Introduction to XPath

Sources
- XML Path Language (XPath) Version 1.0, http://www.w3.org/TR/xpath
- http://www.w3schools.com/xpath/xpath_examples.asp
- Essential XML Quick Reference (A. Skonnard and M. Gudgin)
- http://www.w3schools.com/xpath
- XML In a Nutshell, O'Reilly, Harold & Means

XPath 1.0
- Examples
- Data Model
- Syntax
- Location paths
- Expressions
- Functions

XPath Examples
A CD catalog with entries such as:
<cd>
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <country>UK</country>
  <company>CBS Records</company>
  <price>9.90</price>
  <year>1988</year>
</cd>
/catalog/cd: selects all the cd nodes

<cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
</cd>
<cd>
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <country>UK</country>
  <company>CBS Records</company>
  <price>9.90</price>
  <year>1988</year>
</cd>

........
File at www.cs.technion.ac.il/~oshmu/cd.xml

/catalog/cd[1]: selects the first cd node

<cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
</cd>

/catalog/cd/price: selects price nodes

<price>10.90</price>
<price>9.90</price>
<price>9.90</price>
<price>10.20</price>
<price>9.90</price>
<price>10.90</price>
<price>8.90</price>
<price>7.90</price>
<price>7.20</price>
<price>7.80</price>
<price>8.20</price>

/catalog/cd/price/text(): selects price text nodes


/catalog/cd [price < 7.80] : selects cd nodes whose price text value is less than 7.80

<cd>
  <title>Picture book</title>
  <artist>Simply Red</artist>
  <country>EU</country>
  <company>Elektra</company>
  <price>7.20</price>
  <year>1985</year>
</cd>

/catalog/cd [price < 7.80]/ price : selects price nodes whose text value is less than 7.80

<price>7.20</price>

/catalog/cd [price < 7.80]/ price/text() : selects text nodes within price nodes whose text value is less than 7.80

7.20

Semantics of Location Paths

- A relative location path consists of a sequence of one or more location steps separated by /.
- The steps in a relative location path are composed together from left to right.
- Each step in turn selects a set of nodes relative to a context node.
- An initial sequence of steps is composed together with a following step as follows.
  - The initial sequence of steps selects a set of nodes relative to a context node.
  - Each node in that set is used as a context node for the following step.
  - The sets of nodes identified by that step are unioned together. The set of nodes identified by the composition of the steps is this union.
**Semantics of Location Paths**

- An absolute location path consists of `/` optionally followed by a relative location path.
- A `/` by itself selects the root node of the document containing the current context node.
- If it is followed by a relative location path, then the location path selects:
  - the set of nodes that would be selected by the relative location path relative to the root node of the document containing the context node.

**Xpath Data Types**

- **Boolean**: true, false
- **Number**: floating point
- **String**: sequence of characters
- **Node-sets**: node collection, no duplicates
- **Document order**: the order in which start-tags appear (DFS on the tree)

