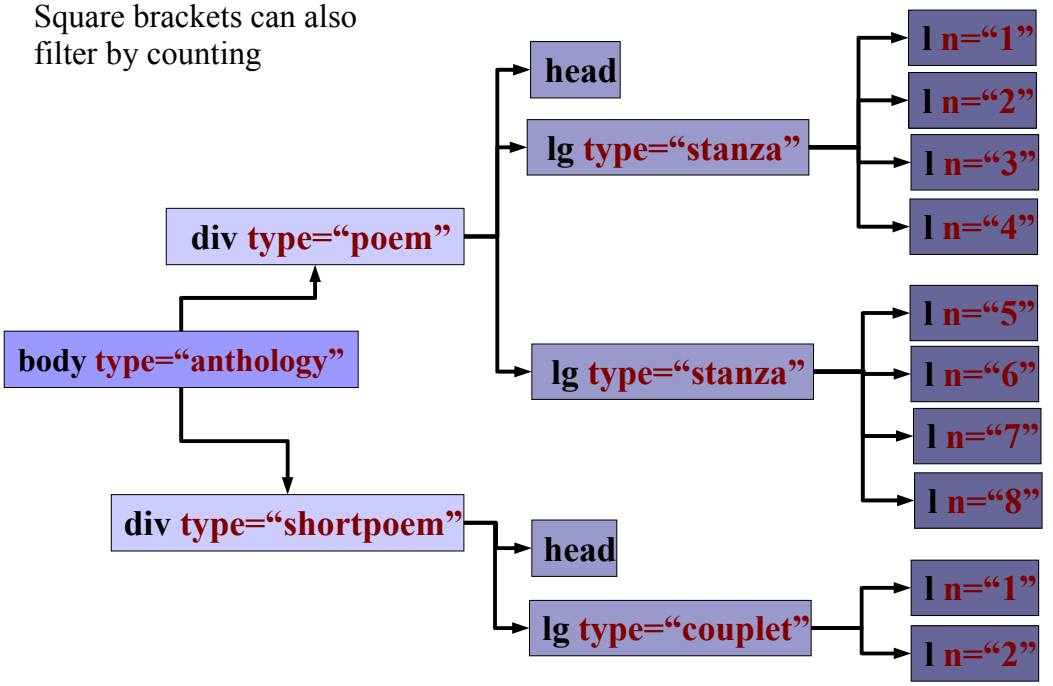


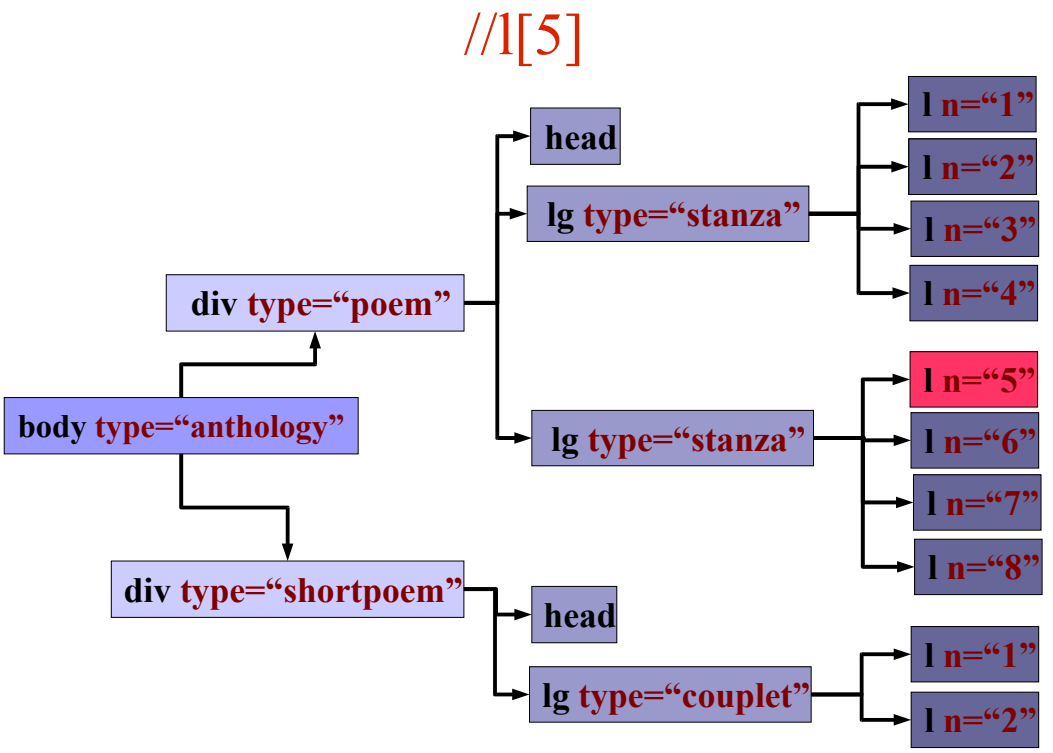
//div[@type="poem"]//l

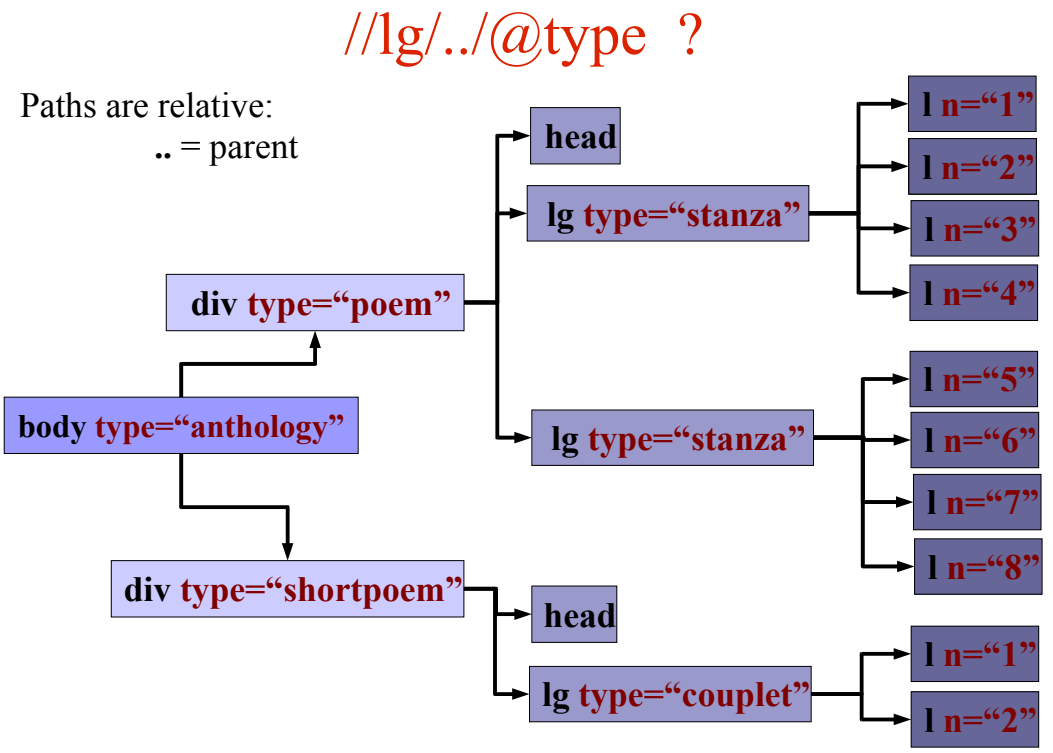


//l[5] ?

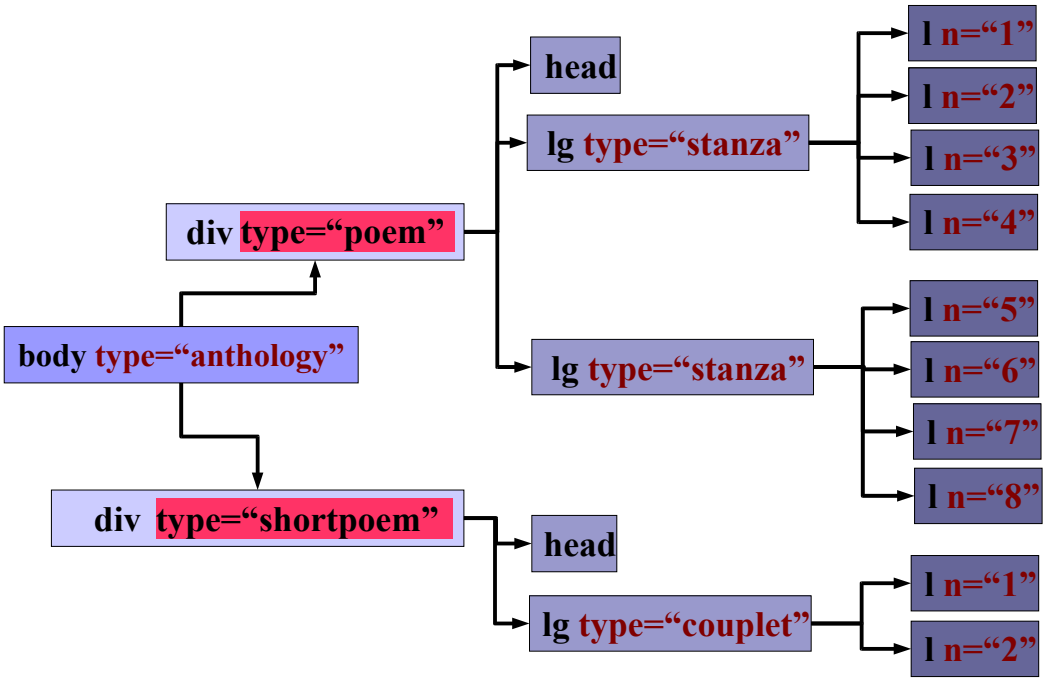
Square brackets can also filter by counting




//l[5]
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"

