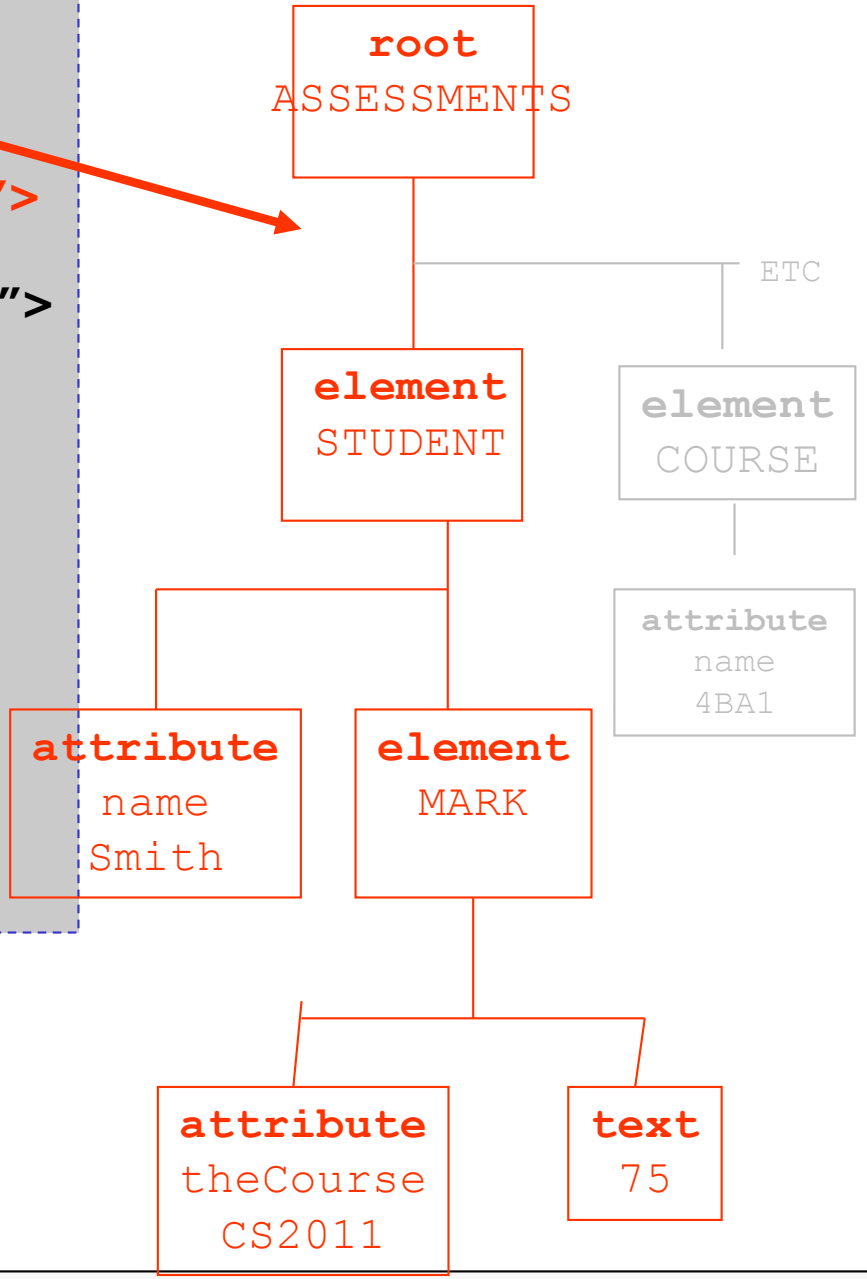
Querying XML documents

XML as a Tree structure and DOM/SAX APIs

XML as a tree structure

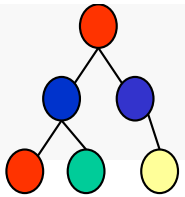
```
<ASSESSMENTS>
        <STUDENT name = "Smith">
            <MARK theCourse = "CS2011">
75 </MARK>
            <MARK theCourse = "CS2012">
99 </MARK>
        </STUDENT> ...


        <COURSE name = "CS2011",
takenBy = "Smith, Jones, ... ">
          <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

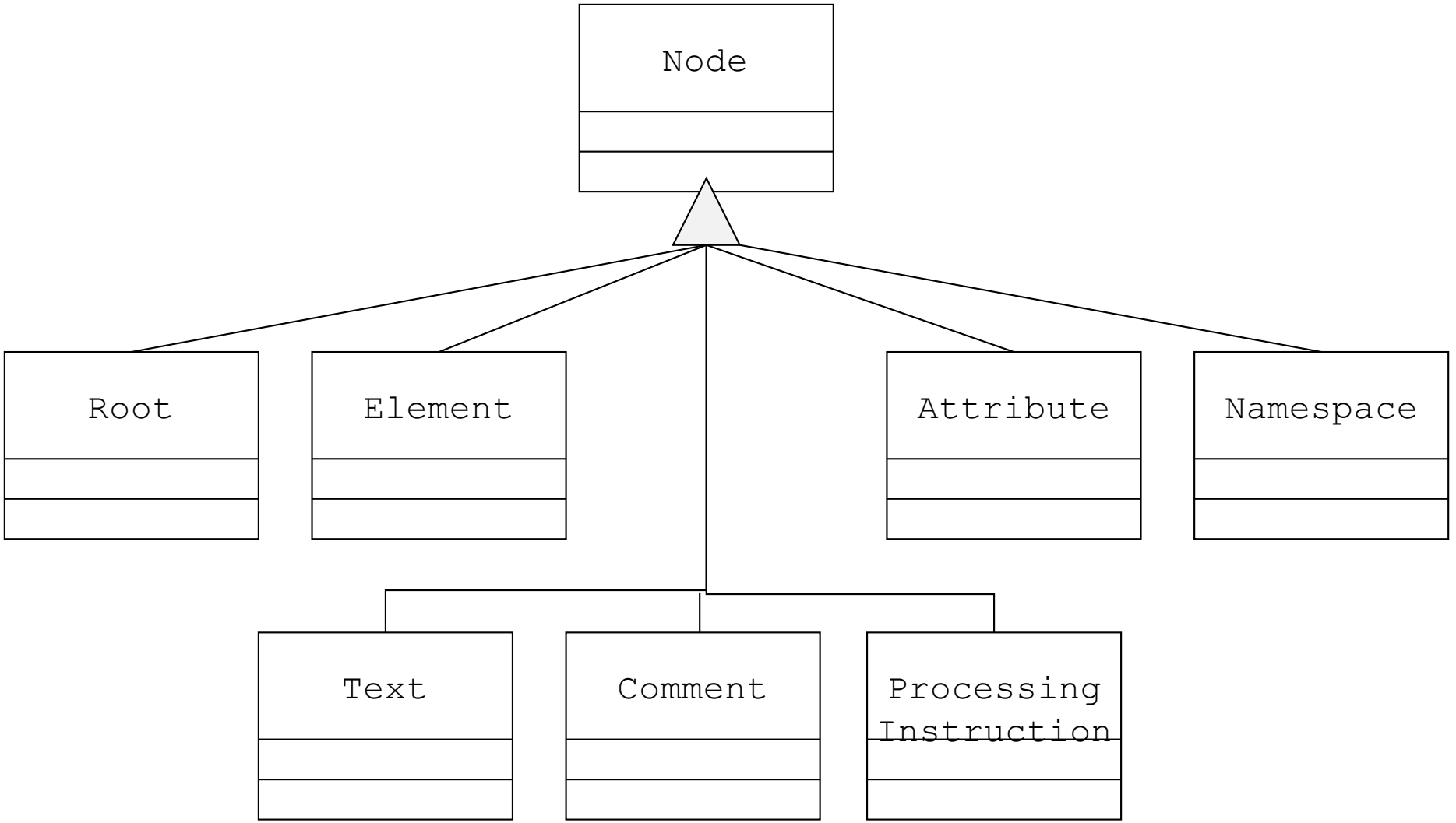


Describes mark for individual student

Describes average mark for course

Nodes in a Tree Model

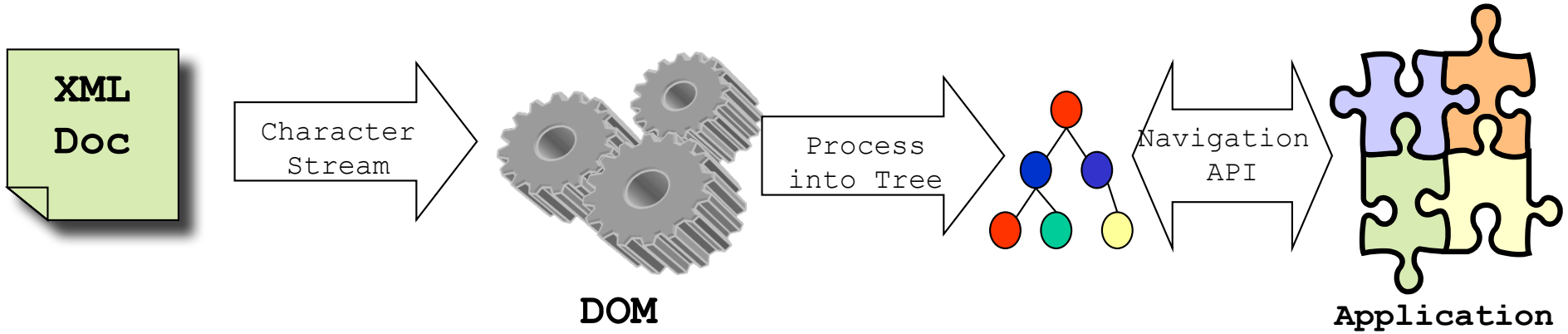




Useful Properties of a Node

- Name
  - Except root, text and comment nodes
- String-value
  - E.g. text if text node, comment text if comment node, attribute value if attribute node
- Child
  - List of child nodes
- Parent
  - Every node except root
- Has-attribute
  - List of attribute nodes associated with element node
- Has-namespace
  - List of namespace nodes associated with element node

XML Processing: DOM Processing



- DOM stands for Document Object Model
- It views an XML tree as a data structure
- It is quite large and complex...
  - Level 1 Core: W3C Recommendation, October 1998
    - primitive navigation and manipulation of XML trees
    - other Level 1 parts: HTML
  - Level 2 Core: W3C Recommendation, November 2000
    - adds Namespace support and minor new features
    - other Level 2 parts: Events, Views, Style, Traversal and Range
  - Level 3 Core: W3C Working Draft, April 2002
    - adds minor new features
    - other Level 3 parts: Schemas, XPath, Load/Save

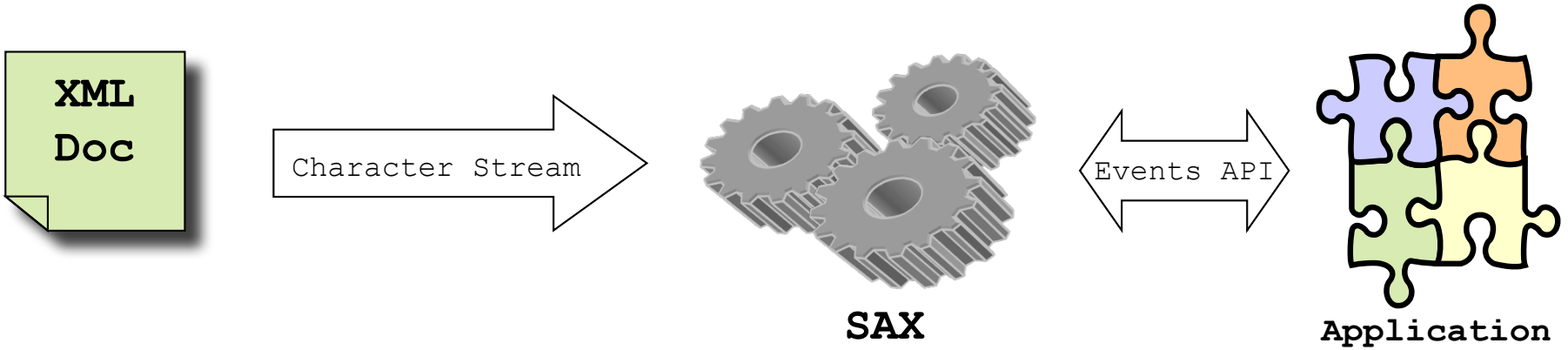
Example: A Recipe