**Example File**

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <musthave>yes</musthave>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <scratch>yes</scratch>
    <price>9.90</price>
  </cd>
</catalog>
```

**Navigation**

- Generally the syntax is either of the form `/location step/.../location step` or `location step/.../location step`
- If path starts with `/` then it matches the root document node (absolute path)
- Otherwise, it is a relative path that matches the context node
- With each step there is an associated set of context nodes
- For each node in this set the next step is evaluated
- The union of the resulting sets forms the next context set
Xpath 1.0 Semantics of Location Paths

- Start with context set $M$ ({root} if starts with /).
- Let $e = \text{loc1} / \text{loc2} / \ldots / \text{locn};$
- For $i := 1$ to $n$ do:
  - Let $loc_i = \text{loc pred1} \ldots \text{predm};$
  - For each context set node $v$ in $M$:
    - Evaluate loc on $v$ and obtain a set of nodes $S_v_0$.
    - For $j := 1$ to $m$ do
      - $S_v_{j} := \text{apply}(S_v_{j-1}, \text{predj});$
    - $S_v := S_v_{m};$
    - $M := \text{Union}_v S_v.$
- Return $M$

Apply($S, \text{pred}$)
- Return $\{v | v \in S \text{ and pred}(v, S) == \text{true}\}$

Navigation Example

- Select all the price elements of all the cd elements of the catalog element:
  - $/\text{catalog}/\text{cd}/\text{price}$
    - $<$price>10.90</price>$
    - $<$price>9.90</price>$
    - $<$price>9.90</price>$
  - In fact, in its full version this is:
    - $/\text{child::catalog}/\text{child::cd}/\text{child::price}$

Axes

- A location step is of the form
  - $/\text{catalog}/\text{cd}/\text{price} | /\text{catalog}/\text{cd}/\text{title}$
    - $<$title>Empire Burlesque</title>$
    - $<$price>10.90</price>$</cd>
    - $<$title>Hide your heart</title>$
    - $<$price>9.90</price>$</cd>
    - $<$title>Greatest Hits</title>$
    - $<$price>9.90</price>$</cd>
  - The result order is document order (always, for union)
  - $\text{cd}/\text{price}$ identifies the child price elements of the context node’s cd child
Axes (Cont.)

- Each axis has a principal node type
- When identifying nodes via * or via name, only nodes of the principal type are candidates
- The attribute axis has principal node type of Attribute
- The namespace axis has principal node type of Namespace
- All other axes have principal node type of Element

self

- Identifies the context node
- /catalog/cd/self::cd
- Same as:
  - /catalog/cd

child

- Identifies the child nodes of the context node
- Default axis
  - /catalog/cd
- Same as:
  - /catalog/child::cd
- Same as:
  - /child::catalog/child::cd

parent

- Identifies the parent node of the context node
- /catalog/cd/parent::catalog
- Same as:
  - /catalog/cd/parent::catalog/cd/parent::catalog
descendant and descendant-or-self

- Identifies the descendant nodes of the context node
- `/catalog/descendant::title`
  - `<title>Empire Burlesque</title>`
  - `<title>Hide your heart</title>`
  - `<title>Greatest Hits</title>`
- `/catalog/descendant-or-self::title` returns the same result
- `/catalog/descendant-or-self::catalog` returns the catalog node

ancestor and ancestor-or-self

- Identifies the ancestor nodes of the context node
- `/catalog/descendant::title/ancestor::cd`
  - returns the three cd nodes
- `/catalog/descendant::title/ancestor::catalog` returns the catalog node
- `/catalog/descendant::title/ancestor-or-self::title` returns the three title nodes:
  - `<title>Greatest Hits</title>`
  - `<title>Hide your heart</title>`
  - `<title>Empire Burlesque</title>`
- `/catalog/descendant::title/ancestor-or-self::cd` returns the three cd nodes.

following

- Identifies the nodes, except for descendant nodes, attribute nodes and namespace nodes, which follow the context node in document order
- `/catalog/descendant::scratch/following::*` returns `<price>9.90</price>`
- `/catalog/descendant::scratch/parent::cd/child::title/following::*` returns
  - `<artist>Dolly Parton</artist>`
  - `<scratch>yes</scratch>`
  - `<price>9.90</price>`

following-sibling

- `/catalog/cd/following-sibling` returns
  `<cd country="UK">
   <title>Hide your heart</title>
   <artist>Bonnie Tyler</artist>
   <price>9.90</price>
  </cd>`
  `<cd country="USA">
   <title>Greatest Hits</title>
   <artist>Dolly Parton</artist>
   <scratch>yes</scratch>
   <price>9.90</price>
  </cd>`
preceding

- Identifies the nodes, except for ancestor nodes, attribute nodes and namespace nodes, which precede the context node in document order
- /catalog/descendant::musthave/preceding::* returns (note the reverse document order)
  - <artist>Bonnie Tyler</artist>
  - <title>Hide your heart</title>
  - <price>10.90</price> </cd>
  - <artist>Bob Dylan</artist>
  - <title>Empire Burlesque</title>
  - <cd country="USA">
      <title>Empire Burlesque</title>
      <artist>Bob Dylan</artist>
      <price>10.90</price> </cd>

preceding-sibling

- /catalog/descendant::musthave/
  preceding-sibling::*
  - <artist>Bonnie Tyler</artist>
  - <title>Hide your heart</title>

attribute

- Identifies the attributes of the context node.
- /catalog/cd/attribute::* returns
  - country="USA"
  - country="UK"
  - country="USA"

namespace

- Identifies the namespace nodes of the context node.
Node Tests

- Node Test by name
- Node Test by type

Node Test by name - Examples

- Suppose prefix j is bound to namespace urn:eorg:invoice
- Then, child::j:item identifies child item element nodes of the context node in the namespace urn:eorg:invoice
- child::j:* identifies child element nodes of the context node in the namespace urn:eorg:invoice
- /child::catalog identifies child catalog element nodes of the root that belong to no namespace

Node Test by type

- text() : node is a text node
- comment() : node is a comment node
- processing-instruction(target?)
- node()

Node Test by type - Examples

- child::node() identifies all child nodes of the context node regardless of type
- //scratch/child::text() returns text node yes
- //scratch/text() also returns text node yes
- //cd/price/text() returns 3 text nodes 10.90 9.90 9.90
- /catalog/comment() identifies comment child nodes of the root’s catalog child element
- /processing-instruction(‘xsl-stylesheet’) identifies processing instruction child nodes of the document node that has target equal to xsl-stylesheet
### Shorthand notation

<table>
<thead>
<tr>
<th>Long form</th>
<th>Short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>child::</td>
<td>Short form</td>
</tr>
<tr>
<td>attribute::</td>
<td>@</td>
</tr>
<tr>
<td>self::node()</td>
<td>.</td>
</tr>
<tr>
<td>parent::node()</td>
<td>..</td>
</tr>
<tr>
<td>/descendant-or-self::node()</td>
<td>//</td>
</tr>
<tr>
<td>[position() = number]</td>
<td>[number]</td>
</tr>
</tbody>
</table>

### Shorthand notation - Examples

<table>
<thead>
<tr>
<th>Long form</th>
<th>Short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>/child::catalog//child::cd</td>
<td>/catalog//cd</td>
</tr>
<tr>
<td>/child::catalog/@attribute::country</td>
<td>/catalog/@country</td>
</tr>
<tr>
<td>/self::node()//descendant-or-self::node()//child::title</td>
<td>//title (how about //title or /title/title ?)</td>
</tr>
<tr>
<td>/descendant-or-self::node()//scratch//parent::node()</td>
<td>//scratch/..</td>
</tr>
</tbody>
</table>

### File as XML, ignore spaces

