XML, XPATH, and XQUERY

Dr. Rabah Mokhtari

Department of Computer Science,
UNIVERSITY OF M'SILA

November 26, 2022

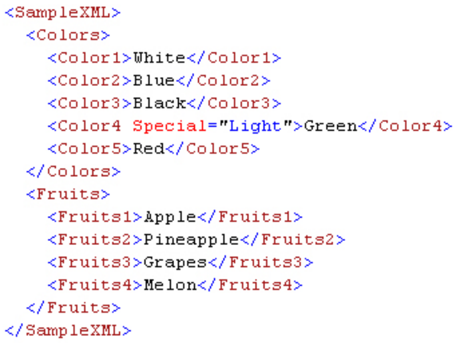
Content

Introduction XML language

Introduction

Definition

- XML stands for eXtensible Markup Language.
- XML was designed to store and transport data.
- XML is often used for distributing data over the Internet.
- XML was designed to be both human- and machine-readable.



```
<SampleXML>
  <Colors>
    <Color1>White</Color1>
    <Color2>Blue</Color2>
    <Color3>Black</Color3>
    <Color4 Special="Light">Green</Color4>
    <Color5>Red</Color5>
  </Colors>
  <Fruits>
    <Fruits1>Apple</Fruits1>
    <Fruits2>Pineapple</Fruits2>
    <Fruits3>Grapes</Fruits3>
    <Fruits4>Melon</Fruits4>
  </Fruits>
</SampleXML>
```

XML Standards

some XML standards

- XML AJAX (Asynchronous JavaScript And XML)
- XML DOM (Document Object Model)
- XML XPath (XML Path Language)
- XML XSLT (eXtensible Stylesheet Language)
- XML XQuery (XML Query Language)
- XML DTD (Document Type Definition)
- XML Schema
- ...

XML how to use?

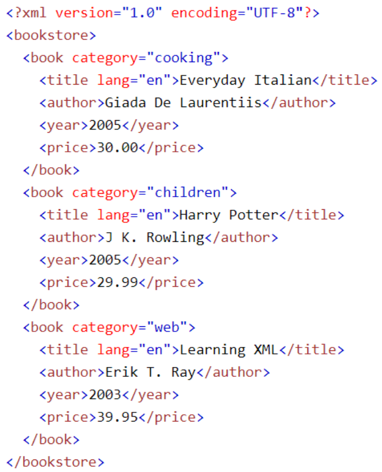
XML Separates Data from Presentation

- XML does not carry any information about how to be displayed.
- The same XML data can be used in many different presentation scenarios.
- Because of this, with XML, there is a full separation between data and presentation.

XML separates data from HTML

- using XML, the data can be stored in separate XML files.
- JavaScript codes can read an XML file and update the data content of any HTML page.

XML Tree Structure



```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Figure: book store

XML Tree Structure

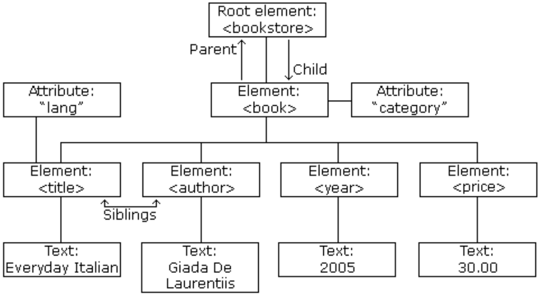
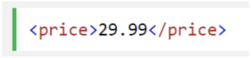


Figure: book store

XML Element

What's XML element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.



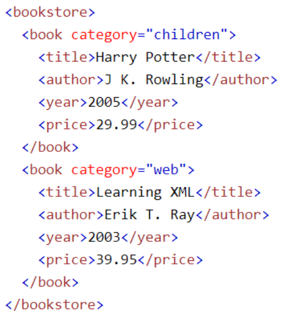
```xml
<price>29.99</price>
```

Figure: XML element

XML element content

An element can contain:

- text
- attributes
- other elements
- or a mix of the above



```xml
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

How to get information from XML documents

XPath and XQuery languages

XPath and XQuery is a query language designed by the W3C to address the needs to navigate through xml data and to get information.

XPath

XPath can be used to navigate through elements and attributes in an XML document.

XQuery

XQuery is a language allowing you to select the XML data elements of interest, reorganize and possibly transform them, and return the results in a structure of your choosing.

