What is XPath?

XPath is an expression language that allows the processing of values conforming to a data model that includes:

• a tree representation of XML documents and
• atomic values such as integers, strings, and booleans, and
• sequences that may contain both references to nodes in an XML document and atomic values.

The result of an XPath expression may be a selection of nodes from the input documents, or an atomic value, or more generally, any sequence allowed by the data model.

The name of the language derives from its most distinctive feature, the path expression (μονοπάτι), which provides a means of hierarchical addressing of the nodes in an XML tree.
Data Model

• Treats an XML document as a logical tree consisting of 7 kinds of nodes:
  • Root Node – the root of the document
  • Element Nodes – one for each element in the document
    • Unique ID’s
  • Attribute Nodes
  • Namespace Nodes
  • Processing Instruction Nodes
  • Comment Nodes
  • Text Nodes
• The tree structure is ordered and reads from top to bottom and left to right
Data Model Example

For this simple doc:

```
<doc>
  <?Pub Caret?>
  <para>Some <em>emphasis</em> here. </para>
  <para>Some more stuff.</para>
</doc>
```

Might be represented as:

```
root
  ↓
  doc
    ↓
    ?pi?
    ↓
    para
      ↓
      em
        ↓
        text
      ↓
      para
        ↓
        text
```
What is XPath?

- A Flexible notation for navigating around trees
  - Address ‘parts’ of an XML document, and provide basic facilities for manipulation of strings, numbers and booleans.

- A basic technology that is widely used
  - uniqueness and scope in XML Schema
  - pattern matching and selection in XSLT
  - relations in XLink and XPointer
  - computations on values in XSLT and Xquery

- W3C Recommendation. November 16, 1999
- Latest version: http://www.w3.org/TR/xpath
Introductory Remarks

• XPath uses a compact, string-based, rather than XML element-based, syntax.

• Operates on the abstract, logical structure of an XML document (tree of nodes) rather than its surface syntax.

• Uses a path notation (like URLs/directory paths) to navigate through this hierarchical tree structure.
Introductory Remarks

• Defines a way to compute a string-value for each type of node: element, attribute, text.

• Supports Namespaces.

• Name of a node (a pair consisting of a local part and namespace URI).

• Expression (Expr) is the primary syntactic construct.
Outline

- Location steps and paths
- Typical locations paths
- Abbreviations
- XPath functions
Location Paths

- A *location path* (μονοπάτι) evaluates to a *sequence* of nodes
- The sequence is *sorted* in document order
- The sequence will *never* contain *duplicates*

- Location Paths can be *absolute* or *relative*:
  - `/books` : absolute location path starts from the root of an XML document
  - `book/title`

- Location Paths can be *compound*:
  `/customers/customer/invoice | /customers/customer/cust_info`
Locations Steps

- The location path is a sequence of steps
- The *Location Steps* (βήματα) are found between the slashes in a full location path
- A *location step* consists of
  - an *axis*
  - a *nodetest*
  - some *predicates*
- Location step syntax:
  
  \[
  \text{axis} :: \text{nodetest} \ [\text{Exp}_1] \ [\text{Exp}_2] \ldots
  \]
  
  - Only the “nodetest” part is required
Evaluating a Location Path

- A Location Step maps a context node (συγκείμενο) into a sequence of nodes
- This also maps sequences to sequences
  - each node in the sequence is used as context node
  - and is replaced with the result of applying the step
- The path then applies each step in turn
An Example
An Example

Context node
An Example

descendant::C
An Example

descendant::C/child::E
An Example

descendant::*C/child::*E/child::*F
Axis

- Αφού εντοπίσουμε τους κόμβους-καταλήξεις κάποιων μονοπατιών, ο “άξονας” (axis) μας επιτρέπει να καθορίσουμε την “κατεύθυνση” προς την οποία θέλουμε να αναζητήσουμε τους κόμβους που ικανοποιούν την αναζήτησή μας.
- Η “κατεύθυνση” καθορίζεται πάντοτε σε σχέση με κάποιον τρέχοντα κόμβο.
- An axis is a sequence of nodes
- The axis is evaluated relative to the context node
Axes