```xml
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <b>
    <c>
      <d></d>
    </c>
    <e>
      <f>
        <g></g> <h></h>
      </f> <i></i>
    </e> <j></j>
  </b> <k>
    <l></l> <m></m>
  </k> <n>
    <o></o>
  </n>
</a>
```

### Nodes and Axes

<table>
<thead>
<tr>
<th>Axis</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>self</td>
<td>E</td>
</tr>
<tr>
<td>parent</td>
<td>B</td>
</tr>
<tr>
<td>child</td>
<td>F,I</td>
</tr>
<tr>
<td>descendant</td>
<td>F,G,H,I</td>
</tr>
<tr>
<td>descendant-or-self</td>
<td>E,F,G,H,I</td>
</tr>
<tr>
<td>ancestor</td>
<td>B,A, root</td>
</tr>
<tr>
<td>Ancestor-or-self</td>
<td>E,B,A,root</td>
</tr>
<tr>
<td>preceding</td>
<td>D,C</td>
</tr>
<tr>
<td>preceding-sibling</td>
<td>C</td>
</tr>
<tr>
<td>following</td>
<td>J,K,L,M,N,O</td>
</tr>
<tr>
<td>following-sibling</td>
<td>J</td>
</tr>
</tbody>
</table>

**E is the context node**
Predicates

- Zero or more predicates appear, each in square brackets, following the node test
- A predicate may contain any expression; the result is coerced to Boolean
- The 1st predicate is applied to each of the resulting nodes after the node test
- If any evaluates to false, the node is eliminated
- Then, the next predicate is evaluated and so on
- /a/node() [1][2] will always result in an empty result set whereas /a/node() [2][1] may be nonempty

Predicates - Examples

- /catalog/cd [artist = "Bob Dylan"] returns
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>

Expressions

- Boolean Expressions
- Equality expressions
- Relational Expressions
- Numerical expressions
Boolean Expressions

- The operators are **and**, **not** and **or**
- Each operand is evaluated and converted to boolean (similar to applying boolean())
- `/catalog/cd/scratch` *or* `/catalog/@country` returns true
- `/catalog/cd [scratch and price]` returns

  ```xml
  <cd country="USA">
  <title>Greatest Hits</title>
  <artist>Dolly Parton</artist>
  <scratch>yes</scratch>
  <price>9.90</price>
  </cd>
  ```

- `/catalog/cd [not ( scratch and price )]` returns