```
<recipe>
<title>Zuppa Inglese</title>
        <ingredient name="egg yolks" amount="4" />
        <ingredient name="milk" amount="2.5" unit="cup" />
        <ingredient name="Savoiardi biscuits" amount="21" />
        <ingredient name="sugar" amount="0.75" unit="cup" />
        <ingredient name="Alchermes liquor" amount="1" unit="cup" />
        <ingredient name="lemon zest" amount="*" />   <ingredient name="flour"
amount="0.5" unit="cup" />
        <ingredient name="fresh whipping cream" amount="*" /> -
        <preparation>  <step>Warm up the milk in a nonstick sauce pan</step>
<step>In a large bowl beat the egg yolks with the sugar, add the flour and
combine the ingredients until well mixed.</step>   <step>Add the milk, a little
bit at the time to the egg mixture, mixing well.</step>   <step>Put the mixture
into the sauce pan and cook it on the stove at a medium low heat. Mix the cream
continuously with a wooden spoon. When it starts to thicken remove it from the
heat and pour it on a large plate to cool off.</step>   <step>Stir the cream
now and then so that the top doesn't harden.</step>   <step>Dip quickly both
sides of the lady fingers in the liquor. Layer them one at the time in a glass
bowl large enough to contain 7 biscuits.</step>   <step>Spread 1/3 of the cream
and repeat the layer with lady fingers. Finish with the cream.</step>
        </preparation>
        <comment>Refrigerate for at least 4 hours better yet overnight. Before
serving decorate the zuppa inglese with whipped cream.</comment>
        <nutrition calories="612" fat="49" carbohydrates="45" protein="4"
alcohol="2" />
</recipe>
```

Example: Getting a Recipe

```
import java.io.*;
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.*;
public class FirstRecipeDOM {
  public static void main(String[] args) {
    try {
      DOMParser p = new DOMParser();
      p.parse(args[0]);
      Document doc = p.getDocument();
      Node n = doc.getDocumentElement().getFirstChild();
      while (n!=null && !n.getNodeName().equals("recipe"))
        n = n.getNextSibling();
      PrintStream out = System.out;
      out.println("<?xml version=\"1.0\"?>");
      out.println("<collection>");
      if (n!=null)
        print(n, out);
      out.println("</collection>");
    } catch (Exception e) {e.printStackTrace();}}
```

XML Processing: SAX Processing



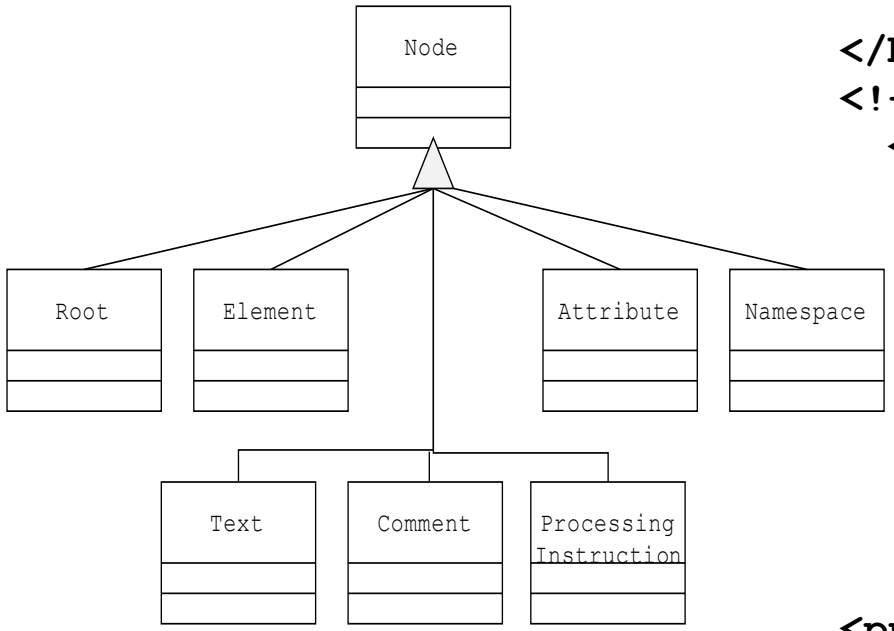
- SAX stands for Simple API for XML
- An XML tree is not viewed as a data structure, but as a stream of events generated by the parser.
- The kinds of events are:
  - the start of the document is encountered
  - the end of the document is encountered
  - the start tag of an element is encountered
  - the end tag of an element is encountered
  - character data is encountered
  - a processing instruction is encountered
- Scanning the XML file from start to end, each event invokes a corresponding callback method that the programmer writes

Example: Getting total amount of Flour

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;


  public static void main(String[] args) {
    Flour f = new Flour();
    SAXParser p = new SAXParser();
    p.setContentHandler(f);
    try { p.parse(args[0]); }
    catch (Exception e) {e.printStackTrace();}
    System.out.println(f.amount);
  }

public class Flour extends DefaultHandler {
  float amount = 0;
  public void startElement(String namespaceURI, String localName,
                           String qName, Attributes atts) {
    if (namespaceURI.equals("http://recipes.org") && localName.equals("ingredient")) {
       String n = atts.getValue("","name");
       if (n.equals("flour")) {
         String a = atts.getValue("","amount"); // assume 'amount' exists
         amount = amount + Float.valueOf(a).floatValue();
       }
    }
  }

}
```