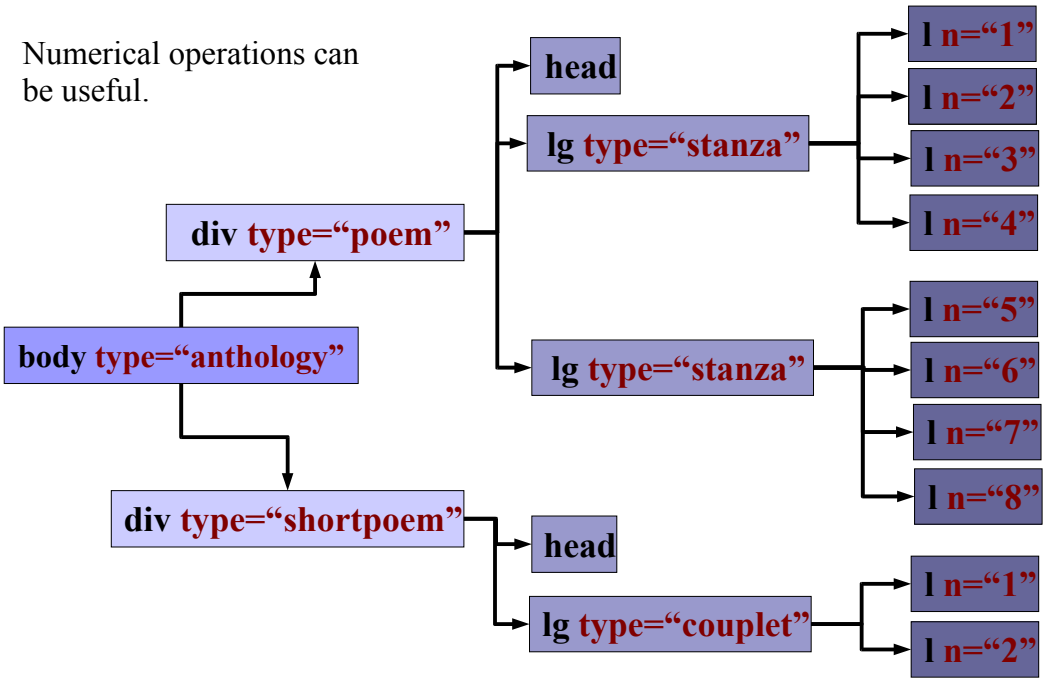

//lg/../@type ?
Paths are relative:
.. = parent
body type="anthology"
div type="poem"
div type="shortpoem"
head
lg type="stanza"
lg type="stanza"
head
lg type="couplet"
l n="1"
l n="2"
l n="3"
l n="4"
l n="5"
l n="6"
l n="7"
l n="8"
l n="1"
l n="2"