- XPath supports 12 different axes

- child
- descendant
- parent
- ancestor
- following-sibling
- preceding-sibling

- attribute
- following
- preceding
- self
- descendant-or-self
- ancestor-or-self
Axis Directions

- Each axis has a *direction*
- *Forwards* means document order:
  - child, descendant, following-sibling, following, self, descendant-or-self
- *Backwards* means reverse document order:
  - parent, ancestor, preceding-sibling, preceding
- Stable but depends on the implementation:
  - attribute
# Location step axes

<table>
<thead>
<tr>
<th>Axis</th>
<th>Description</th>
<th>Direction</th>
<th>Visible node types</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::</td>
<td>Εντοπίζει κόμβους που είναι απευθείας παιδία του τρέχοντος</td>
<td>Forward</td>
<td>Elements, comments, Pis, text nodes</td>
</tr>
<tr>
<td>parent::</td>
<td>Εντοπίζει τον μοναδικό κόμβο-πατέρα</td>
<td>Reverse</td>
<td>Root node, elements</td>
</tr>
<tr>
<td>descendant::</td>
<td>Εντοπίζει όλους τους απογόνους του τρέχοντος κόμβου</td>
<td>Forward</td>
<td>Elements, comments, Pis, text nodes</td>
</tr>
<tr>
<td>descendant-or-self::</td>
<td>Όπως και παραπάνω, με την πρόσθεση του τρέχοντος</td>
<td>Forward</td>
<td>Any but attributes or namespaces</td>
</tr>
<tr>
<td>following::</td>
<td>Όλοι οι ορατοί κόμβοι που ακολουθούν τον τρέχοντα (χωρίς τους απογόνους του)</td>
<td>Forward</td>
<td>Any but root node, attributes or namespaces</td>
</tr>
</tbody>
</table>
## Location step axes

<table>
<thead>
<tr>
<th>Axis</th>
<th>Description</th>
<th>Dir</th>
<th>Visible node types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ancestor::</td>
<td>Όλοι οι κόμβοι που είναι πρόγονοι του τρέχοντος</td>
<td>Reverse</td>
<td>Root node, Elements</td>
</tr>
<tr>
<td>preceding::</td>
<td>Όλοι οι κόμβοι που προηγούνται του τρέχοντος (εκτός των προγόνων)</td>
<td>Reverse</td>
<td>Any but root node, attributes or namespaces</td>
</tr>
<tr>
<td>following-sibling::</td>
<td>Όλοι οι κόμβοι που έπονται του τρέχοντος και με τον οποίο έχουν τον ίδιο πατέρα</td>
<td>Forward</td>
<td>Any but root node, attributes or namespaces</td>
</tr>
<tr>
<td>preceding-sibling::</td>
<td>Όλοι οι ορατοί κόμβοι που προηγούνται του τρέχοντος και με τον οποίο έχουν τον ίδιο πατέρα</td>
<td>Reverse</td>
<td>Any but root node, attributes or namespaces</td>
</tr>
<tr>
<td>namespace::</td>
<td>Κόμβοι ονοματολογίας</td>
<td>Forward</td>
<td>Namespaces only</td>
</tr>
<tr>
<td>attribute::</td>
<td>Κόμβοι-Κατηγορήματα του τρέχοντος κόμβου</td>
<td>Forward</td>
<td>Attributes only</td>
</tr>
<tr>
<td>self::</td>
<td>Ο τρέχων κόμβος</td>
<td>-</td>
<td>Any</td>
</tr>
</tbody>
</table>
The parent Axis
The child Axis
The descendant Axis
The ancestor Axis
The **following-sibling** Axis
The preceding-sibling Axis
The following Axis
The preceding Axis
Defaults & Shortcuts

- "child::" is the default axis: The following two location steps are identical:
  - child::circle
  - circle

- Steps that access the parent node via the parent axis are common and are abbreviated using "..":
  - parent::node() <==> ..

- When seeking an attribute node:
  - attribute::copyright <==> @copyright
The context of an XPath evaluation consists of:

- a context node (a node in an XML tree)
- a context position and size (two nonnegative integers)
- a set of variable bindings mapping variable names to values
- a function library
- a set of namespace declarations mapping prefixes to namespace URIs

The application determines the initial context.