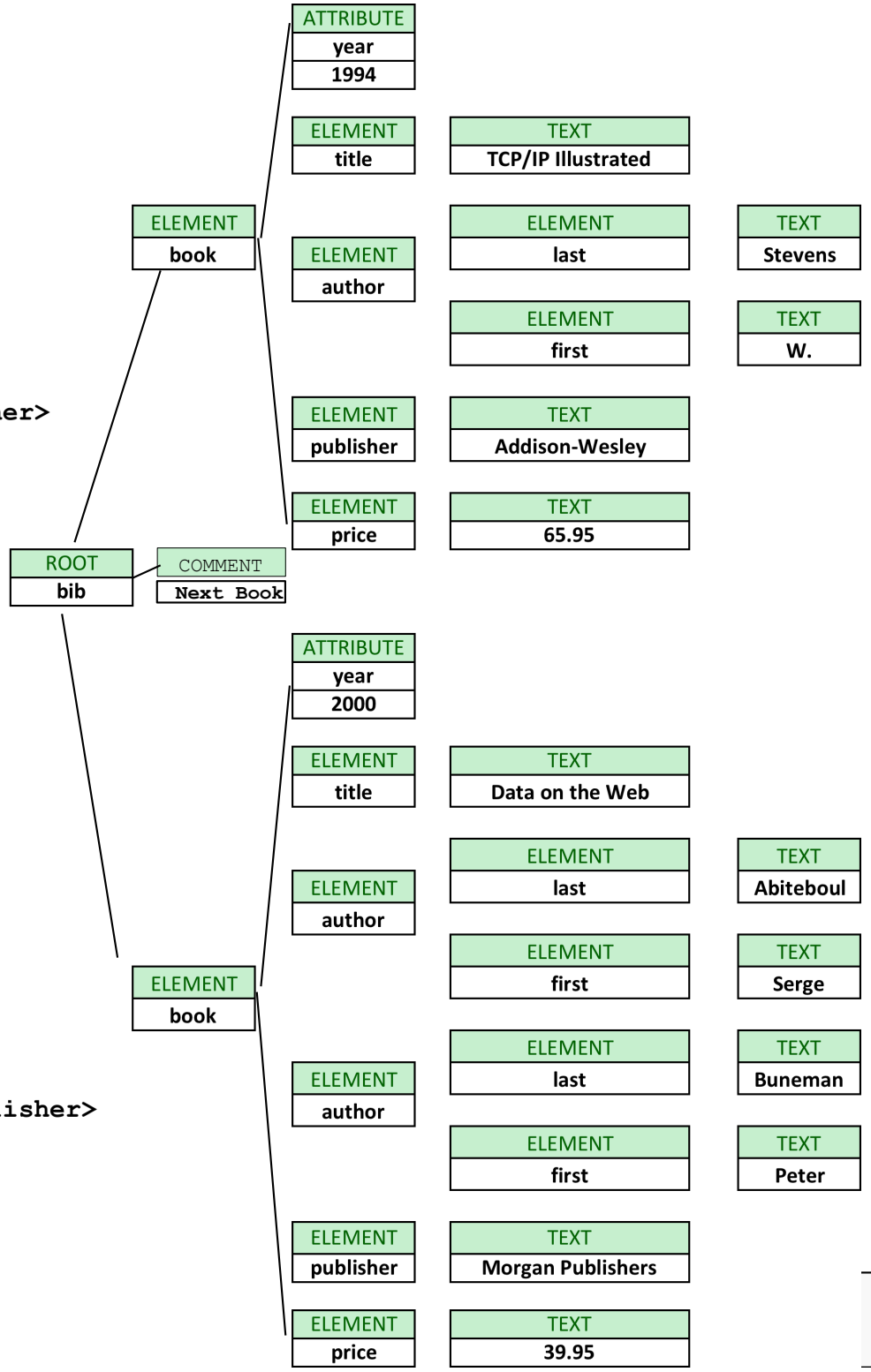
Exercise

- Create a XML Tree representation for the snippet of XML




```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
</book>
<!-- Next Book --!>
  <book year="2000">
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
<publisher>Morgan Publishers</publisher>
    <price>39.95</price>
  </book>

</bib>
```

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

  <book year="2000">
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <publisher>Morgan Publishers</publisher>
    <price>39.95</price>
  </book>

</bib>
```



Querying XML Documents

XPath

What is XPath?

- Addresses parts of an XML document
- W3C Recommendation
- Expression language
- Wildcards allowed
- Provides basic facilities for manipulation of strings, numbers and booleans
- Compact, non XML syntax for use within URIs and XML attribute values
- Operates on the abstract, logical structure of the XML document

Path Expression

- "Xpath, essentially specification of path for walking the XML tree"

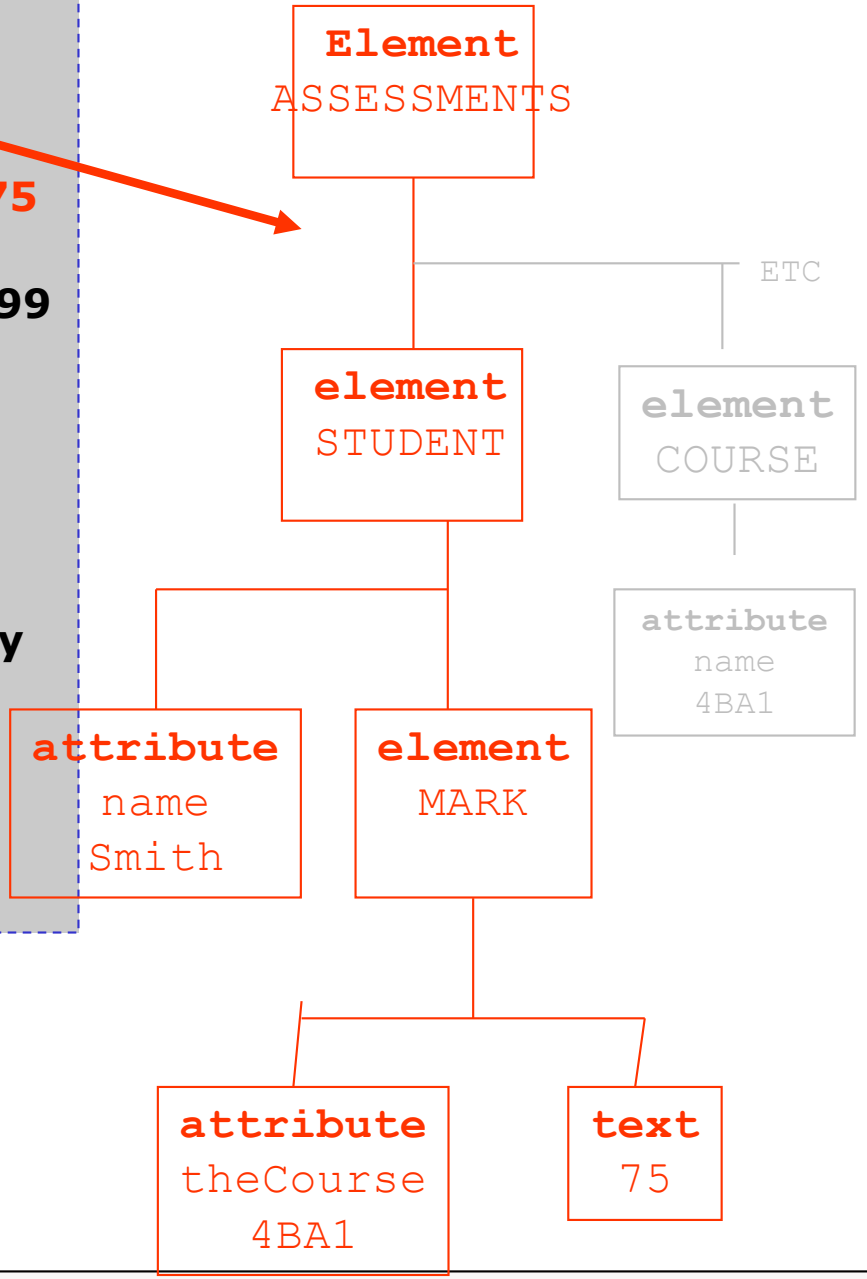
- Simple path expression is a sequence of *steps* separated by slashes (/)
  - More formally, "/" is a binary operator that applies the expression on its right-hand side to set of node selected by the expression on the left hand side
  - Informally, try to find a match for what is right of the slash in the tree/set of nodes returned by sequence of operations to the left of the slash

Example Document

```
<ASSESSMENTS>
        <STUDENT name = 'Smith'>
            <MARK theCourse = '4BA1'> 75
</MARK>
            <MARK theCourse = '4BA5'> 99
</MARK>
        </STUDENT> ...


        <COURSE name = '4BA1', takenBy
= 'Smith, Jones, ... '>
            <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

Describes mark for individual student

Describes average mark for course



Example: /ASSESSMENTS

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones, … ">
        </COURSE> …
</ASSESSMENTS>
```

Example: /ASSESSMENTS/STUDENT

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones, … ">
        </COURSE> …
</ASSESSMENTS>
```

Example: /ASSESSMENTS/STUDENT/MARK

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones, ... ">
        </COURSE> ...
</ASSESSMENTS>
```

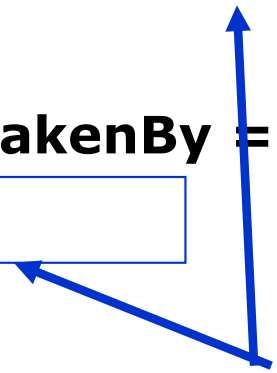
Describes the set with these two MARK element nodes as well as any other MARK elements nodes for any other STUDENT

Some Defaults

- By default trying to apply expression against any immediate child nodes in the left hand side set of nodes
- Sequence begins with //
  - Short hand trying to match any descendent nodes in the set of nodes

Example: //MARK

```
<ASSESSMENTS>
   <STUDENT name = "Smith">
        <MARK theCourse = "4BA1"> 75 </MARK>
        <MARK theCourse = "4BA5"> 99 </MARK>
   </STUDENT> ...