//lg/../@type

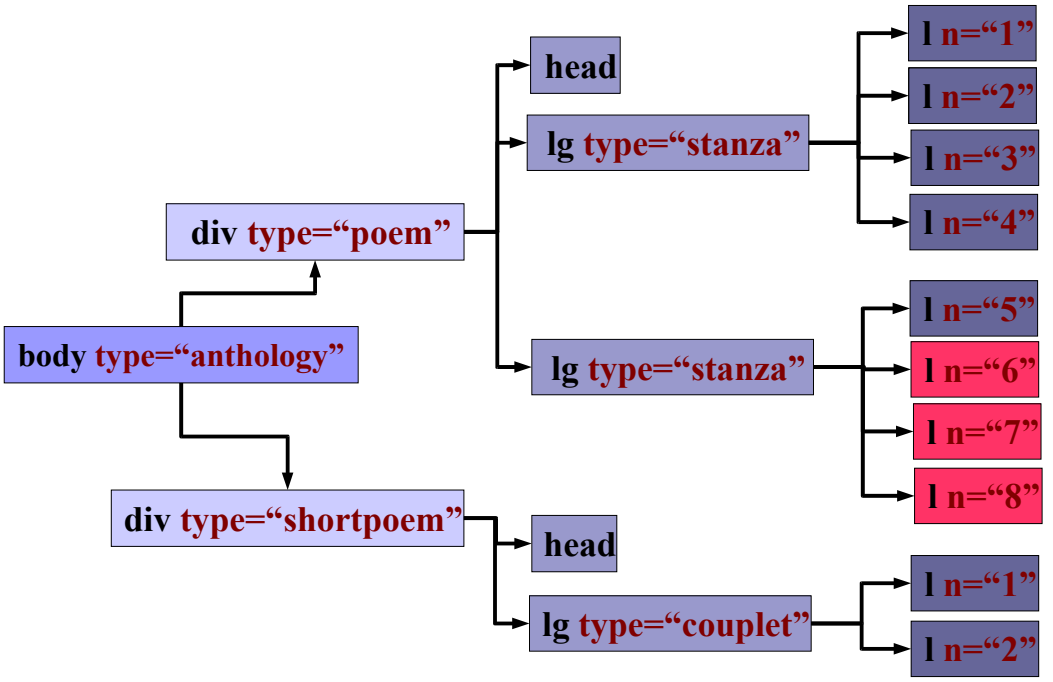


//l[@n > 5] ?

Numerical operations can be useful.

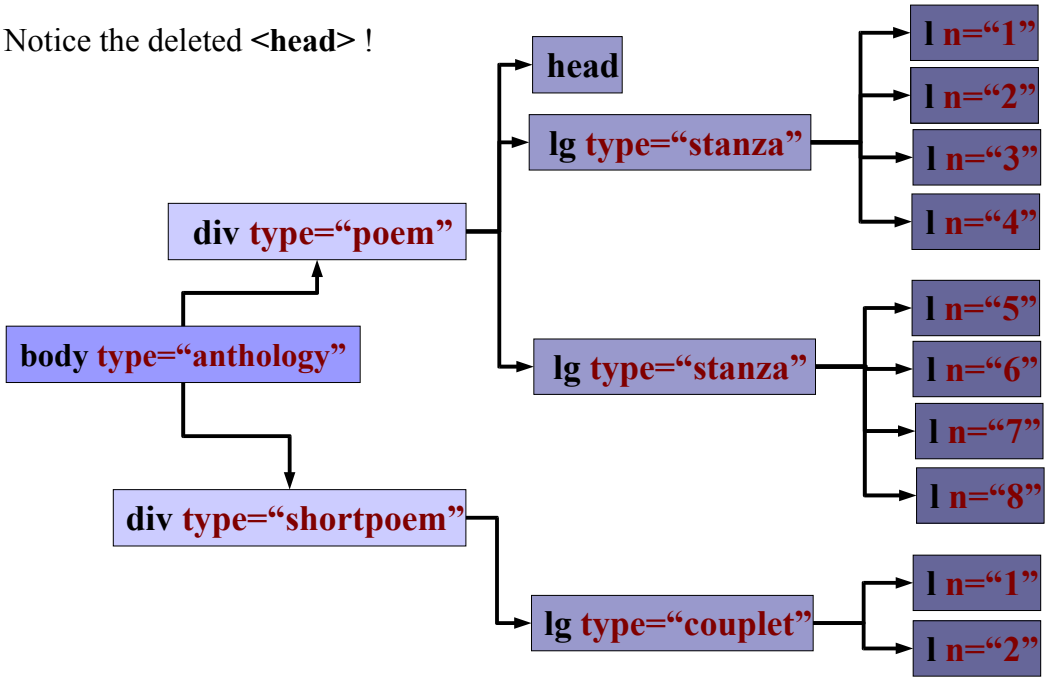


//l[@n > 5]