If the path starts with ‘/’ then:

- the initial context node is the root
- the initial position and size are 1

During evaluation of multiple location steps, the context node, position and size change.
The Node Test

- Selects the sort of nodes in a document, in which we are interested.
- Two main approaches to select nodes:
  - Identify the *names* of the nodes we are interested in
  - Identify the *types* of the nodes
Node Tests

- `text()` - selects all text nodes that are children of the context node
  - `paragraph/text()`
- `comment()` - selects all comment nodes that are children of the context node
- `processing-instruction(target)` - selects all PI nodes that are children of the context node and have a particular target
  - `/processing-instruction(‘xml-stylesheet’)`
- `processing-instruction()` - selects all PI nodes that are children of the context node
- `node()` - selects all nodes regardless of their type or name
- `*` - selects elements with any name
- `name` - selects nodes with the particular name
- `*:name` - selects nodes with the given name in any namespace
- `prefix:*` - selects elements with any name in the prefix namespace
**XPATH Predicates**

- Usually coded as Boolean statements of the general form:

  \[ value1 \text{ operator } value2 \]

- Where \( value1 \) and \( value2 \) are General XPath expressions (location paths or literal values)

- **Operators** are: \( = \), \( != \), \( > \), \( < \), \( >= \), \( <= \)

- Result is coerced into a boolean
  - a number yields true if it equals the context position
  - a string yields true if it is not empty
  - a sequence yields true if it is not empty
XPath Predicates

- Evaluated with the *current node as context*
- Any relative location paths appearing in the predicate are considered relative to the context node established by the portions of the location step that precede the predicate, *not* relative to the context node in effect for the location step as a whole.

```xml
<person name="John">
  <child name="John">
    <child name="Cindy">
      <child name="Connie">
      </child>
    </child>
  </child>
</person>
```

```xml
/person/child[@name='John']
/person/child[@name='Cindy']
```

```xml
<html/body/p[@align='center']/img[@border &gt; 0]
```
Predicates with a single value

- Use a location path with no operator or value
- Used when we want to check for the existence of a node along some axis

```
//book[descendant::table]
```
- Selects only “book” elements that contain at least one “table” descendant
- Works because XPath treats an empty node-set as a Boolean false and a non empty node-set as a Boolean true
Nested & Compound Predicates

//roofing_material[descendant::type[preceding-sibling::manufacturer='Smith']]  
- Select all “roofing_material” elements that have a “type” descendant for which there exists a “manufacturer” on the preceding sibling axis whose name is “Smith”

  camera[brand/@name = ‘Minolta’ and brand/@list < 300]
- Selects a camera child of the context node only if:
  - the “camera” element has a brand child with a “name” attribute whose value is Minolta
    and
  - the “camera” element has a brand child, which has a “list” attribute whose value is less than 300

//reading[@time="1200" or @time="1800") or (wind_spd < 15 and wind_dir="S") or (temp < 25)]
- Selects a “reading” element with a “time” attribute taken either at noon or at 6pm, OR reporting a wind speed less than 15, as long as the wind direction was south (“S”) OR reporting a temperature less than 25 degrees.
Numeric-valued predicates

- When a predicate’s value is (or evaluates to) a number, the predicate is used to select a node that has a particular context position within the node-set selected by the preceding portion of the location step.

  //reading[3] or //reading[position() = 3]

- Returns the third reading child of all elements in a document, regardless of its contents

  //reading[position()=3 or temp < 25]
Stacked predicates

axis:node-test[predicate1]…[predicateN]

- Each succeeding stacked predicate is evaluated in terms of the narrowed context provided by the preceding ones.

```xml
<tosses>
  <toss result="heads"/>
  <toss result="heads"/>
  <toss result="tails"/>
  <toss result="heads"/>
</tosses>
```

Empty node-set

```xml
(//toss)[@result="heads"][3]
(//toss)[3][@result="heads"]
```
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient
Abbreviations
Abbreviations
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute::amount
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount
Abbreviations

/rcp:collection/rcp:recipe/rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount

/descendant-or-self::node()/
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount

/descendant-or-self::node()//
Abbreviations

/rcp:collection/rcp:recipe/rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount

/descendant-or-self::node()/. //

self::node()
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount

/descendant-or-self::node()/
Abbreviations

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient

/rcp:collection/rcp:recipe/rcp:ingredient

/child::rcp:collection/child::rcp:recipe/child::rcp:ingredient/attribute:::amount

/rcp:collection/rcp:recipe/rcp:ingredient/@amount

/descendant-or-self::node()/

//

self::node() .

parent::node() ..
XPath Functions

- Function types (according to argument types):
  - String: argument is a string value
  - Nodeset: argument is a node-set, represented by an Xpath location path
  - Boolean: argument has a Boolean value (true or false)
  - Number: argument has a numeric value
  - Anytype: argument can be any of several types (function handles type conversion)
  - ? - the questionmark appended to one of the above data types means the argument is optional
Functions

- XPath has an extensive *function library*
- Default *namespace* for functions:
  
  http://www.w3.org/2004/07/xpath-functions

- 106 functions are required
- More functions with the *namespace*:
  
  http://www.w3.org/2001/XMLSchema
Function Invocation

- Calling a function with 4 arguments:
  \[ \text{fn:avg}(1, 2, 3, 4) \]

- Calling a function with 1 argument:
  \[ \text{fn:avg}((1, 2, 3, 4)) \]
# Node-Set Functions

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>last()</td>
<td>number</td>
<td>Number of nodes in the context node-set</td>
</tr>
<tr>
<td>position()</td>
<td>number</td>
<td>Ordinal position of the context node within the context node-set</td>
</tr>
<tr>
<td>count(nodeset)</td>
<td>number</td>
<td>Number of nodes in nodeset</td>
</tr>
<tr>
<td>id(string)</td>
<td>Node-set</td>
<td>Element node with an ID-type attribute equals to the value of string</td>
</tr>
<tr>
<td>local-name(nodeset?)</td>
<td>String</td>
<td>QName without a namespace prefix of first node in nodeset</td>
</tr>
<tr>
<td>namespace-uri (nodeset?)</td>
<td>String</td>
<td>URI associated with the namespace prefix of first node in nodeset</td>
</tr>
<tr>
<td>name(nodeset?)</td>
<td>String</td>
<td>QName of first node in nodeset</td>
</tr>
</tbody>
</table>
# String Functions

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string(anytype?)</code></td>
<td>String</td>
<td>Converts <em>anytype</em> to string and returns it</td>
</tr>
<tr>
<td><code>concat(string1,string2,...)</code></td>
<td>String</td>
<td>Concatenates strings and returns result</td>
</tr>
<tr>
<td><code>starts-with(string1,string2)</code></td>
<td>boolean</td>
<td>True if <em>string1</em> begins with <em>string2</em></td>
</tr>
<tr>
<td><code>contains(string1,string2)</code></td>
<td>boolean</td>
<td>True if <em>string1</em> contains <em>string2</em></td>
</tr>
<tr>
<td><code>substring(string,num1,num2)</code></td>
<td>String</td>
<td>Returns portion of <em>string</em> from character <em>num1</em>, and length <em>num2</em></td>
</tr>
<tr>
<td><code>substring-before (string1,string2)</code></td>
<td>String</td>
<td>Returns portion of <em>string1</em> occurring before <em>string2</em></td>
</tr>
<tr>
<td><code>String-length(string?)</code></td>
<td>number</td>
<td>Returns number of characters in <em>string</em></td>
</tr>
</tbody>
</table>
**String Functions**

\[
\begin{align*}
\text{concat("X","ML")} &= \text{"XML"} \\
\text{concat("X","ML"," ","book")} &= \text{"XML book"} \\
\text{string-join(("XML","book")," ")} &= \text{"XML book"} \\
\text{string-join(("1","2","3"),"+")} &= \text{"1+2+3"} \\
\text{substring("XML book",5)} &= \text{"book"} \\
\text{substring("XML book",2,4)} &= \text{"ML b"} \\
\text{string-length("XML book")} &= 8 \\
\text{upper-case("XML book")} &= \text{"XML BOOK"} \\
\text{lower-case("XML book")} &= \text{"xml book"}
\end{align*}
\]
## Boolean Functions