   <COURSE name = "4BA1", takenBy = "Smith, Jones, ... ">
        <MARK> 60 </MARK>
   </COURSE> ...
</ASSESSMENTS>
```



Still returns set of nodes from the document
with an element node named "MARK"
but this time not just those noted in
student assessment statements
e.g. a mark allocated to a course
by an external examiner

Example: /ASSESSMENTS

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                 <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

Example: /ASSESSMENTS//MARK

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

Getting just the text from any "mark" elements

Example: /ASSESSMENTS//MARK/string()

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
... ">
                <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

Getting just the text from any "mark" elements
Using the string() function

Wildcard *

- A asterix (*) put in place in a tag represents any one tag
- Example /*/*/MARK will return any MARK object appearing at the third level of nesting in the document

Attribute @

- Attributes are referred to by putting ampersand (@) before the name
- Appear in the path as if nested within the tag

Example:
/ASSESSMENTS/STUDENT/string(@name)

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
... ">
                 <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

Getting at an attribute value, string() function

Predicate Filters []

- A tag in a path that is followed by a condition [..] will ensure that only nodes that satisfy the condition are included in the resultant set

Example:

/ASSESSMENTS/STUDENT[MARK > 80]

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                 <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

Example:

/ASSESSMENTS/STUDENT[MARK > 80]

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
... ">
                <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

This set of nodes is returned as it satisfies the condition

Example Attribute in the selection:

/ASSESSMENTS/STUDENT/MARK[@theCourse = “4BA1"]

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

This set is returned as well
as any other student
MARK subtree sets of nodes
for 4BA1 elsewhere in doc

Example Attribute in the selection:

/ASSESSMENTS/STUDENT/MARK[@theCourse = "4BA1"]

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

This set of nodes is returned as well as any other student MARK subtree nodes for 4BA1 elsewhere

Over to you…

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- /database
- //surname
- /*/person[@age]
- /*/person/string(@age)

Over to you…

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- /database
- //surname
- /*/person[@age]
- /*/person/string(@age)

Over to you…

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- /database
- //surname
  - /*/person[@age]
- /*/person/string(@age)

Over to you…

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- /database
- //surname
- /*/person[@age]
- /*/person/string(@age)

Over to you…

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

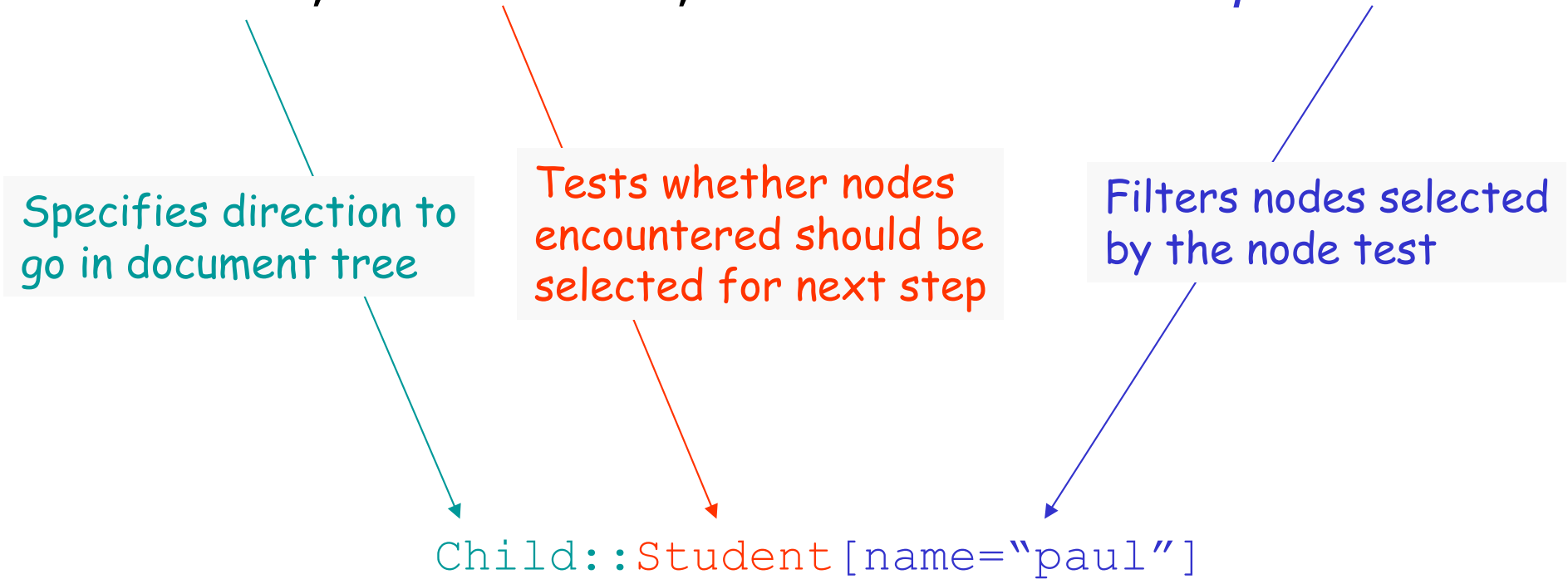
- /database
- //surname
- /*/person[@age]
- /*/person/string(@age)

Selecting Several steps

- By using the | operator in an XPath expression you can select several steps.
- //book/title | //book/price
  - Selects all the title together with price elements of all book elements
- //title | //price
  - Selects all the title together with price elements in the document
- //book/title | //price
  - Selects all the title elements of the book element together with all the price elements in the document

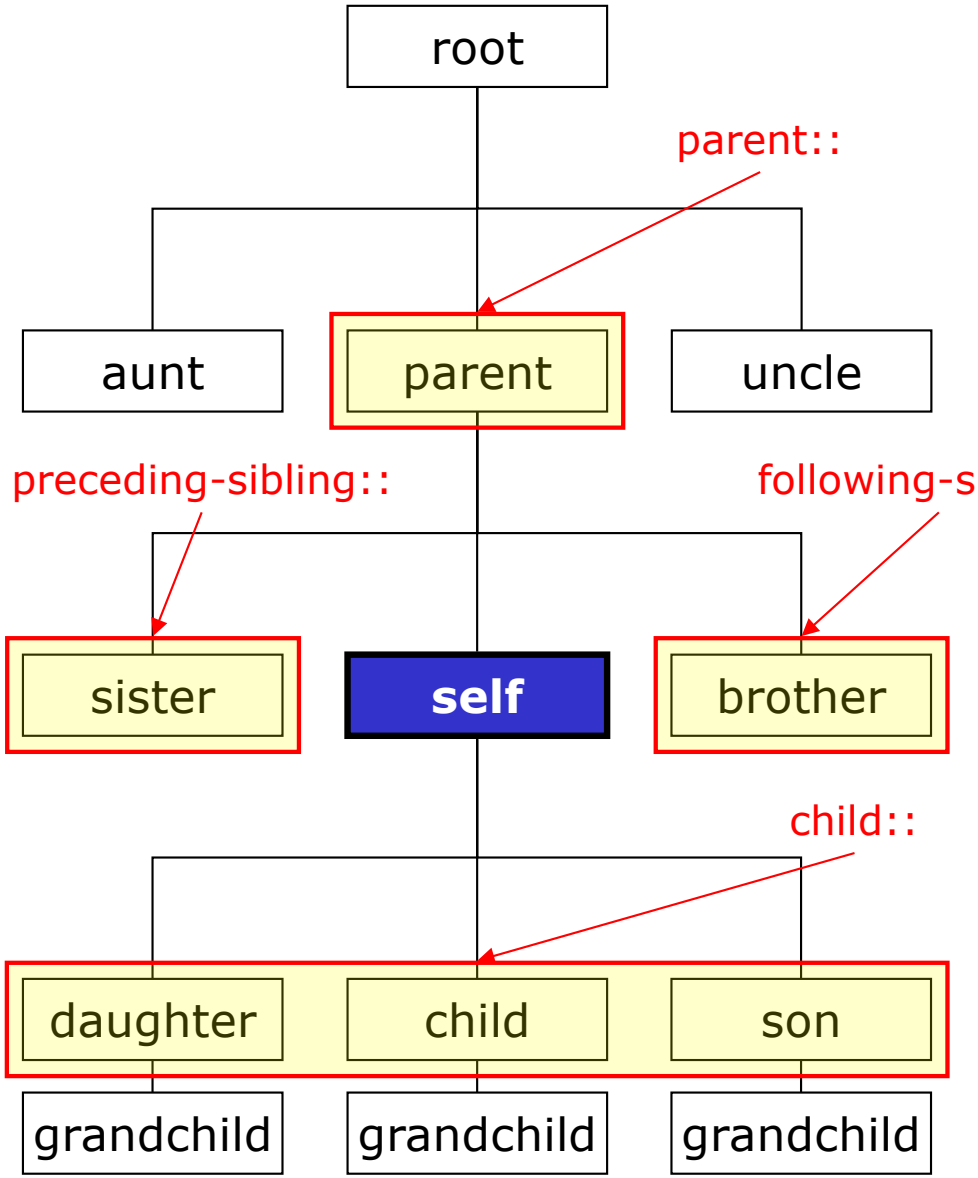
More Generally: Location Steps

- A step in an XPath expression consists of three parts: an *axis*, a *node* test, and zero or more *predicate* tests



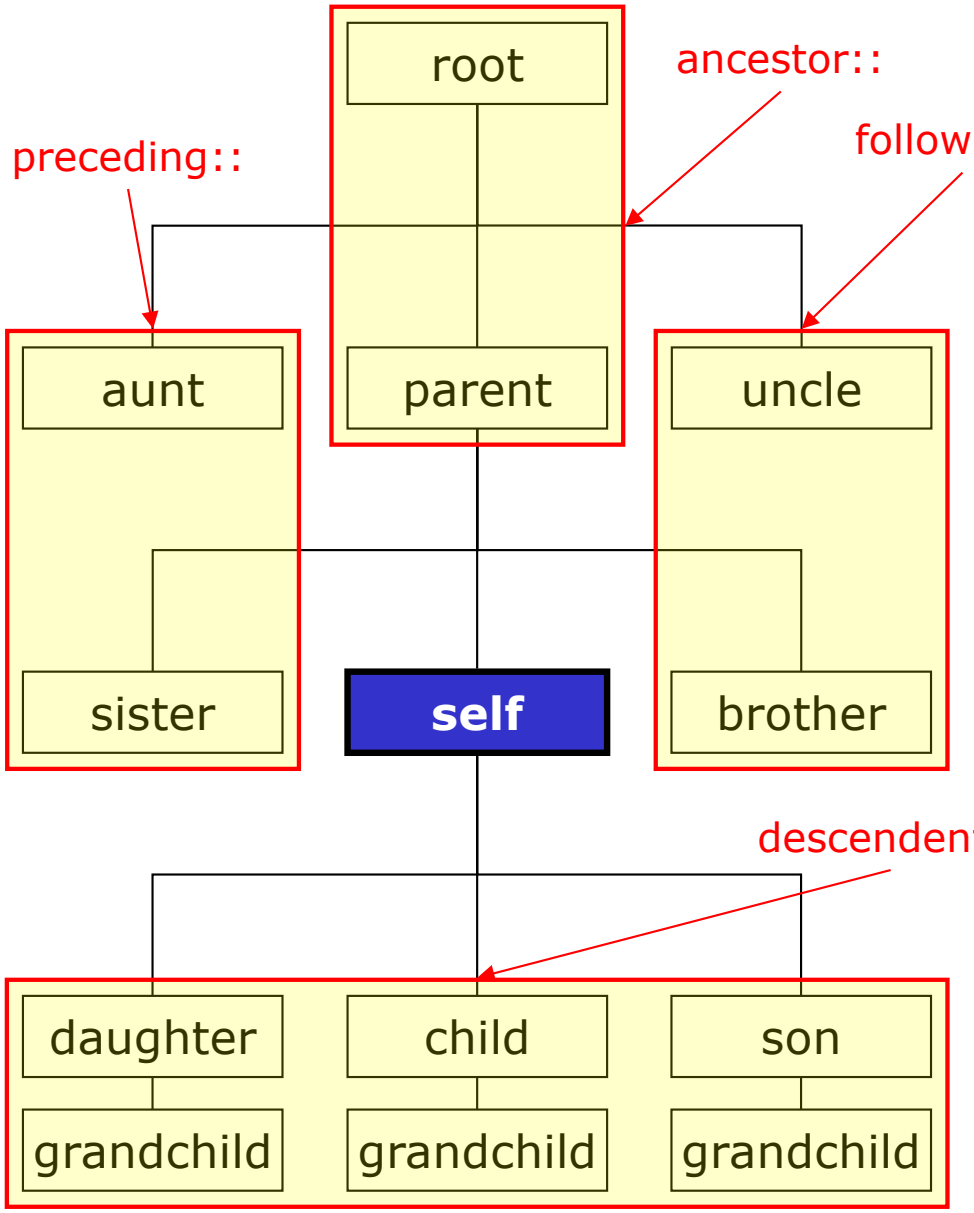
Axes spec (1)

There are several directions/axes we can traverse from a node