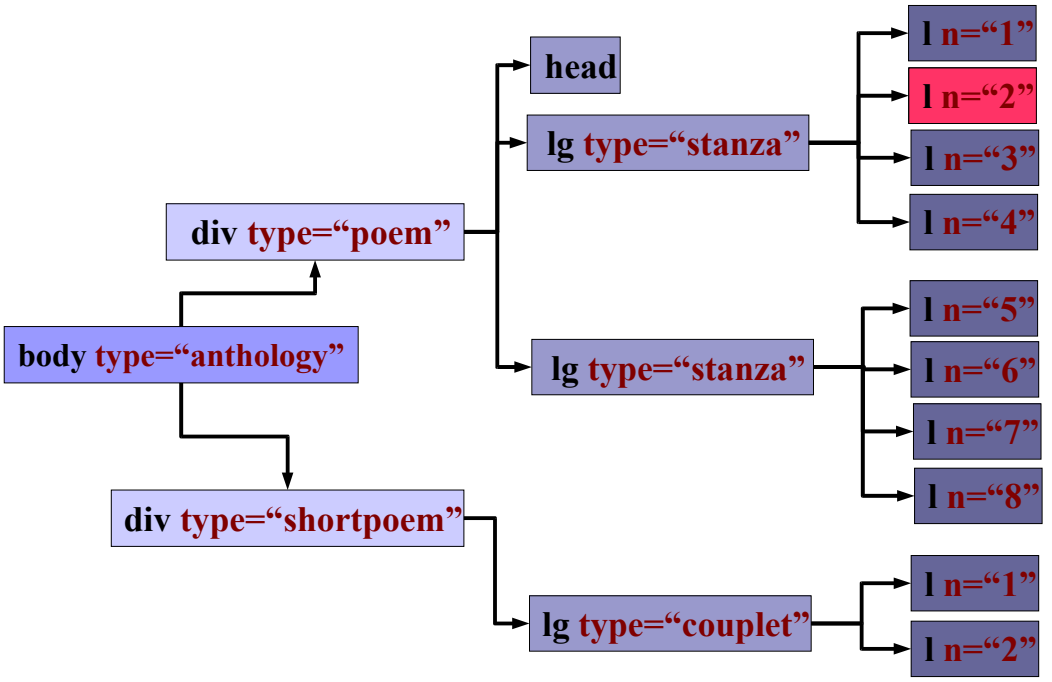


//div[head]/lg/l[@n="2"] ?

Notice the deleted **<head>** !

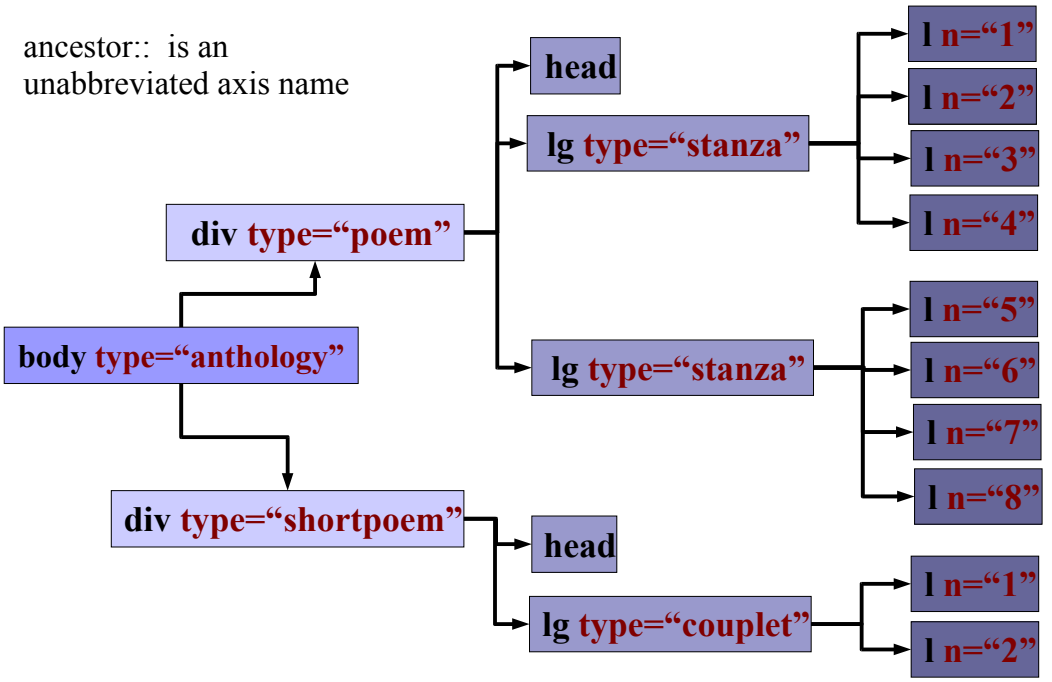


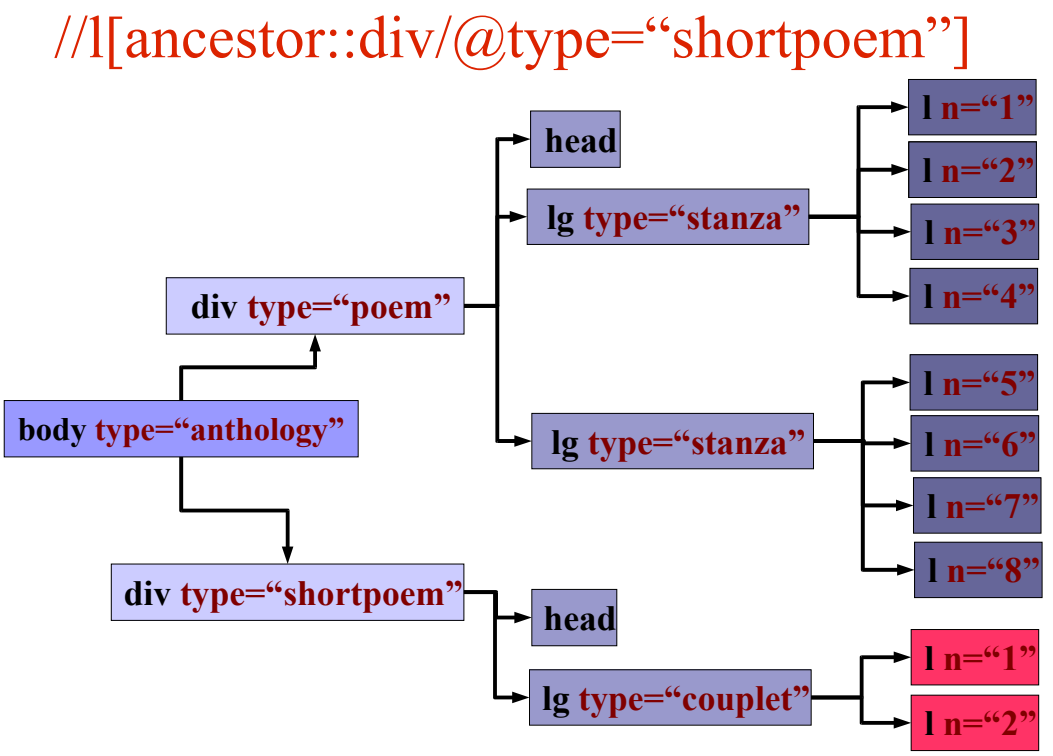
//div[head]/lg/l[@n="2"]