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean(anytype)</td>
<td>boolean</td>
<td>Converts anytype to true or false</td>
</tr>
<tr>
<td>not(boolean)</td>
<td>boolean</td>
<td>Returns true</td>
</tr>
<tr>
<td>true</td>
<td>boolean</td>
<td>Returns true</td>
</tr>
<tr>
<td>false</td>
<td>boolean</td>
<td>Returns false</td>
</tr>
<tr>
<td>lang(string)</td>
<td>boolean</td>
<td>True if language in the context node matches the value of string</td>
</tr>
</tbody>
</table>
Boolean Functions

\[\text{fn:not}(0) = \text{fn:true}()\]
\[\text{fn:not}() = \text{fn:false}()\]
\[\text{fn:not}(\text{""}) = \text{fn:true}()\]
\[\text{fn:not}(1) = \text{fn:false}()\]
Arithmetic Functions

fn:abs(-23.4) = 23.4
fn:ceiling(23.4) = 24
fn:floor(23.4) = 23
fn:round(23.4) = 23
fn:round(23.5) = 24
fn:number(anytype?): converts anytype to numeric value
fn:contains("XML book","XML") = \text{fn:}true()
fn:matches("XML book","XM..[a-z]*") = \text{fn:}true()
fn:matches("XML book",".*Z.*") = \text{fn:}false()
fn:replace("XML book","[a-z],"8") = "XML 8888"
Cardinality Functions

\[
\begin{align*}
\text{fn:exists(())} &= \text{fn:false()} \\
\text{fn:exists((1,2,3,4))} &= \text{fn:true()} \\
\text{fn:empty(())} &= \text{fn:true()} \\
\text{fn:empty((1,2,3,4))} &= \text{fn:false()} \\
\text{fn:count((1,2,3,4))} &= 4 \\
\text{fn:count(//rcp:recipe))} &= 5
\end{align*}
\]
Sequence Functions

\[
\begin{align*}
\text{fn:distinct-values}((1, 2, 3, 4, 3, 2)) &= (1, 2, 3, 4) \\
\text{fn:insert-before}((2, 4, 6, 8), 2, (3, 5)) &= (2, 3, 5, 4, 6, 8) \\
\text{fn:remove}((2, 4, 6, 8), 3) &= (2, 4, 8) \\
\text{fn:reverse}((2, 4, 6, 8)) &= (8, 6, 4, 2) \\
\text{fn:subsequence}((2, 4, 6, 8, 10), 2) &= (4, 6, 8, 10) \\
\text{fn:subsequence}((2, 4, 6, 8, 10), 2, 3) &= (4, 6, 8)
\end{align*}
\]
Coercion Functions

xs:integer("5") = 5
xs:integer(7.0) = 7
xs:decimal(5) = 5.0
xs:decimal("4.3") = 4.3
xs:decimal("4") = 4.0
xs:double(2) = 2.0E0
xs:double(14.3) = 1.43E1
xs:boolean(0) = fn:false()
xs:boolean("true") = fn:true()
xs:string(17) = "17"
xs:string(1.43E1) = "14.3"
xs:string(fn:true()) = "true"
Aggregate Functions

\[
\begin{align*}
\text{fn:avg}((2, 3, 4, 5, 6, 7)) &= 4.5 \\
\text{fn:max}((2, 3, 4, 5, 6, 7)) &= 7 \\
\text{fn:min}((2, 3, 4, 5, 6, 7)) &= 2 \\
\text{fn:sum}((2, 3, 4, 5, 6, 7)) &= 27
\end{align*}
\]
Node Functions

fn:doc("http://www.brics.dk/ixwt/recipes/recipes.xml")
fn:position()
fn:last()
General XPath Expressions

- Every expression evaluates to a sequence of items, which are:
  - atomic values
  - nodes
- Atomic values may be:
  - numbers
  - booleans
  - Unicode strings
  - datatypes defined in XML Schema
- Nodes identify a particular node in a given XML tree, and have identity
Atomization

- A sequence may be *atomized* resulting in a sequence of *atomic values*.