```
<?xml version='1.0' ?>

<root>
  <aunt />
  <parent>
    <sister />
    <self>
      <son>
        <grandchild />
      </son>
      <child />
      <daughter>
        <grandchild />
      </daughter>
    </self>
    <brother />
  </parent>
  <uncle></uncle>
</root>
```

Axes spec (2)




```
<?xml version='1.0' ?>
<root>
  <aunt />
  <parent>
    <sister />
    <self>
      <son>
        <grandchild />
      </son>
      <child />
      <daughter>
        <grandchild />
      </daughter>
    </self>
    <brother />
  </parent>
  <uncle></uncle>
</root>
```

Node tests

- The default is to test the node to see if it has an element name the same as that specified
  - E.g. child::Student would test if the child node has an element named “Student”
- Tests for checking element, attribute, and namespace name
- Tests for checking if the node is a text, comment, or processing instruction node
  - E.g. text()

Predicate Filters

- [] are used to hold predicates (conditions)
  - Attribute Tests
    - @ indicates attribute
  - Boolean Tests (Functions)
    - boolean, true, false, not, …
  - Node Set Tests (Functions)
    - count, id, position, last, …
  - Number Tests (Functions)
    - ceiling, floor, round, sum, …
  - String Tests (Functions)
    - concat, contains, string-length, substring, translate, …
  - Multiple Tests
    - Keywords (and, or), consecutive predicates [][]

Built-In Functions

- Accessor Functions
  - e.g. fn:node-name(*node) Returns the node-name of the argument node*
- Functions on Numeric Values
- Functions on Strings
- Functions on Durations, Dates and Times
- Functions on Nodes
- Functions on Sequences
- Aggregate Functions
- Context Functions

http://www.w3schools.com/xpath/xpath_functions.asp

XPath Operators

An XPath expression returns either a node-set, a string, a Boolean, or a number.

Operator	Description	Example	Return value
|	Computes two node-sets	//book | //cd	Returns a node-set with all book and cd elements
+	Addition	6 + 4	10
-	Subtraction	6 - 4	2
*	Multiplication	6 * 4	24
div	Division	8 div 4	2
=	Equal	price=9.80	true if price is 9.80 false if price is 9.90
!=	Not equal	price!=9.80	true if price is 9.90 false if price is 9.80
<	Less than	price<9.80	true if price is 9.00 false if price is 9.80
<=	Less than or equal to	price<=9.80	true if price is 9.00 false if price is 9.90
>	Greater than	price>9.80	true if price is 9.90 false if price is 9.80
>=	Greater than or equal to	price>=9.80	true if price is 9.90 false if price is 9.70
or	or	price=9.80 or price=9.70	true if price is 9.80 false if price is 9.50
and	and	price>9.00 and price<9.90	true if price is 9.80 false if price is 8.50
mod	Modulus (division remainder)	5 mod 2	1

Summary XPath example

```
<doc type="book" isbn="1-56592-796-9">
  <title>A Guide to XML</title>
  <author>Norman Walsh</author>
  <chapter>[...]</chapter>
  <chapter>
    <title>What Do XML Documents Look
      Like?</title>
    <paragraph>If you are [...]</paragraph>
    <paragraph>A few things [...]</paragraph>
    <ol>
      <item><paragraph>The document begins
         [...]</paragraph></item>
      <item><paragraph type="warning">There's
         no document [...]</paragraph></item>
      <item><paragraph>Empty elements have
         [...]</paragraph>
         <paragraph>In a very
           [...]</paragraph></item>
    </ol>
    <paragraph>XML documents are
      [...]</paragraph>
    <section>[...]</section>
    [...]
  </chapter>
</doc>
```

//paragraph

```
<paragraph>If you are [...]</paragraph>
<paragraph>A few things[...]</paragraph>
<paragraph>The document begins
  [...]</paragraph>
<paragraph type="warning">There's
   no document [...]</paragraph>
<paragraph>Empty elements have
   [...]</paragraph>
<paragraph>In a very [...]</paragraph>
<paragraph>XML documents are
   [...]</paragraph>
```

//ol//paragraph[@type='warning']

```
<paragraph type="warning">
  There's no document [...]
</paragraph>
```

/doc/chapter[2]/ol/item[position()=last()]

```
<item><paragraph>Empty elements have
  [...]</paragraph>
  <paragraph>In a very [...]</paragraph>
</item>
```

Design XPath queries for

2. Get all the titles of books in the file (without using //)

3. Get just the text from the first name elements of author

4. Return only the book elements that has an editor

5. Return only the books that are published after 1998

6. Return the entire book element whose title is "Data on the Web"

7. Alter the last query to just return the second author

8. Return those books which are priced between 50 and 100 only

9. Return all those books that are NOT published by Addison-Wesley

```
<?xml version="1.0" ?>
  <?xml version="1.0" ?>
  <bib>
     <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
     </book>

     <book year="1992">
        <title>Advanced Programming in the Unix
  environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
     </book>

     <book year="2000">
        <title>Data on the Web</title>
  <author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
     </book>

     <book year="1999">
        <title>The Economics of Technology and Content for
  Digital TV</title>
        <editor>
             <last>Gerbarg</last><first>Darcy</first>
              <affiliation>CITI</affiliation>
        </editor>
           <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
     </book>
```

Sample Solution

2. Get all the titles of books in the file (without using //)

```
/bib/book/title
```

3. Get just the text from the first name elements of author

```
//first/string()
```

4. Return only the book elements that has an editor

```
//book[editor]
```

5. Return only the books that are published after 1998

```
//book[@year>=1998]
```

6. Return the entire book element whose title is "Data on the Web”

```
 //book[title/string()="Data on the Web"]
```

7. Alter the last query to just return the second author

```
//book[title/string()="Data on the Web"]/author[2]
```

8. Return those books which are priced between 50 and 100 only

```
 //book[price>50][price<100]
```

9. Return all those books that are NOT published by Addison-Wesley

```
 //book[publisher!="Addison-Wesley"]
```

Summary

- Selects (a set of) nodes within an XML document based on
  - Conditions
  - Hierarchy
- Usage
  - Retrieving info from a single XML document
  - Applying XSL style sheet rules
  - Making XQuerys

Tutorial available at:
http://www.w3schools.com/xml/xpath_intro.asp

Querying XML Documents

XQuery

What is XQuery?

- Originally focused on **retrieval** of information from XML documents
  - **Update** features added in 2011
    https://www.w3.org/TR/xquery-update-10/
- XQuery is a language for finding and extracting elements and attributes from XML documents.
  - Here is an example of a question that XQuery could solve:
  - "Select all CD records with a price less than 10 euro from the CD collection stored in the XML document called cd_catalog.xml”
- Used in conjunction with XPath
- Latest version W3C recommendation
  “XQuery 3.0” – April 2014
  https://www.w3.org/TR/xquery-30/

For-Let-Where-OrderBy-Return:"FLWOR" expressions

(pronounced "FLOWER")

1. One or more FOR and/or LET expressions
   - For gathering nodes into sets from a series of XPath queries to operate upon in other clauses
2. Optional WHERE clause
   - For filtering nodes in the sets to be operated upon in other clauses
3. Optional ORDER BY clause
   - For returning nodes in the sets in particular order in other clauses
4. RETURN clause
   - How to return the identified nodes in the sets

LET Clause

- LET <variable> := <xpath expression>, <xpath expression>, …
  - Variable (starting with $) "binds to" **the set** returned by xpath expression
  - Does not iterate over set like the FOR clause does
  - It cannot be redefined within the scope of the function
  - More than one variable/path expression binding can be specified by separating with comma (,)

Example LET Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
       </mark>
      <mark thecourse="4BA1"> 75
       </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
let $c:=
doc("data/tcd.xml")/assessments/co
urse/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>
```

Curly brackets {} are used for enclosed expressions and indicate that the expression enclosed in the return clause needs to be evaluated by the Xquery processor

Result

```
<list_of_avg_course_marks>
     <mark>60</mark>
     <mark>70</mark>
</list_of_avg_course_marks>
```

FOR Clause

FOR <variable> IN <xpath expression>, <xpath expression>, <xpath expression>,…

- Variable (starting with $) "binds to" **in turn each member in the set** returned by Xpath expression(s)
- For each variable binding the rest of FLOWR expression is executed
- More than one variable/path expression binding can be specified by separating with comma (,)

Example FOR Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
       </mark>
      <mark thecourse="4BA1"> 75
       </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/co
urse
return
("Course Node:",$j)
```

Result

```
Course Node: <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
</course>
Course Node: <course name="4BA5"
takenby="Smith,Bond">
     <mark>70</mark>
</course>
```

Round Brackets useful for grouping sequence of Operations.

RETURN Clause

- One limitation of Xpath is that it can only operate on existing elements/attributes within the document
- XQuery allows the generation of new elements/attributes nodes
  - The element's content (if any) is either literally given between start- and end-tag, or provided as an “enclosed expression”, or as a mixture of both.
  - Curly brackets {} are used for enclosed expressions in the return clause and indicate that the expression enclosed needs to be evaluated by the Xquery processor

Example RETURN Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
       </mark>
      <mark thecourse="4BA1"> 75
       </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
        doc("data/tcd.xml")/assess
ments/course/@name
return
        <one_of_courses_is>
        {$j}
        </one_of_courses_is>
```

Example of Xquery node generation

Result

```
<one_of_courses_is name="4BA1"/>
<one_of_courses_is name="4BA5"/>
```

WHERE Clause

- Filters the binding tuples produced by the FOR and LET clauses
- If the filter expression evaluates to true then the RETURN clause is executed

Example WHERE Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
       </mark>
      <mark thecourse="4BA1"> 75
       </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/co
urse
where contains($j/@takenby,"Bond")
return
        <Bond_courses_is>
        {string($j/@name)}
        </Bond_courses_is>
```

Result