XPath

XPath language

Definition

- XPath is a W3C recommendation.
- XPath is a language for addressing parts of XML documents.
- XPath is a sort of query language.
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath is a major element in XSLT and in XQuery

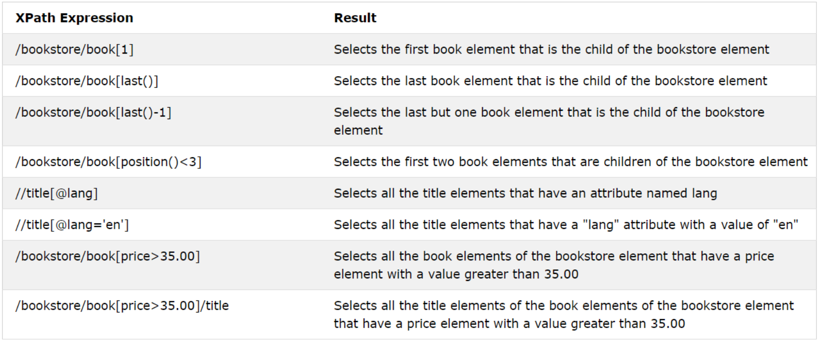
XPath versions

- There are several versions of XPath in use. XPath 1.0 was published in 1999, XPath 2.0 in 2007 (with a second edition in 2010), XPath 3.0 in 2014, and XPath 3.1 in 2017.
- However, XPath 1.0 is still the version that is most widely available.

XPath Language

XPath Expression

in the following, we listed some XPath expressions and the result of the expressions:



XPath Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Figure: XPath Expressions

XSLT

XSLT

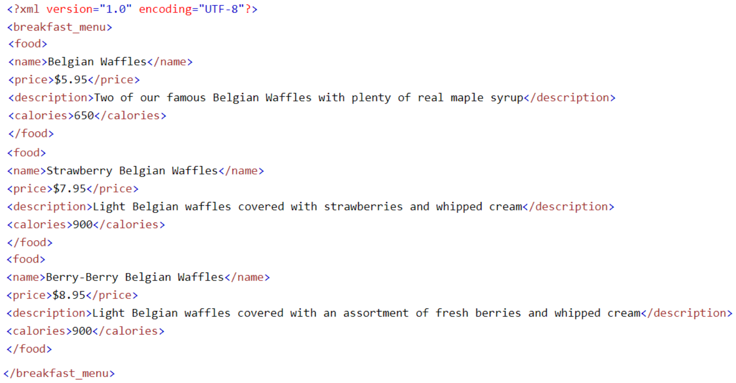
Presentation

- XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
- With XSLT you can
  - Add/remove elements and attributes to or from the output file.
  - Rearrange and sort elements.
  - perform tests and make decisions about which elements to hide and display.
  - ...
- XSLT uses XPath to find information in an XML document.

XSLT example

display xml data in html

The data of the following XML document will be used to be displayed in another html file.



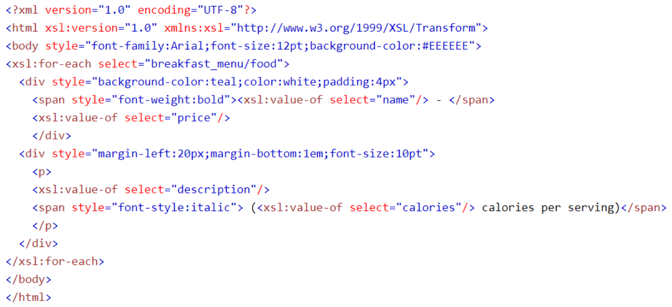
```
<?xml version="1.0" encoding="UTF-8"?>
<breakfast_menu>
<food>
<name>Belgian Waffles</name>
<price>$5.95</price>
<description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
<calories>650</calories>
</food>
<food>
<name>Strawberry Belgian Waffles</name>
<price>$7.95</price>
<description>Light Belgian waffles covered with strawberries and whipped cream</description>
<calories>900</calories>
</food>
<food>
<name>Berry-Berry Belgian Waffles</name>
<price>$8.95</price>
<description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
<calories>900</calories>
</food>
</breakfast_menu>
```