- This sequence is obtained by replacing every node with its *string value* according to these rules:
  - *Text node*: string value is its contents
  - *Element node*: string value is the *concatenation* of all descendant *text nodes*
  - *Attribute node*: the attribute value
  - *Root node*: string value is the concatenation in doc. order of string values of all descendant text nodes.
Literal Expressions

42
3.1415
6.022E23
’XPath is a lot of fun’
”XPath is a lot of fun”
’The cat said ”Meow!”’
”The cat said ””Meow!”””
”XPath is just
so much fun”
Arithmetic Expressions

- +, -, *, div, idiv, mod
- Operators are generalized to sequences
  - if any argument is empty, the result is empty
  - if all arguments are singleton sequences of numbers, the operation is performed
  - otherwise, a runtime error occurs
Variable References

- $foo-17 refers to the variable "foo-17"
- Possible fixes: 
  - ($foo)-17, $foo -17, $foo+-17
Sequence Expressions

- The ',' operator concatenates sequences
- Integer ranges are constructed with 'to'
- Operators: union, intersect, except
- Sequences are always flattened
- These expressions give the same result:

\[
(1, (2, 3, 4), ((5)), (), (((6, 7), 8, 9)))
\]

1 to 9
1, 2, 3, 4, 5, 6, 7, 8, 9
Path Expressions

- Locations paths are expressions
- They may start from arbitrary sequences
  - evaluate the path for each node
  - use the given node as context node
  - context position and size are taken from the sequence
  - the results are combined in document order
Filter Expressions

- Predicates can be generalized to *arbitrary* sequences containing mixtures of atomic values and nodes.

\[
\text{exp1 [ exp2 ]}
\]

This allows filtering by an arbitrary expression that is coerced into boolean.

- The expression ‘.’ is the *context item*

- The expression:

\[
(10 \text{ to } 40)[. \mod 5 = 0 \text{ and position}>20]
\]

has the result:

30, 35, 40
Value Comparison

- Operators: eq, ne, lt, le, gt, ge
- Used on atomic values
- When applied to arbitrary values:
  - atomize
  - if either argument is empty, the result is empty
  - if either has length >1, the result is false
  - if incomparable, a runtime error
  - otherwise, compare the two atomic values

8 eq 4+4
(//rcp:ingredient)[1]/@name eq "beef cube steak"
### General Comparison

- **Operators:** =, !=, <, <=, >, >=
- **Used on general values:**
  - atomize
  - if there exists two values, one from each argument, whose comparison holds, the result is true
  - otherwise, the result is false

8 = 4+4  
(1,2) = (2,4)  
//rcp:ingredient/@name = "salt"
Node Comparison

- Operators: `is`, `<<`, `>>`
- Used to compare nodes on identity and order
- When applied to arbitrary values:
  - if either argument is empty, the result is empty
  - if both are singleton nodes, the nodes are compared
  - otherwise, a runtime error

```
(//rcp:recipe)[2] is
  //rcp:recipe[rcp:title eq "Ricotta Pie"]
/rcp:collection << (//(rcp:recipe)[4]
(//(rcp:recipe)[4] >> (//(rcp:recipe)[3]
```
Be Careful About Comparisons

\[
((//rcp:ingredient)[40]/@name,(//rcp:ingredient)[40]/@amount) \text{ eq } ((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
\]

Yields false, since the arguments are not singletons

\[
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[40]/@amount) = ((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
\]

Yields true, since the two names are found to be equal

\[
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[40]/@amount) \text{ is } ((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[53]/@amount)
\]

Yields a runtime error, since the arguments are not singletons
Algebraic Axioms for Comparisons

- **Reflexivity:** \( x = x \)
- **Symmetry:** \( x = y \implies y = x \)
- **Transitivity:**
  \[ x = y \land y = z \implies x = z \]
  \[ x < y \land y < z \implies x < z \]
- **Anti-symmetry:**
  \[ x \leq y \land y \leq x \implies x = y \]
- **Negation:**
  \[ x \neq y \iff \neg x = y \]
XPath Violates Most Axioms