  ```xml
  <cd country="USA">
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <price>10.90</price>
  </cd>
  <cd country="UK">
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <musthave>yes</musthave>
  <price>9.90</price>
  </cd>
  ```

- `/catalog/cd [(price < 9) or (price > 11)]` returns an empty node set

---

String Value

- Attribute – normalized value
- Text – character data
- **Element, Root** – concatenation of descendant text nodes
- PI – character data following target in `<?target data ?>`
- Comment – character data within comment delimiters
- Namespace – namespace URI

---

Equality expressions: **=**, **!=**

- Equality between objects holds when they are equal
- Equality between node sets holds when there are elements in each that have the same string value, so there is an implicit existential quantifier
- Inequality between node sets holds when there are elements in each that have different string values
- So, two node sets may be equal and unequal at the same time!
- When compared to a number (resp., string, boolean), the string value is converted to a number (resp., string, boolean)
Equality expressions: Examples

• price = 9.90 true if at least one child price element has string value that when converted to a number equals 9.90
• price != 9.90 true if at least one child price element has string value that when converted to a number does not equal 9.90

what if there are no price children?

• not (price = 9.90) true if there is no price child element such that when converted to a number has string value of 9.90

what if there are no price children?

• So, /a [ (price != 5 )] is not the same as /a [ not (price = 5 )]!

• @country = ‘USA’ true for elements that have the value USA for their country attribute

Equality expressions: Examples

• //cd  [scratch = "yes"] returns
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <scratch>yes</scratch>
    <price>9.90</price>
  </cd>
• //cd  [scratch != "yes"] returns an empty node set ("there exists a child of cd whose string value is different than "yes")
• //cd  [not (scratch = "yes") ] returns ("it is not the case that there exists a child of cd whose string value is "yes")
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <musthave>yes</musthave>
    <price>9.90</price>
  </cd>
• //catalog  [cd [not (price = 9.90)] ] returns
  <catalog>
    <cd country="USA">
      <title>Empire Burlesque</title>
      <artist>Bob Dylan</artist>
      <price>10.90</price>
    </cd>
    <cd country="UK">
      <title>Hide your heart</title>
      <artist>Bonnie Tyler</artist>
      <musthave>yes</musthave>
      <price>9.90</price>
    </cd>
    <cd country="USA">
      <title>Greatest Hits</title>
      <artist>Dolly Parton</artist>
      <scratch>yes</scratch>
      <price>9.90</price>
    </cd>
  </catalog>
• //catalog  [not (cd [price = 9.90]) ] returns an empty node set
Equality expressions: Examples

- \( //\text{cd} \ [ \ \text{not ( not ( scratch) ) } ] \) returns

\[
<\text{cd country="USA"}>
  <\text{title}>Greatest Hits</\text{title}>
  <\text{artist}>Dolly Parton</\text{artist}>
  <\text{scratch}>yes</\text{scratch}>
  <\text{price}>9.90</\text{price}>
</\text{cd}>
\]

Coercions

- If neither operand is a node set and the objects have different types, coercion of the lower precedence object to the higher precedence object is performed

- Order: boolean > number > string

- \text{true()} = \text{‘joe’} \ is \ \text{true} \ as \ ‘joe’ \ is \ coerced \ into \ true

- \text{true()} \neq 1.50 \ is \ \text{false} \ as \ 1.50 \ is \ coerced \ into \ true

- \text{“1.56” \ = \ 1.56} \ is \ \text{true} \ as \ “1.56” \ is \ coerced \ into 1.56

Relational Expressions

- \(<, \leq, >, \geq\)
- Both operands are converted into numbers
- Existential semantics as for equality

- \text{price} \geq 50 \ true if there is a child price element
  with a price element whose string value when converted to a number is greater than or equal to 50

- \text{price} < \text{preceding::price} \ true if there is a child price element whose value is smaller than the value of some preceding price element, \textit{what if there is no preceding price element?}

Relational Expressions

- \( //\text{cd} \ [ \ \text{price < preceding::price} ] \) returns

\[
<\text{cd country="UK"}>
  <\text{title}>Hide your heart</\text{title}>
  <\text{artist}>Bonnie Tyler</\text{artist}>
  <\text{musthave}>yes</\text{musthave}>
  <\text{price}>9.90</\text{price}>
</\text{cd}>
<\text{cd country="USA"}>
  <\text{title}>Greatest Hits</\text{title}>
  <\text{artist}>Dolly Parton</\text{artist}>
  <\text{scratch}>yes</\text{scratch}>
  <\text{price}>9.90</\text{price}>
</\text{cd}>
\]
Numerical Expressions

- Increasing precedence: +, -, div, mod, *, unary -
- Each operand is coerced into a number
- $5 + 7 \times 2$ yields 19
- $5 + 7 \times 2 = 19.0$ yields true
- $5 \mod 2$ yields 1
- $\lfloor \text{price div } 2 = 1 \rfloor$ is true for odd prices

Functions

- Node-test functions: id, lang, last, local-name, name, namespace-uri, position
- Boolean functions: boolean, false, not, true
- Numerical functions: ceiling, count, floor, number, round, sum
- String functions: concat, contains, normalize-space, starts-with, string, string-length, substring, substring-after, substring-before, translate