```
<Bond_courses_is>4BA5</Bond_courses_is>
```

Querying over several interlinked documents

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
details.xml

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
```

XQuery

```
for $w in
doc("data/details.xml")/studentdet
ails/student,
$x in
doc("data/tcd.xml")/assessments/st
udent
where $x/@name = $w/@name
return
<studentpercourse>
    {$w/@name}
    {$w/address}
    {$x/mark/@thecourse}
</studentpercourse>
```

Result

```
<studentpercourse name="Smith"
thecourse="4BA5" thecourse="4BA1">
    <address> 101 Pine </address>
</studentpercourse>
```

Over to you…

Source

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- Example syntax

```
let $c:=
doc("data/tcd.xml")/assessments/co
urse/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>

for $j in
        doc("data/tcd.xml")/assess
ments/course/@name
return
        <one_of_courses_is>
        {$j}
        </one_of_courses_is>
```

Define a query which will return an element called "paul_hobbys" which contains the hobby elements for each of person elements who have "Paul" as a firstname

Over to you…

Source

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

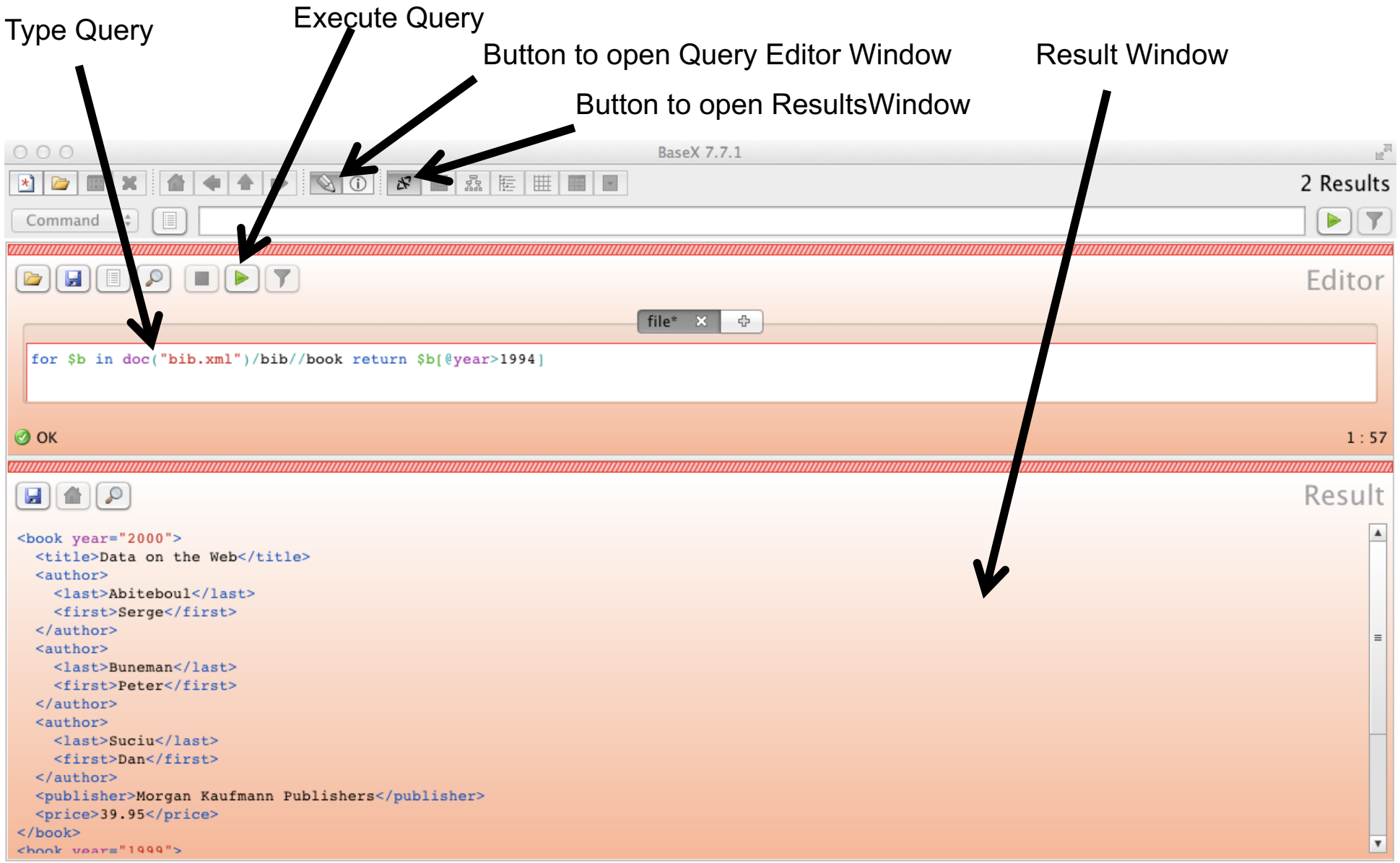
XQuery

```
for $p in
    doc("persondb.xml")/database/person
where $p/name/firstname=" Paul "
return
<paul_hobbys>
{$p/hobby}
</paul_hobbys>
```

Result

```
<paul_hobbys>
   <hobby> Football </hobby>
   <hobby> Racing </hobby>
</paul_hobbys>
```