- Reflexivity?
  \((\) = (\) yields false

- Transitivity?
  \((1, 2) = (2, 3), (2, 3) = (3, 4), \text{ not } (1, 2) = (3, 4)\)

- Anti-symmetry?
  \((1, 4) \leq (2, 3), (2, 3) \leq (1, 4), \text{ not } (1, 2) = (3, 4)\)

- Negation?
  \((1) \neq (\) yields false, \((1) = (\) yields false
Boolean Expressions

- Operators: `and`, `or`
- Arguments are coerced, false if the value is:
  - the boolean `false`
  - the empty sequence
  - the empty string
  - the number zero
- Constants use functions `true()` and `false()`
- Negation uses `not(...)"
For Expressions

- The expression

```xml
for $r$ in //rcp:recipe
  return fn:count($r//rcp:ingredient[fn:not(rcp:ingredient)])
```

returns the value

```
11, 12, 15, 8, 30
```

- The expression

```xml
for $i$ in (1 to 5)
  for $j$ in (1 to $i$)
    return $j$
```

returns the value

```
1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5
```
Conditional Expressions

fn:avg(
   for $r in //rcp:ingredient return
   if ( $r/@unit = "cup" )
      then xs:double($r/@amount) * 237
   else if ( $r/@unit = "teaspoon" )
      then xs:double($r/@amount) * 5
   else if ( $r/@unit = "tablespoon" )
      then xs:double($r/@amount) * 15
   else ()
)

Quantified Expressions
Quantified Expressions

Some $r$ in //rcp:ingredient satisfies $r/@name$ eq "sugar"

$$\text{fn:exists(}
\text{for } r \text{ in } //rcp:ingredient \text{ return }
\text{if } (r/@name \text{ eq } "sugar") \text{ then } \text{fn:true()} \text{ else } ()
\)$$
XPath 1.0 Restrictions

- Many implementations only support XPath 1.0
- Smaller function library
- Implicit casts of values
- Some expressions change semantics:

  "4" < "4.0"

  is false in XPath 1.0 but true in XPath 2.0
XPointer

- A fragment identifier mechanism based on XPath
- Different ways of pointer to the fourth recipe:

```xml
...#xpointer//recipe[4])
...#xpointer(rcp:recipe[./rcp:title ='Zuppa Inglese'])
...#element(/1/5)
...#r102
```
XLink

- Generalizing hyperlinks from HTML to XML
- Allow many-to-many relations
- Allow third party links
- Allow arbitrary element names

- Not widely used…
An XLink Link

```xml
<mylink xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:type="extended">
  <myresource xlink:type="locator"
      xlink:href="students.xml#Carl" xlink:label="student"/>
  <myresource xlink:type="locator"
      xlink:href="students.xml#Fred" xlink:label="student"/>
  <myresource xlink:type="locator"
      xlink:href="teachers.xml#Joe" xlink:label="teacher"/>
  <myarc xlink:type="arc"
      xlink:from="student" xlink:to="teacher"/>
</mylink>
```
A Picture of the XLink

mylink

Carl

Fred

myresource

myarc

myresource

myresource

myarc

Joe
Simple Links
Simple Links

```xml
<mylink xlink:type="simple" xlink:href="..." xlink:show="..."/>

<mylink xlink:type="extended">
  <myresource xlink:type="resource"
    xlink:label="local"/>
  <myresource xlink:type="locator"
    xlink:label="remote" xlink:href="..."/>
  <myarc xlink:type="arc"
    xlink:from="local" xlink:to="remote" xlink:show="..."/>
</mylink>
```
<a xlink:type="simple"
xlink:href="...
xlink:show="replace"
xlink:actuate="onRequest"/>
The HLink Alternative

<hlink namespace="http://www.w3.org/1999/xhtml"
  element="a"
  locator="@href"
  effect="replace"
  actuate="onRequest"
  replacement="@target"/>
Essential Online Resources

- http://www.w3.org/TR/xpath/
- http://www.w3.org/TR/xpath20/
- http://www.w3.org/TR/xlink/
- http://www.w3.org/TR/xptr-framework/