Node-test functions

- id('101') returns the unique element with id 101
- id('101 102') returns the unique elements with ids 101 or 102
- When applied to a node set, each node is converted to its string value and then id is applied to each string value

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>count()</td>
<td>number of nodes in a node-set [count(item) &gt; 5]</td>
<td>number count(node-set)</td>
</tr>
<tr>
<td>id()</td>
<td>Selects elements by their unique ID, see next id (\text{id(book/@similarbook)})</td>
<td>node-set id(value)</td>
</tr>
<tr>
<td>last()</td>
<td>Return position number of the last node in the node sequence, relative to THIS context node (\text{invoice/item [last() &gt; 3]})</td>
<td>number last() (\text{Note: /catalog/cd/node() [last()=3]})</td>
</tr>
<tr>
<td>local-name()</td>
<td>the local-name part of a node (\text{prefix::local-name})</td>
<td>string local-name(node)</td>
</tr>
<tr>
<td>name()</td>
<td>the Qname of a node (\text{note: I\u2019m sorry}:)</td>
<td>string name(node-set?)</td>
</tr>
<tr>
<td>namespace-uri()</td>
<td>the namespace URI of a specified node (\text{uri namespace-uri(node)})</td>
<td></td>
</tr>
<tr>
<td>position()</td>
<td>the position in the node sequence of the node (\text{number position()})</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Signature &amp; Example</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>concat()</td>
<td>the concatenation of all its arguments</td>
<td>string concat(val1, val2, ..)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>concat('The', ' ', 'XML') = 'The XML'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>concat(catalog(id:concat(title,artist)) = &quot;Hide your heart Bonnie Tyler&quot;)</td>
</tr>
<tr>
<td>contains()</td>
<td>true if the second string is contained within the first string</td>
<td>Boolean contains(val, substr)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>contains('XML', 'X') = true</td>
</tr>
<tr>
<td>normalize-space()</td>
<td>Removes leading and trailing spaces from a string</td>
<td>string normalize-space(string)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>normalize-space(' The XML ') = 'The XML'</td>
</tr>
<tr>
<td>starts-with()</td>
<td>true if the first string starts with the second string</td>
<td>boolean starts-with(string, substr)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>starts-with('XML', 'X') = true</td>
</tr>
<tr>
<td>string()</td>
<td>convert the argument to a string</td>
<td>string(value)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>string(128) = '128'</td>
</tr>
<tr>
<td>string-length()</td>
<td>the number of characters in a string</td>
<td>number string-length(string)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>string-length('Israel') = 6</td>
</tr>
<tr>
<td>substring()</td>
<td>the part of the string argument specified in the argument by start and length</td>
<td>string substring(string, start, length)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>substring('Beatles', 1, 4) = 'Beat'</td>
</tr>
<tr>
<td>substring-after()</td>
<td>the part of the string argument that occurs after the substr argument</td>
<td>string substring-after(string, substr)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>substring-after('12/10', '/') = '10'</td>
</tr>
<tr>
<td>substring-before()</td>
<td>the part of the string argument that occurs before the substr argument</td>
<td>string substring-before(string, substr)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>substring-before('12/10', '/') = '12 '</td>
</tr>
<tr>
<td>translate()</td>
<td>character by character replacement, the value argument characters contained in string1 are each replaced, by character for the in the same position in string2</td>
<td>string translate(value, string1, string2)</td>
</tr>
<tr>
<td></td>
<td>Examples</td>
<td>translate('12:30', '30', '45') = '12:45'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>translate('12:30', '03', '54') = '12:45'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>translate('12:30', '0123', 'abcd') = 'bc:da'</td>
</tr>
</tbody>
</table>
### Boolean functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Signature &amp; Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean()</td>
<td>Converts the value argument to Boolean and returns true or false</td>
<td>boolean boolean(value)</td>
</tr>
<tr>
<td>false()</td>
<td>false</td>
<td>false()</td>
</tr>
<tr>
<td>lang()</td>
<td>true if the language argument matches the language of the xml:lang attribute</td>
<td>boolean lang(language)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ignore suffix as in en-us</td>
</tr>
</tbody>
</table>

### Summary

- A simple navigational language.
- Navigates a tree structure.
- Uses various axes.
- Useful functions.
- The “work-horse” of other languages such as Xquery and XSLT.

---

### Boolean functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Signature &amp; Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>not()</td>
<td>true if the condition argument is false</td>
<td>boolean not(condition)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong> not(false())</td>
</tr>
<tr>
<td>true()</td>
<td>true</td>
<td>true()</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong> number(true()) = 1</td>
</tr>
</tbody>
</table>