BaseX: Xqueries upon XML files on Filesystem

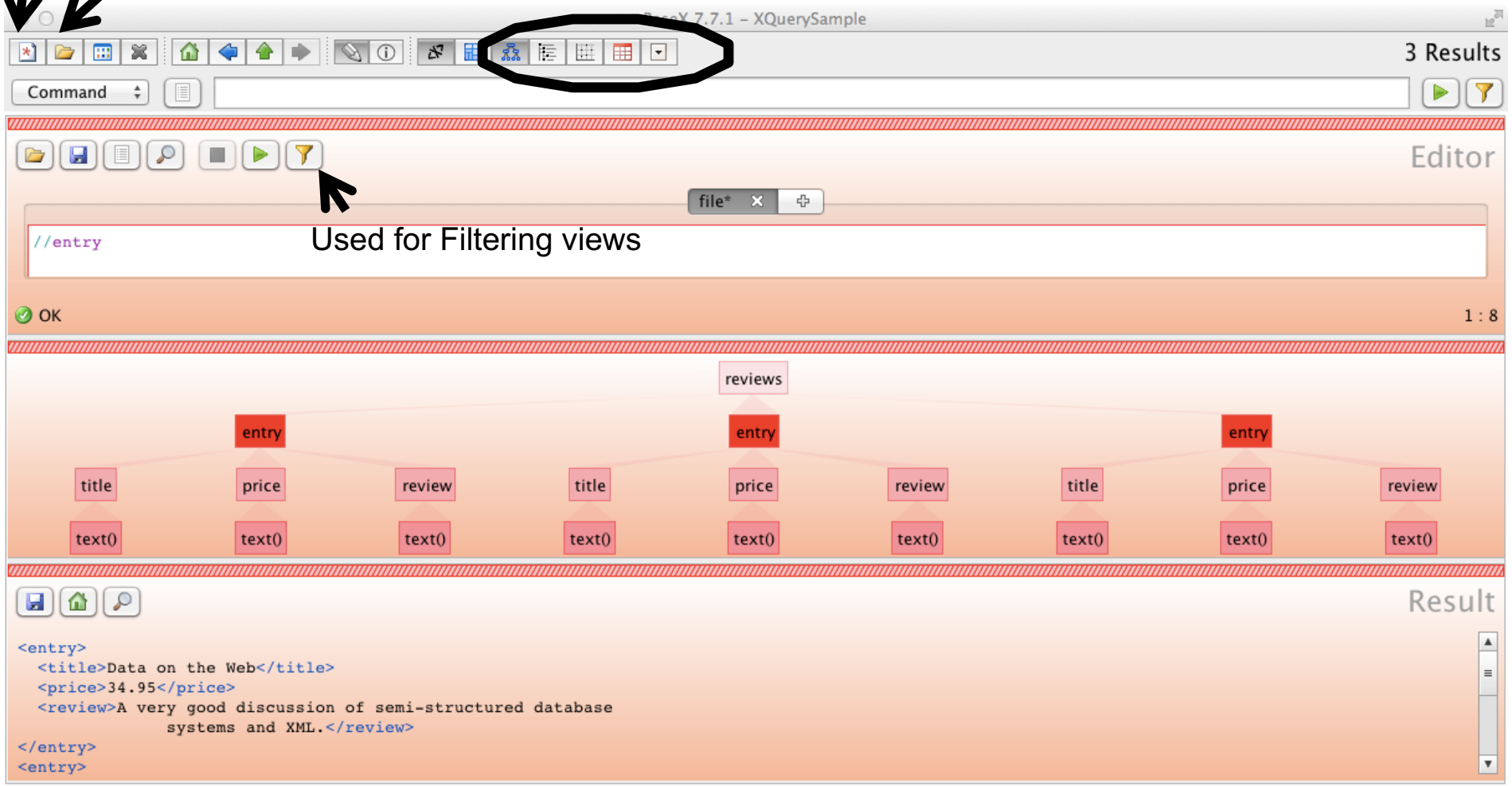


BaseX: Database functionality

Creating Database

Open Database
Load and Delete XML

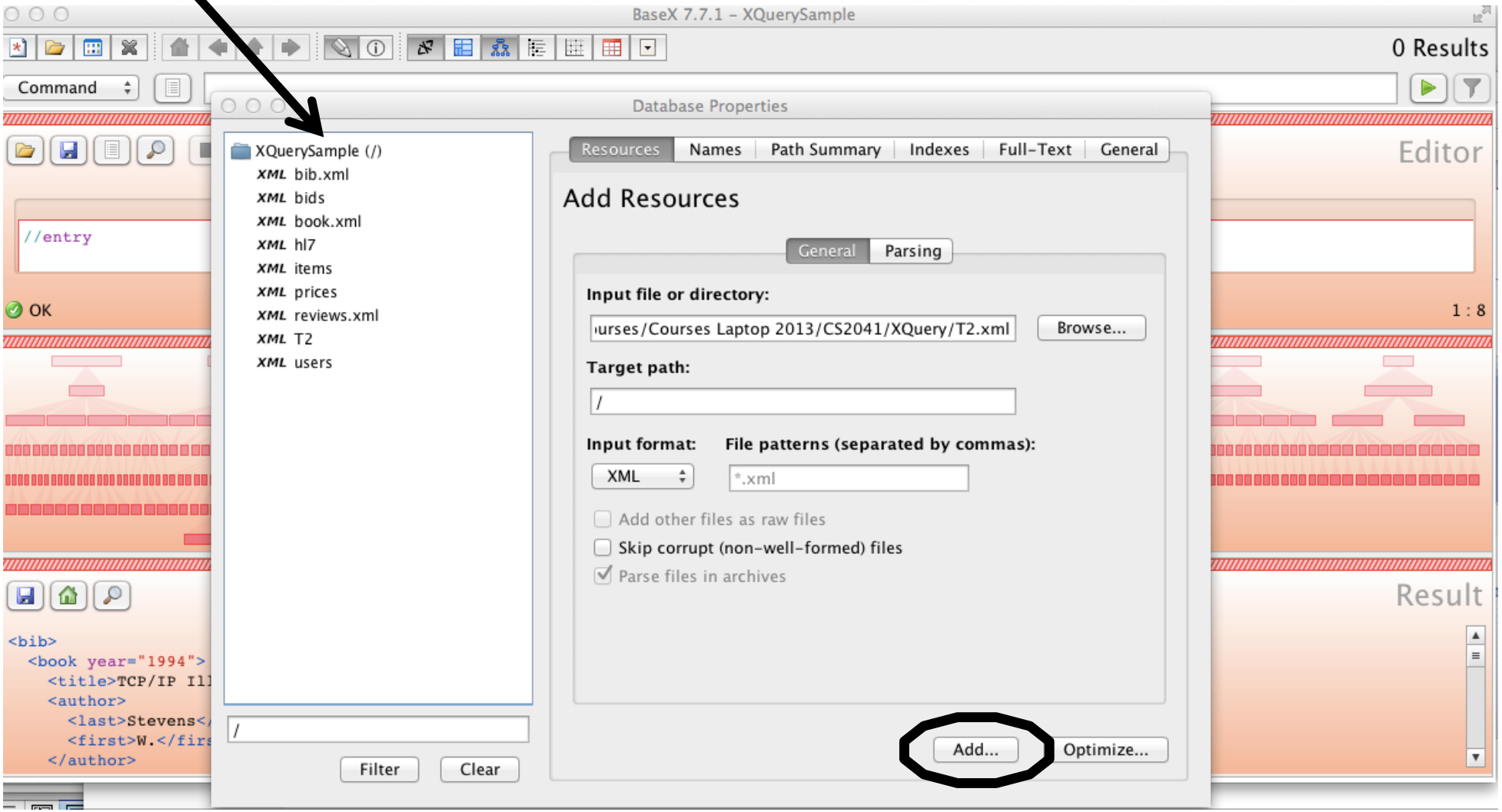
Buttons for Different Visualisations of XML
that is in the Database



Note that when XML is loaded in the database, you no longer need doc("filename")
in the XQuery

BaseX: Database functionality

XML files that were added to the Database using Add button below



Programmatic APIs

- Different programming APIs to connect to BaseX XML database
  - REST-Style Web API
  - Variety of Client APIs for different programming languages

- See http://docs.basex.org/wiki/Developing

Sorting

- The return clause of a FLWOR expression is evaluated once for each tuple in the tuple stream, and the results of these evaluations are concatenated to form the result of the FLWOR expression.
  - If no order by clause is present, the order of the tuple stream is determined by the orderings of the sequences returned by the expressions in the for clauses.
  - If an order by clause is present, it determines the order of the tuple stream

Example ORDER BY clause

XML Source

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2011 </enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming
</address>
    <enrolled> 2012 </enrolled>
  </student>
```

XQuery

```
for $x in
doc("data/details.xml")/studentdet
ails/student
order by $x/enrolled descending
return
  <byyear>
   {$x}
  </byyear>
```

Result

```
<byyear>
    <student name="Bond">
        <address> 007 Fleming
</address>
        <enrolled> 2012 </enrolled>
    </student>
</byyear>
<byyear>
    <student name="Smith">
        <address> 101 Pine </address>
        <enrolled> 2011 </enrolled>
    </student>
</byyear>
```

Sequence Operations

- A union of two node sequences is a sequence containing all the nodes that occur in either of the operands.
- The intersect operator produces a sequence containing all the nodes that occur in both of its operands.
- The except operator produces a sequence containing all the nodes that occur in its first operand but not in its second operand.

Example UNION clause

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
<recent_good_results>
{doc("data/tcd.xml")/assessments/s
tudent[mark > 80]
union
doc("data/tcd2.xml")/assessments/s
tudent[mark > 80]}
</recent_good_results>
```

Result

```
<recent_good_results>
    <student name="Smith">
        <mark thecourse="4BA5"> 99 </mark>
        <mark thecourse="4BA1"> 75 </mark>
    </student>
    <student name="ONeill">
        <mark thecourse="4BA5"> 85 </mark>
    </student>
</recent_good_results>
```

Conditional Clauses

- if (test expression) then expression else expression
- The result of a conditional expression depends on the value of the test expression in the if clause
  - If the value of the test expression is the Boolean value true, or a sequence containing at least one node (serving as an “existence test”), the then clause is executed.
  - If the value of the test expression is the Boolean value false or an empty sequence, the else clause is executed.
- **All three clauses (if, then, and else) are required**
- Nesting of if clauses possible

Example Conditional clause

XML Source

```
<?xml version="1.0"?>
<studentdetails>
 <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2012</enrolled>
  </student>
<student name="Bond">
    <address> 007 Fleming
</address>
    <enrolled> 2011 </enrolled>
  </student>
```

XQuery

```
for $x in
doc("data/details.xml")/studentdet
ails/student
return
 <status>
{$x/@name}
{if ($x/enrolled = 2012)
then "new student"
else if ($x/enrolled < 2008)
        then "should be finished"
else "student"}
</status>
```

Result

```
<status name="Smith">
student</status>
 <status name="Bond">
new student</status>
```

Quantified Expression

- Allows a variable to iterate over the items in a sequence.
  For each variable binding, a test expression is evaluated.
  - A quantified expression that begins with some keyword returns the value true if the test expression is true for some variable binding
  - A quantified expression that begins with every keyword, returns the value true if the test expression is true for every variable binding

Example Quantified Expression clauses

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
<results>
{if (every $m in
doc("data/tcd2.xml")/assessments/s
tudent satisfies $m/mark > 60)
then "excellent results"
else if (some $t in
doc("data/tcd2.xml")/assessments/s
tudent satisfies $t/mark > 50)
then "average results"
else "bad results"
}
</results>
```

Result

```
<results>average results</results>
```

Built-in Functions

- Over 100 XPath built-in functions (see last lecture)
- Functions that operate on Basic Types
  - manipulation of dates, strings, numbers etc
  - E.g. fn:string-join for joining strings together
- Functions that operate on Nodes
  - E.g. fn:name() returns name of node
- Functions that operate on Sequences
  - A sequence is an ordered collection of zero or more items
  - E.g. fn:distinct-values returns sequence with all duplicates removed
- Functions that operate on Context
  - obtain information from the evaluation context
  - E.g. fn:last returns the number of items in the sequence being processed

Useful summary of InBuilt functions provided by Oracle

Function	Commentary
Math: +, -, *, div, idiv, mod, =, !=, <, >, <=, >= floor(), ceiling(), round(), count(), min(), max(), avg(), sum()	Division is done using div rather than a slash because a slash indicates an XPath step expression. idiv is a special operator for integer-only division that returns an integer and ignores any remainder.
Strings and Regular Expressions: compare(), concat(), starts-with(), ends-with(), contains(), substring(), string-length(), substring-before(), substring-after(), normalize-space(), upper-case(), lower-case(), translate(), matches(), replace(), tokenize()	compare() dictates string ordering. translate() performs a special mapping of characters. matches(), replace(), and tokenize() use regular expressions to find, manipulate, and split string values.
Date and Time: current-date(), current-time(), current-dateTime() +, -, div eq, ne, lt, gt, le, gt	XQuery has many special types for date and time values such as duration, dateTime, date, and time. On most you can do arithmetic and comparison operators as if they were numeric. The two-letter abbreviations stand for equal, not equal, less than, greater than, less than or equal, and greater than or equal.
XML node and QNames: node-kind(), node-name(), base-uri() eq, ne, is, isnot, get-local-name-from-QName(), get-namespace-from-QName() deep-equal() >>, <<	node-kind() returns the type of a node (i.e. "element"). node-name() returns the QName of the node, if it exists. base-uri() returns the URI this node is from. Nodes and QName values can also be compared using eq and ne (for value comparison), or is and isnot (for identity comparison). deep-equal() compares two nodes based on their full recursive content. The << operator returns true if the left operand preceeds the right operand in document order. The >> operator is a following comparison.
Sequences: item-at(), index-of(), empty(), exists(), distinct-nodes(), distinct-values(), insert(), remove(), subsequence(), unordered().position(), last()	item-at() returns an item at a given position while index-of() attempts to find a position for a given item. empty() returns true if the sequence is empty and exists() returns true if it's not. dictinct-nodes() returns a sequence with exactly identical nodes removed and distinct-values() returns a sequence with any duplicate atomic values removed. unordered() allows the query engine to optimize without preserving order. position() returns the position of the context item currently being processed. last() returns the index of the last item.
Type Conversion: string(), data(), decimal(), boolean()	These functions return the node as the given type, where possible. data() returns the "typed value" of the node.
Booleans: true(), false(), not()	There's no "true" or "false" keywords in XQuery but rather true() and false() functions. not() returns the boolean negation of its argument.
Input: document(), input(), collection()	document() returns a document of nodes based on a URI parameter. collection() returns a collection based on a string parameter (perhaps multiple documents). input() returns s general engine-provided set of input nodes.

User Functions

- XQuery allows users to define functions of their own
  - A function may take zero or more parameters.
  - A function definition must specify the name of the function and the names of its parameters if they exist
    - It may optionally specify types for the parameters
      - If no type is specified for a function parameter, that parameter accepts values of any type.
    - It may optionally specify types for the result of the function.
      - If no type is specified for the result of the function, the function may return a value of any type.
  - Body of the function is an expression enclosed in curly braces.

Example Function clause

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2002.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:all_students()
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
return
<student>
         {$s/@name}
         {$s/mark/@thecourse}
</student>
   };

<all>
{local:all_students()}
</all>
```

