XQuery Declarations

The **declare** clause specifies things like

- Namespaces: declare namespace pref='value'
  - Predefined prefixes include XML, XML Schema, XML Schema-Instance, XPath, and **local**
- Settings: declare boundary-space preserve (or strip)
- Default collation: a URI to be used for collation when no collation is specified
XQuery Quantification: 1

- Two quantifiers **some** and **every**
- Each quantifier expression evaluates to true or false
- Each quantifier introduces a bound variable, analogous to **for**

```
for $x in ... where some $y in ...
satisfies $y ... $x
return ...
```

Here the second $x refers to the **same** variable as the first
XQuery Quantification: 2

A typical useful quantified expression would use variables that were introduced outside of its scope

- The order of evaluation is implementation-dependent: enables optimization
- If some bindings produce errors, this can matter
- **some**: trivially false if no variable bindings are found that satisfy it
- **every**: trivially true if no variable bindings are found
Variables: Scoping, Bound, and Free

*for, let, some,* and *every* introduce variables

- The visibility of a variable follows typical scoping rules.
- A variable referenced within a scope is
  - *Bound* if it is declared within the scope
  - *Free* if it is not declared within the scope

```c
1 for $x$ in ...
where some $x$ in ...
satisfies ...
return ...
```

Here the two $x$ refer to *different* variables.
NO SIDE EFFECT

let $x := 1$

let $x := 2$

$x = 2$

$x = 1$
XQuery Conditionals

Like a classical **if-then-else** clause

- The **else** is not optional
- Empty sequences or node sets, written ( ), indicate that nothing is returned
XQuery Constructors

Braces { } to delimit expressions that are evaluated to generate the content to be included; analogous to macros

- **document { }**: to create a document node with the specified contents
- **element { } { }**: to create an element
  - **element foo { 'bar' }**: creates `<foo>Bar</foo>`
  - **element { 'foo' } { 'bar' }**: also evaluates the name expression
- **attribute { } { }**: likewise
- **text { body}**: simpler, because anonymous
XQuery Effective Boolean Value

Analogous to Lisp, a general value can be treated as if it were a Boolean

- A `xs:boolean` value maps to itself
- Empty sequence maps to `false`
- Sequence whose first member is a node maps to `true`
- A numeric that is 0, negative, or NaN maps to `false`, else `true`
- An empty string maps to `false`, others to `true`
[] generalized array accessors
(Song-A, Song-B, Song-C)

... /Song [17]
[1]  [last()]
[0]  ← nothing
[<foo/7>]
[1 or 1]  all songs
Defining Functions

```
declare function local:itemftop($t)
{
    local:itemf($t,())
}
```

- Here `local:` is the namespace of the query
- The arguments are specified in parentheses
- All of XQuery may be used within the defining braces
- Such functions can be used in place of XPath expressions
Functions with Types

```perl
$ declare function local::itemftop($t as element())
   as element() *
{
    local::itemf($t,())
};
```

- Return types as above
- Also possible for parameters, but ignore such for this course
XSLT

A programming language with a functional flavor

- Specifies (stylesheet) transforms from documents to documents
- Can be included in a document (best not to)

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
    href="URL-to-xsl-sheet"?>
<main-element>
  5  ...
</main-element>
```
XQuery versus XSLT: 1

Competitors in some ways, but

- Share a basis in XPath
- Consequently share the same data model
- Same type systems (in the type-sensitive versions)
- XSLT got out first and has a sizable following, but XQuery has strong backing among vendors and researchers
XQuery versus XSLT: 2

- XQuery is geared for querying databases
  - Supported by major relational DBMS vendors in their XML offerings
  - Supported by native XML DBMSs
  - Offers superior coverage of processing joins
  - Is more logical (like SQL) and potentially more optimizable

- XSLT is geared for transforming documents
  - Is functional rather than declarative
  - Based on template matching
XQuery versus XSLT: 3

There is a bit of an arms race between them

- Types
  - XSLT 1.0 didn’t support types
  - XQuery 1.0 does
  - XSLT 2.0 does too

- XQuery presumably will be enhanced with capabilities to make updates, but XSLT could too
XSLT Stylesheets

A programming language that follows XML syntax

- Use the XSLT namespace (conventionally abbreviated `xsl`)
- Includes a large number of primitives, especially:
  - `<copy-of>` (deep copy)
  - `<copy>` (shallow copy)
  - `<value-of>`
  - `<for-each select="...">`
  - `<if test="...">`
  - `<choose>`
XSLT Templates: 1

A pattern to specify where the given transform should apply: an XPath expression

- This match only works on the root:

  ```xml
  <xsl:template match="/">
    <output>...
  </xsl:template>
  ```

- Example: Duplicate text in an element

  ```xml
  <xsl:template match="text()">
    2
    <xsl:value-of select='.'/>
    <xsl:value-of select='.'/>
  </xsl:template>
  ```
XSLT Templates: 2

- If no pattern is specified, apply recursively on et-children via `<xsl:apply-templates/>`

- By default, if no other template matches, recursively apply to et-children of current node (ignores attributes) and to root:

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

(IMPlicitly) Control Flows Down A TREE
XSLT Templates: 3

- Copy text node by default
- Use an empty template to override the default:

```xml
  <xsl:template match="X"/>
  <!-- X = desired pattern -->
```

Confine ourselves to the examples discussed in class (ignore explicit priorities, for example)
XSLT Templates: 4

- Templates can be named
- Templates can have parameters
  - Values for parameters are supplied at invocation
  - Empty node sets by default
  - Additional parameters are ignored
XSLT Variables

- Explicitly declared
- Values are node sets
- Convenient way to document templates
REVERSE:
BEGINNING FROM item[1]

items
/ item item item
a b c

template match = 'item'
next-sib 

when EST $next-sib
apply template $next-sib
copy $""
\( \Rightarrow \) `item` [last()]

match: `item`

```plaintext
prec = preceding rule
if child* then
  \& \& \text{last()} [last()]
else just local
```