Figure: break fast menu xml document

XSLT example

display xml data in html

We use XSLT to extract data value from the break fast menu xml file to be displayed in the following HTML file.



```
<?xml version="1.0" encoding="UTF-8"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<body style="font-family:Arial;font-size:12pt;background-color:#EEEEEE">
<xsl:for-each select="breakfast_menu/food">
  <div style="background-color:teal;color:white;padding:4px">
    <span style="font-weight:bold"><xsl:value-of select="name"/> - </span>
    <xsl:value-of select="price"/>
    </div>
  <div style="margin-left:20px;margin-bottom:1em;font-size:10pt">
    <p>
    <xsl:value-of select="description"/>
    <span style="font-style:italic"> (<xsl:value-of select="calories"/> calories per serving)</span>
    </p>
  </div>
</xsl:for-each>
</body>
</html>
```

Figure: Add break fast menu data in an HTML file

XQuery

Presentation

- XQuery is a query language designed by the W3C to address XML databases.
- XQuery uses a set of clauses similar to that used by SQL language in Relational DB.
- The basic structure of many (but not all) XQuery queries is the FLWOR (pronounced “flower”) expression (**F**or, **L**et, **W**here, **O**derd by, and **R**eturn.

Capabilities of XQuery

XQuery has a rich set of features that allow many different types of operations on XML data and documents:

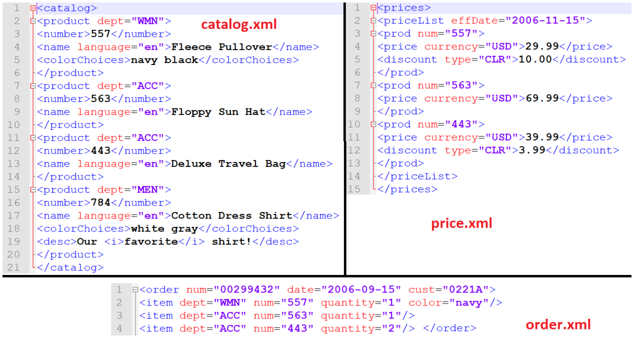
- Selecting information based on specific criteria
- Filtering out unwanted information
- Joining data from multiple documents or collections of documents
- Sorting, grouping, and aggregating data
- Transforming and restructuring XML data into another XML vocabulary or structure
- Performing arithmetic calculations on numbers and dates

Example

Example

The following figure represents an XML database consisting in three documents

- **catalog.xml** contains list of products.
- **order.xml** contains list of order made on the catalog's product list
- **price.xml** contains the prices of the catalog's product list



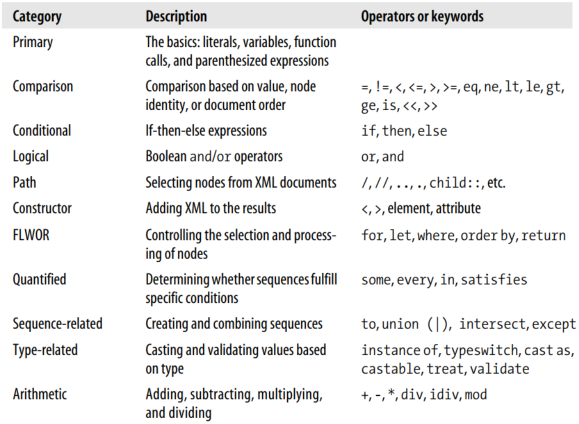
```xml
<catalog>
<product dept="WMN">
<number>557</number>
<name language="en">Fleece Pullover</name>
<colorChoices>navy black</colorChoices>
</product>
<product dept="ACC">
<number>563</number>
<name language="en">Floppy Sun Hat</name>
</product>
<product dept="ACC">
<number>443</number>
<name language="en">Deluxe Travel Bag</name>
</product>
<product dept="MEN">
<number>784</number>
<name language="en">Cotton Dress Shirt</name>
<colorChoices>white gray</colorChoices>
<desc>Our <i>favorite</i> shirt!</desc>
</product>
</catalog>
```

```xml
<prices>
<priceList effDate="2006-11-15">
<prod num="557">
<price currency="USD">29.99</price>
<discount type="CLR">10.00</discount>
</prod>
<prod num="563">
<price currency="USD">69.99</price>
</prod>
<prod num="443">
<price currency="USD">39.99</price>
<discount type="CLR">3.99</discount>
</prod>
</priceList>
</prices>
```

```xml
<order num="00299432" date="2006-09-15" cust="0221A">
<item dept="WMN" num="557" quantity="1" color="navy"/>
<item dept="ACC" num="563" quantity="1"/>
<item dept="ACC" num="443" quantity="2"/> </order>
```