Result

```
<all>
    <student name="Smith" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="Ledwidge" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="ONeill" thecourse="4BA5"/>
</all>
```

Type References

- Sometimes necessary to refer to a particular type in query
- One way to refer to a type is by its qualified name, or QName.
  - A QName may refer to a built-in type such as xs:integer or to a type that is defined in some schema, such as *abc:student*.
  - If the QName has a namespace prefix (the part to the left of the colon), that prefix must be bound to a specific namespace URI using the "declare namespace" clause

Example Function clause with param

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:
find_students($stuname as xs:string)
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
where $stuname = string($s/@name)
return
<student>
         {$s/@name}
         {$s/mark/@thecourse}
</student>
   };

local:find_students("Smith")
```

Result

```
<student name="Smith" thecourse="4BA5"
thecourse="4BA1"/>
```

Type References

- Another way to refer to a type is by a generic keyword such as element or attribute.
  - May optionally be followed by a QName that further restricts the name or type of the node.
  - For example,
    - element denotes any element;
    - element student denotes an element whose name is *student*;
    - element of type abc:student denotes an element whose type is *student* as declared in the namespace *abc*.
  - A reference to a type may optionally be followed by one of three occurrence indicators:
    - "*" means "zero or more";
    - "+" means "one or more,"
    - "?" means "zero or one."
    - The absence of an occurrence indicator denotes exactly one occurrence of the indicated type.

Example Function clause with output type declared

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
           takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
tcd2.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
      <mark thecourse="4BA5"> 45
</mark>
      <mark thecourse="4BA1"> 55
</mark>
  </student>
  <student name="ONeill">
      <mark thecourse="4BA5"> 85
</mark>  </student> </assessments>
```

XQuery

```
declare function local:all_students()
as element()*
   {
for $s in
doc("data/tcd.xml")/assessments/student
union
doc("data/tcd2.xml")/assessments/studen
t
return
<student>
         {$s/@name}
         {$s/mark/@thecourse}
</student>
   };

<all>
{local:all_students()}
</all>
```

Result

```
<all>
    <student name="Smith" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="Ledwidge" thecourse="4BA5"
thecourse="4BA1"/>
    <student name="ONeill" thecourse="4BA5"/>
</all>
```

Type References

- Not only occur in function definitions
- Can test for type using instanceof operator, e.g.
  - 49 instance of xs:integer returns true
  - "Hello" instance of xs:integer returns false
- Can convert result of an expression into one of XML schema built-in types using cast operator, e.g.
  - xs:double(11 div 5) returns 2.0
  - xs:string(11 div 5) returns “2”

Updating

- XQuery Update Facility (XQUF)
- All examples assume the data is imported into the XBase database
- High level declaration of manipulation of XML
- insert, delete, replace, rename, and transform experressions

Examples

- Insert node/nodes [after, before, after, into, last, first] *TargetExpression*

insert node <year>2005</year> after
/bib/books/book[1]/publisher

- Delete node/nodes

delete node /bib/book[1]/year

- Replace [value of] node *TargetExpression* with *TargetExpression*

Replace value of node /bib/book[1]/price
with /bib/book[1]/price * 1.1

- Rename node as *TargetExpression*

Rename node /bib/book[2]/author[1] as "principal-author"

Transform Expression

- Can be used to create modified copies of existing node
- Three clauses, denoted by the keywords copy, modify, and return
- Example

```
for $e in //employee[skill = "Java"]
return
   copy $je := $e
   modify delete node $je/salary
   return $je
```

Note that BaseX has a convenience operator called “update” to replace “copy, modify, return”. See http://docs.basex.org/wiki/XQuery_Update

XQuery and other XML technologies

- XQuery versus XPath
  - Operates over several documents
  - Allows the construction of new nodes
  - Has mechanism for variables
  - Has mechanism for user defined functions
- XQuery versus XSLT
  - Less verbose
  - Less document-centric
  - But does not have XML vocabulary for expressing
    - (although there is XQueryX vocabulary proposed but not high take up)

Summary

- Basis of XQuery is the FLOWR expression
  1. One or more FOR and/or LET expressions
  2. Optional WHERE clause
  3. Optional ORDER BY clause
  4. RETURN clause
- XQuery uses XPath in its clauses in order to identify individual parts of a document

References

- W3C site http://www.w3.org/XML/Query

- W3 Schools Tutorial

http://www.w3schools.com/xml/xquery_intro.asp

Over to you…

- NOW using **bib.xml (from earlier Xpath exercise), List books published by Addison-Wesley after 1991, including their year and title.**

You should get

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
   </book>
   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
   </book>
</bib>
```

- Example syntax

```
let $c:=
doc("data/tcd.xml")/assessments/course/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>

for $j in
        doc("data/tcd.xml")/assessments/course/@name
return
        <one_of_courses_is>
        {$j}
        </one_of_courses_is>
```

Over to you…

- NOW using **bib.xml (from earlier Xpath exercise), List books published by Addison-Wesley after 1991, including their year and title.**

You should get

```
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
   </book>
   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
   </book>
</bib>
```

- Sample Solution

```
<bib>
 {
  for $b in doc("bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }
</bib>
```

Review

XML Source
bib.xml

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</ti

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price> </book>

    <book year="1992">
        <title>Advanced Programming in the Unix
environment</title>

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price></book>

    <book year="2000">
        <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></auth
or>
<author><last>Buneman</last><first>Peter</first></author
>
<author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann
Publishers</publisher>
        <price>39.95</price>    </book>

    <book year="1999">
        <title>The Economics of Technology and Content
for Digital TV</title>
        <editor>
                <last>Gerbarg</last><first>Darcy</first>
                 <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic
Publishers</publisher>
        <price>129.95</price></book>
</bib>
```

XQuery

```
<bib>
 {
  for $b in
doc("http://www.bn.com/bib.xml")/b
ib/book
  where $b/publisher = "Addison-
Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }
</bib>
```

Result

```
<bib>
    <book year="1994">
        <title>TCP/IP
Illustrated</title>
    </book>
    <book year="1992">
        <title>Advanced Programming in
the Unix environment</title>
    </book>
</bib>
```

Review

XML Source
bib.xml

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</tit

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price> </book>

    <book year="1992">
        <title>Advanced Programming in the Unix
environment</title>

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price></book>

    <book year="2000">
        <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</first></auth
or>
<author><last>Buneman</last><first>Peter</first></author
>
<author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann
Publishers</publisher>
        <price>39.95</price>    </book>

    <book year="1999">
        <title>The Economics of Technology and Content
for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic
Publishers</publisher>
        <price>129.95</price></book>
</bib>
```

XML Source
reviews.xml

```
<reviews>
    <entry>
        <title>Data on the Web</title>
        <price>34.95</price>
        <review>
          A very good discussion of semi-structured database
              systems and XML.
        </review>
    </entry>
    <entry>
        <title>Advanced Programming in the Unix
environment</title>
        <price>65.95</price>
        <review>
          A clear and detailed discussion of UNIXprogramming.
        </review>
    </entry>
    <entry>
        <title>TCP/IP Illustrated</title>
        <price>65.95</price>
        <review>
              One of the best books on TCP/IP.
        </review>
    </entry>
</reviews>
```

Result

```
    <book-with-prices>
        <title>TCP/IP Illustrated</title>
        <price-amazon>65.95</price-amazon>
        <price-bn> 65.95</price-bn>
    </book-with-prices>
    <book-with-prices>
        <title>Advanced Programming in the Unix
environment</title>
        <price-amazon>65.95</price-amazon>
        <price-bn>65.95</price-bn>
    </book-with-prices>
    <book-with-prices>
        <title>Data on the Web</title>
        <price-amazon>34.95</price-amazon>
        <price-bn>39.95</price-bn>
    </book-with-prices>
```

Review

XML Source
bib.xml

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</tit

<author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price> 65.95</price> </book>

    <book year="1992">
        <title>Advanced Programming in the Unix
environment</title>

<author><last>Stevens</last><first>W.</first>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price></book>

    <book year="2000">
        <title>Data on the Web</title>

<author><last>Abiteboul</last><first>Serge</f
or>
<author><last>Buneman</last><first>Peter</fir
>
<author><last>Suciu</last><first>Dan</first><
        <publisher>Morgan Kaufmann
Publishers</publisher>
        <price>39.95</price>    </book>

    <book year="1999">
        <title>The Economics of Technology and Content
for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic
Publishers</publisher>
        <price>129.95</price></book>
</bib>
```

XML Source
reviews.xml

```
<reviews>
    <entry>
        <title>Data on the Web</title>
        <price>34.95</price>
        <review>
          A very good discussion of semi-structured database
              systems and XML.
        </review>
    </entry>
    <entry>
        <title>Advanced Programming in the Unix
environment</title>
```

XQuery

```
for $b in doc("http://www.bn.com/bib.xml")//book,
    $a in doc("http://www.amazon.com/reviews.xml")//entry
where $b/title = $a/title
 return
    <book-with-prices>
       { $b/title }
       <price-amazon>{ $a/price/text() }</price-amazon>
       <price-bn>{ $b/price/text() }</price-bn>
    </book-with-prices>
```

Result

```
    <book-with-prices>
        <title>TCP/IP Illustrated</title>
        <price-amazon>65.95</price-amazon>
        <price-bn> 65.95</price-bn>
    </book-with-prices>
    <book-with-prices>
        <title>Advanced Programming in the Unix
environment</title>
        <price-amazon>65.95</price-amazon>
        <price-bn>65.95</price-bn>
    </book-with-prices>
    <book-with-prices>
        <title>Data on the Web</title>
        <price-amazon>34.95</price-amazon>
        <price-bn>39.95</price-bn>
    </book-with-prices>
```