//l[ancestor::div/@type="shortpoem"] ?

ancestor:: is an unabbreviated axis name





1.4 XPath: More About Paths

- A location path results in a node-set
- Paths can be absolute (`/div/lg[1]/l`)
- Paths can be relative (`l/../../head`)
- Formal Syntax: (`axisname::nodetest[predicate]`)
- For example:`child::div[contains(head, 'ROSE')]`

1.5 XPath: Axes

`ancestor::` Contains all ancestors (parent, grandparent, etc.) of the current node

`ancestor-or-self::` Contains the current node plus all its ancestors (parent, grandparent, etc.)

`attribute::` Contains all attributes of the current node

`child::` Contains all children of the current node

`descendant::` Contains all descendants (children, grandchildren, etc.) of the current node

`descendant-or-self::` Contains the current node plus all its descendants (children, grandchildren, etc.)

1.6 XPath: Axes (2)

`following::` Contains everything in the document after the closing tag of the current node

`following-sibling::` Contains all siblings after the current node

`parent::` Contains the parent of the current node

`preceding::` Contains everything in the document that is before the starting tag of the current node

`preceding-sibling::` Contains all siblings before the current node

`self::` Contains the current node

1.7 Axis examples

- `ancestor::lg` = all `<lg>` ancestors
- `ancestor-or-self::div` = all `<div>` ancestors or current
- `attribute::n` = n attribute of current node
- `child::l` = `<l>` elements directly under current node
- `descendant::l` = `<l>` elements anywhere under current node
- `descendant-or-self::div` = all `<div>` children or current
- `following-sibling::l[1]` = next `<l>` element at this level
- `preceding-sibling::l[1]` = previous `<l>` element at this level
- `self::head` = current `<head>` element

1.8 XPath: Predicates

- `child::lg[attribute::type='stanza']`
- `child::l[@n='4']`
- `child::div[position()=3]`
- `child::div[4]`
- `child::l[last()]`
- `child::lg[last()-1]`

1.9 XPath: Abbreviated Syntax

- nothing is the same as `child::`, so `lg` is short for `child::lg`
- @ is the same as `attribute::`, so `@type` is short for `attribute::type`
- . is the same as `self::`, so `./head` is short for `self::node()/child::head`
- .. is the same as `parent::`, so `../lg` is short for `parent::node()/child::lg`
- // is the same as `descendant-or-self::`, so `div//l` is short for `child::div/descendant-or-self::node()/child::l`

1.10 XPath: Operators

XPath has support for numerical, equality, relational, and boolean expressions

+	Addition	3 + 2 = 5	
-	Subtraction	10 - 2 = 8	
	Multiplication	6 * 4 = 24	
div	Division	8 div 4 = 2	
mod	Modulus	5 mod 2 = 1	
=	Equal	@age ≠ '74'	True
or	Boolean OR	@age = '74' or @age = '64'	True

1.11 XPath: Operators (cont.)

<	Less than	@age < '84'	True
!=	Not equal	@age != '74'	False
<=	Less than or equal	@age <= '72'	False
>	Greater than	@age > '25'	True
>=	Greater than or equal	@age >= '72'	True
and	Boolean AND	@age <= '84' and @age > '70'	True

1.12 XPath Functions: Node-Set Functions

- `count()` Returns the number of nodes in a node-set: `count(person)`
- `id()` Selects elements by their unique ID : `id('S3')`
- `last()` Returns the position number of the last node : `person[last()]`
- `name()` Returns the name of a node: `//*[name('person')]`
- `namespace-uri()` Returns the namespace URI of a specified node: `namespace-uri(persName)`
- `position()` Returns the position in the node list of the node that is currently being processed : `//person[position()=6]`

1.13 XPath Functions: String Functions

- `concat()` Concatenates its arguments: `concat('http://', $domain, '/', $file, '.html')`
- `contains()` Returns true if the second string is contained within the first string: `//persName[contains(surname, 'van')]`
- `normalize-space()` Removes leading and trailing whitespace and replaces all internal whitespace with one space: `normalize-space(surname)`
- `starts-with()` Returns true if the first string starts with the second: `starts-with(surname, 'van')`
- `string()` Converts the argument to a string: `string(@age)`

1.14 XPath Functions: String Functions (2)

- `substring` Returns part of a string of specified start character and length: `substring(surname, 5,4)`
- `substring-after()` Returns the part of the string that is after the string given: `substring-after(surname, 'De')`
- `substring-before` Returns the part of the string that is before the string given: `substring-before(@date, '-')`
- `translate()` Performs a character by character replacement. It looks at the characters in the first string and replaces each character in the first argument by the corresponding one in the second argument: `translate('1234', '24', '68')`

1.15 XPath Functions: Numeric Functions

- `ceiling()` Returns the smallest integer that is not less that the number given: `ceiling(3.1415)`
- `floor()` Returns the largest integer that is not greater than the number given: `floor(3.1415)`
- `number()` Converts the input to a number: `number('100')`
- `round()` Rounds the number to the nearest integer: `round(3.1415)`
- `sum()` Returns the total value of a set of numeric arguments: `sum(//person/@age)`
- `not()` Returns true if the condition is false: `not(position() >5)`

1.16 XPath: Where can I use XPath?

Learning all these functions, though a bit tiring to begin with, can be very useful as they are used throughout XML technologies, but especially in XSLT and XQuery.

2 XSLT

XSLT is Extensible Stylesheet Language - Transformations, and is the main method used today for transforming input XML into output text/HTML/XML.

2.1 XSLT

The XSLT language is

- Expressed in XML; uses namespaces to distinguish output from instructions
- Purely functional
- Reads and writes XML trees
- Designed to generate XSL FO, but now widely used to generate HTML

2.2 How is XSLT used? (1)

- With a command-line program to transform XML (eg to HTML)
  - Downside: no dynamic content, user sees HTML
  - Upside: no server overhead, understood by all clients
- In a web server *servlet*, eg serving up HTML from XML (eg Cocoon, Axkit)
  - Downside: user sees HTML, server overhead
  - Upside: understood by all clients, allows for dynamic changes

2.3 How is XSLT used? (2)

- In a web browser, displaying XML on the fly
  - Downside: many clients do not understand it
  - Upside: user sees XML
- Embedded in specialized program
- As part of a chain of production processes, performing arbitrary transformations

2.4 XSLT implementations

MSXML Built into Microsoft Internet Explorer

Saxon Java-based, standards leader, implements XSLT 2.0 (basic version free)

Xalan Java-based, widely used in servlets (open source)

libxslt C-based, fast and efficient (open source)

transformiix C-based, used in Mozilla (open source)

2.5 What do you mean, `transformation'?

Take this

```
<recipe>
<title>Pasta for beginners</title>
<ingredients><item>Pasta</item>
<item>Grated cheese</item>
</ingredients>
<cook>Cook the pasta and mix with the cheese</cook>
</recipe>
```

and make this

```
<html>
<h1>Pasta for beginners</h1>
<p>Ingredients: Pasta Grated cheese</p>
<p>Cook the pasta and mix with the cheese</p>
</html>
```

2.6 How do you express that in XSL?

```
<xsl:stylesheet version="1.0"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="recipe">
  <html>
   <h1>
    <xsl:value-of select="title"/>
   </h1>
   <p>Ingredients:
   <xsl:apply-templates select="ingredients/item"/>
   </p>
   <p>
    <xsl:value-of select="cook"/>
   </p>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

2.7 Structure of an XSL file

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:tei="http://www.tei-c.org/ns/1.0">
 <xsl:template match="tei:div">
<!-- .... do something with TEI div elements....-->
 </xsl:template>
 <xsl:template match="tei:p">
<!-- .... do something with TEI p elements....-->
 </xsl:template>
</xsl:stylesheet>
```

The `tei:div` and `tei:p` are *XPath expressions*, which specify which bit of the document is matched by the template.

Any element not starting with **xsl:** in a template body is put into the output.

2.8 The Golden Rules of XSLT

1. If there is no template matching an element, we process the elements inside it
2. If there are no elements to process by Rule 1, any text inside the element is output
3. Children elements are not processed by a template unless you explicitly say so
4. `xsl:apply-templates select="XX"` looks for templates which match element "XX"; `xsl:value-of select="XX"` simply gets any text from that element
5. The order of templates in your program file is immaterial
6. You can process any part of the document from any template
7. Everything is well-formed XML. Everything!

2.9 Technique (1): `apply-templates`

```
<xsl:template match="/"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <html>
  <xsl:apply-templates/>
 </html>
</xsl:template>
```

```
<xsl:template match="tei:TEI"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:apply-templates select="tei:text"/>
</xsl:template>
```

```
<xsl:template match="tei:text"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <h1>FRONT MATTER</h1>
```

```
 <xsl:apply-templates select="tei:front"/>
 <h1>BODY MATTER</h1>
 <xsl:apply-templates select="tei:body"/>
</xsl:template>
```

2.10 Technique (2): `value-of`

Templates for paragraphs and headings:

```
<xsl:template match="tei:p"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <p>
  <xsl:apply-templates/>
 </p>
</xsl:template>
<xsl:template match="tei:div">
 <h2>
  <xsl:value-of select="tei:head"/>
 </h2>
 <xsl:apply-templates/>
</xsl:template>
<xsl:template match="tei:div/tei:head"/>
```

Notice how we avoid getting the heading text twice.
Why did we need to qualify it to deal with just `<head>` inside `<div>`?

2.11 Technique (3): `choose`

Now for the lists. We will look at the `type' attribute to decide what sort of HTML list to produce:

```
<xsl:template match="tei:list"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:choose>
  <xsl:when test="@type='ordered'">
   <ol>
    <xsl:apply-templates/>
   </ol>
  </xsl:when>
  <xsl:when test="@type='unordered'">
   <ul>
    <xsl:apply-templates/>
   </ul>
  </xsl:when>
  <xsl:when test="@type='gloss'">
   <dl>
    <xsl:apply-templates/>
   </dl>
  </xsl:when>
 </xsl:choose>
</xsl:template>
```

```
<xsl:template match="tei:item"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <li>
  <xsl:apply-templates/>
 </li>
</xsl:template>
```

2.12 Technique (4): `number`

It would be nice to get sections numbered, so let us change the template and let XSLT do it for us:

```
<xsl:template match="tei:div"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <h2>
  <xsl:number level="multiple" count="tei:div"/>
  <xsl:text>. </xsl:text>
  <xsl:value-of select="tei:head"/>
 </h2>
 <xsl:apply-templates/>
</xsl:template>
```

2.13 Technique (5): `number`

We can number notes too:

```
<xsl:template match="tei:note"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">[<xsl:number level="any"/>
 <xsl:text>: </xsl:text>
 <xsl:apply-templates/>]
</xsl:template>
```

2.14 Technique (6): `sort`

Let's summarize some manuscripts, *sorting* them by repository and ID number

```
<xsl:template match="tei:TEI"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <ul>
  <xsl:for-each select=".//tei:msDesc">
   <xsl:sort select="tei:msIdentifier/tei:repository"/>
   <xsl:sort select="tei:msIdentifier/tei:idno"/>
   <li>
    <xsl:value-of select="tei:msIdentifier/tei:repository"/>:
   <xsl:value-of select="tei:msIdentifier/tei:settlement"/>:
   <xsl:value-of select="tei:msIdentifier/tei:idno"/>
   </li>
  </xsl:for-each>
 </ul>
</xsl:template>
```

2.15 Technique (7): @mode

You can process the same elements in different ways using modes:

```
<xsl:template match="/"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:apply-templates select=".//tei:div" mode="toc"/>
 <xsl:apply-templates/>
</xsl:template>
<xsl:template match="tei:div" mode="toc">Heading<xsl:value-of se-
lect="tei:head"/>
</xsl:template>
```

This is a very useful technique when the same information is processed in different ways in different places.

2.16 Technique (8): variable

Sometimes you want to store some information in a variable

```
<xsl:template match="tei:figure"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:variable name="n">
  <xsl:number/>
 </xsl:variable>
Figure<xsl:value-of select="$n"/>
 <a name="P{$n}"/>
 <xsl:apply-templates/>
</xsl:template>
```

2.17 Technique (9): template

You can store common code in a named template, with parameters:

```
<xsl:template match="tei:div"
   xmlns:tei="http://www.tei-c.org/ns/1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <html>
  <xsl:call-template name="header">
   <xsl:with-param name="title" select="tei:head"/>
  </xsl:call-template>
  <xsl:apply-templates/>
 </html>
</xsl:template>
<xsl:template name="header">
 <xsl:param name="title"/>
 <head>
  <title>
   <xsl:value-of select="$title"/>
  </title>
 </head>
</xsl:template>
```

2.18 Top-level commands

`<xsl:import href="...">`: include a file of XSLT templates, overriding them as needed
`<xsl:include href="...">`: include a file of XSLT templates, but do not override them
`<xsl:output>`: specify output characteristics of this job

2.19 Some useful `xsl:output` attributes

`method="`xml | html | text"
`encoding="`*string*"
`omit-xml-declation="`yes | no"
`doctype-public="`*string*"
`doctype-system="`*string*"
`indent="`yes | no"

2.20 An identity transform

```
<xsl:output
  method="xml"
  indent="yes"
  encoding="iso-8859-1"
  doctype-system="teixlite.dtd"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:tei="http://www.tei-c.org/ns/1.0"/>
<xsl:template match="/">
 <xsl:copy-of select="."/>
</xsl:template>
```

2.21 A near-identity transform

```
<xsl:template match="*|@*|processing-instruction()"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:tei="http://www.tei-c.org/ns/1.0">
 <xsl:copy>
  <xsl:apply-templates
    select="*|@*|processing-instruction()|comment()|text()"/>
 </xsl:copy>
</xsl:template>
<xsl:template match="tei:p">
 <para xmlns="http://www.tei-c.org/ns/1.0">
 <xsl:apply-templates/>
 </para>
</xsl:template>
```

2.22 Summary

The core techniques:

- template rules for nodes in the incoming XSL
- taking material from other nodes
- processing nodes several times in different modes
- variables and functions
- choosing, sorting, numbering
- different types of output

3 XQuery

While XSLT is good for transforming XML to other formats (XML, HTML, PDF, Text, etc.) sometimes you may wish to query a large database of XML documents and extract only matching records. In cases such as this, XQuery might be more appropriate.

3.1 What is XQuery?

- It is a domain-specific method for accessing and manipulating XML
- It is meant for querying XML
- It is built upon XPath
- It is like SQL but for XML
- A W3C recommendation

3.2 XQuery: Expressions

path expressions return a nodeset

element constructors return a new element

FLWOR expressions analogous to SQL Select statement

list expressions operations on lists or sets of values

conditional expressions traditional if then else construction

qualified expressions boolean operations over lists or sets of values

datatype expressions test datatypes of values

3.3 XQuery: Path Expression

The simplest kind of XQuery that you've already seen:

```
document("test.xml")//p
//p/foreign[@xml:lang='lat']
//foreign[@xml:lang='lat']/text()
//tei:TEI//tei:person[@age >= 25]
```

3.4 XQuery: Element constructor

You may construct elements and embed XQueries or create well-formed results in the output you return. These may contain literal text or variables:

```
<latin>o tempora o mores</latin>
<latin>{$s}</latin>
<li>Name: {$surname}, {$forename}</li>
<li>Birth Country:
{data($person/tei:birth/tei:placeName/tei:country)}
</li>
```

3.5 XQuery: FLWOR expressions

For - Let - Where - Order - Return

```
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $t in document("book.xml")//tei:text
let $latinPhrases := $t//tei:foreign[@xml:lang='lat']
where count($latinPhrases) > 1
order by count($latinPhrases)
return
ID: {data($t/@xml:id)}Phrases: {$latinPhrases}
```

- `for` defines a cursor over an xpath
- `let` defines a name for the contents of an xpath
- `where` selects from the nodes
- `order` sorts the results
- `return` specifies the XML fragments to construct
- Curly braces are used for grouping, and defining scope

3.6 XQuery: List Expressions

XQuery expressions manipulate lists of values, for which many operators are supported:

- constant lists: (7, 9, <thirteen/>)
- integer ranges: i to j
- XPath expressions
- concatenation
- set operators: | (or union), intersect, except
- functions: remove, index-of, count, avg, max, min, sum, distinct-values ...

3.7 XQuery: List Expressions (cont.)

When lists are viewed as sets:

- XML nodes are compared on their node identity
- Any duplicates which exist are removed
- Unless re-ordered the database order is preserved

3.8 XQuery: Conditional Expressions

```
<div> {
if document("xqt")//tei:title/text()
="Introduction to XQuery"
then <p>This is true.</p>
else <p>This is false.</p> }
</div>
```

More often used in user-defined functions

3.9 XQuery: Datatype Expressions

- XQuery supports all datatypes from XML Schema, both primitive and complex types
- Constant values can be written:
  - as literals (like string, integer, float)
  - as constructor functions (true(), date("2001-06-07"))
  - as explicit casts (cast as xsd:positiveInteger(47))
- Arbitrary XML Schema documents can be imported into an XQuery
- An `instance of` operator allows runtime validation of any value relative to a datatype or a schema.
- A `typeswitch` operator allows branching based on types.

3.10 XQuery and Namespaces

- As with XPath queries, if your documents are in a particular namespace then this namespace must be declared in your query
- Namespace prefixes must be used to access any element in a namespace
- Multiple namespaces can be declared and used
- Output can be in a different namespace to the input

3.11 XQuery Example: Multiple Variables

One of the real benefits that XQuery gives you over XPath queries is that you can define multiple variables:

```
(: All Person Elements  with their Stone's Description :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $stones in collection('/db/pc')//tei:TEI
let $stoneDesc := $stones//tei:stoneDescription
let $people := $stones//tei:person
return
<div>{$people} {$stoneDesc}</div>
```

3.12 XQuery Example: Element Constructors

You can construct the results into whatever elements you want:

```
( : Women's Birth and Death Countries in placeName elements :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
let $stones := collection('/db/pc')//tei:TEI
for $person in $stones//tei:person[@sex = '2']
let $birthCountry := $person/tei:birth//tei:country/text()
let $deathCountry := $person/tei:death//tei:country/text()
return
<div><p>This woman was born in
<placeName>{$birthCountry}</placeName>
and died in
<placeName>{$deathCountry}</placeName>.</p>
</div>
```

3.13 Result: Element Constructors

A series of `<div>` elements like:

```
<div>
<p>This woman was born in
<placeName> France </placeName>
and died in
<placeName> Italy </placeName>.
</p>
</div>
```

3.14 XQuery Example: Traversing the Tree

You are not limited to one section of the document:

```
(: Getting the Stone's Title :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
let $stones := collection('/db/pc')//tei:TEI
for $person in $stones//tei:person[@sex ='2']
let $birthCountry := $person/tei:birth//tei:country/text()
let $deathCountry := $person/tei:death//tei:country/text()
let $title :=
$person/ancestor::tei:TEI//tei:teiHeader//tei:title[1]/text()
return
<div>
<head>{$title}</head>
<p>This woman was born in
<placeName>{$birthCountry}</placeName> and died in
<placeName>{$deathCountry}</placeName>.</p>
</div>
```

3.15 Result: Traversing the Tree

A series of `<div>` elements like:

```
<div>
<head> Stone 2 </head>
<p> This woman was born in
<placeName> France </placeName>
and died in
<placeName> Italy </placeName>. </p>
</div>
```

3.16 XQuery Example: Using Functions

```
(: Getting birth date attribute and parts thereof :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
let $stones := collection('/db/pc')//tei:TEI
for $person in $stones//tei:person[@sex ='2']
let $birthCountry := $person/tei:birth//tei:country/text()
let $birthDate := $person/tei:birth/@date
let $title := $person/ancestor::tei:TEI//tei:title[1]/text()
```

```
return
<div>
<head>{$title}</head>
<p>This woman was born in
<placeName>{$birthCountry}</placeName>.
The date of their birth was in the year
<date>{$birthDate}
{data(substring-before($birthDate, '-'))}</date>.
</p>
</div>
```

3.17 Result: Using Functions

A series of `<div>` elements like:

```
<div>
<head> Stone 2 </head>
<p> This woman was born in
<placeName> France </placeName>.
The date of their birth was in the year
<date date="1882-10-02"> 1882 </date>.</p>
</div>
```

3.18 XQuery Example: Nesting For Loops

```
(: Number of people on a stone, women's forenames :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
for $stones in collection('/db/pc')//tei:TEI
let $title := $stones//tei:teiHeader//tei:title[1]/text()
let $number := count($stones//tei:person)
return
<div>
<head>{$title}</head>
<p>This stone has {$number} people.</p>
{ for $person in $stones//tei:person[@sex ='2']
let $forename := $person//tei:forename/text()
return
<p>This stone has woman whose
first name is {$forename}.</p>
}</div>
```

3.19 Result: Nesting For Loops

A series of `<div>` elements like:

```
<div>
<head> Stone 2 </head>
<p> This stone has 3 people.</p>
<p> This stone has woman whose first name is Hilda. </p>
</div>
```

3.20 XQuery Example: Embedding in HTML

```
(: XQuery nested inside HTML :)
declare namespace tei="http://www.tei-c.org/ns/1.0";
<html>
<head><title>Stones with numbers</title></head>
<body>
{
for $stones in collection('/db/pc')//tei:TEI
let $title := $stones//tei:teiHeader//tei:title[1]/text()
let $number := count($stones//tei:person)
return
<div>
<h1>{$title}</h1>
<p>This stone has {$number} people.</p>
</div>
}
</body>
</html>
```

3.21 Result: Embedding in HTML

An HTML document like:

```
<html>
<head>
<title> Stones with numbers </title>
</head>
<body>
<div>
<h1> Stone 1 </h1>
<p> This stone has 1 people. </p>
</div>
<div>
<h1> Stone 2 </h1>
<p> This stone has 3 people. </p>
</div>
<!-- Many more stones -->
</body>
</html>
```

3.22 XQuery in Practice

- You can handle requests passed from HTML forms inside your XQueries
- That XQuery is being used is invisible to the user
- eXist may be embedded into Apache's Cocoon web publishing framework
- You can send queries to eXist in all sorts of ways including from within Java programs and via HTTP/REST, XML-RPC, SOAP, WebDAV.
- You can use XUpdate to add/change/delete nodes in the XML database