Figure: Catalog's product list XML database

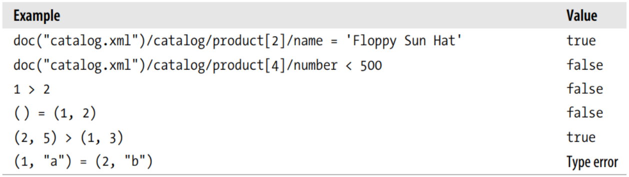
Categories of Expressions



Category	Description	Operators or keywords
Primary	The basics: literals, variables, function calls, and parenthesized expressions	
Comparison	Comparison based on value, node identity, or document order	=, !=, <, <=, >, >=, eq, ne, lt, le, gt, ge, is, <<, >>
Conditional	If-then-else expressions	if, then, else
Logical	Boolean and/or operators	or, and
Path	Selecting nodes from XML documents	/, //, .., ., child::, etc.
Constructor	Adding XML to the results	<, >, element, attribute
FLWOR	Controlling the selection and processing of nodes	for, let, where, order by, return
Quantified	Determining whether sequences fulfill specific conditions	some, every, in, satisfies
Sequence-related	Creating and combining sequences	to, union (|), intersect, except
Type-related	Casting and validating values based on type	instance of, typeswitch, cast as, castable, treat, validate
Arithmetic	Adding, subtracting, multiplying, and dividing	+, -, *, div, idiv, mod

Figure: Categories of expressions

Examples



Example	Value
`doc("catalog.xml")/catalog/product[2]/name = 'Floppy Sun Hat'`	true
`doc("catalog.xml")/catalog/product[4]/number < 500`	false
`1 > 2`	false
`() = (1, 2)`	false
`(2, 5) > (1, 3)`	true
`(1, "a") = (2, "b")`	Type error

Figure: General comparison

General comparisons on multi-item sequences

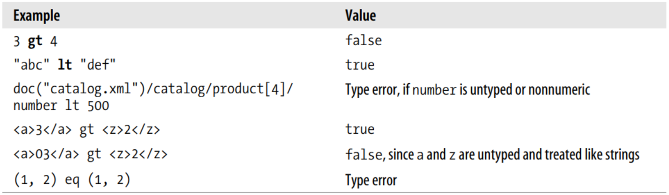
The expression (2, 5) < (1, 3) returns true if one or more of the following conditions is true:

- 2 is less than 1
- 2 is less than 3
- 5 is less than 1
- 5 is less than 3

Examples

Value Comparisons

Value comparisons differ fundamentally from general comparisons in that they can only operate on single atomic values.



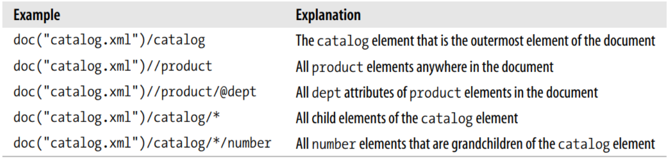
Example	Value
`3 gt 4`	`false`
`"abc" lt "def"`	`true`
`doc("catalog.xml")/catalog/product[4]/number lt 500`	Type error, if `number` is untyped or nonnumeric
`<a>3</a> gt <z>2</z>`	`true`
`<a>03</a> gt <z>2</z>`	`false`, since `a` and `z` are untyped and treated like strings
`(1, 2) eq (1, 2)`	Type error

Figure: value comparison

XPath Expressions

XPath steps

A path expression is made up of one or more steps that are separated by a slash (/) or double slashes (//). For example, the path:



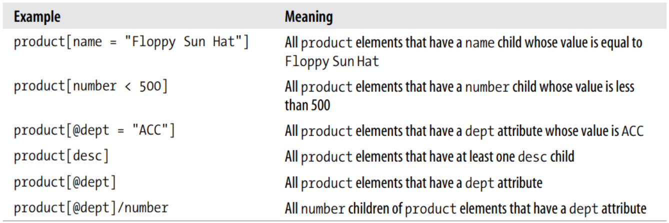
Example	Explanation
`doc("catalog.xml")/catalog`	The `catalog` element that is the outermost element of the document
`doc("catalog.xml")//product`	All `product` elements anywhere in the document
`doc("catalog.xml")//product/@dept`	All `dept` attributes of `product` elements in the document
`doc("catalog.xml")/catalog/*`	All child elements of the `catalog` element
`doc("catalog.xml")/catalog/*/number`	All `number` elements that are grandchildren of the `catalog` element

Figure: Xpath expression

Predicates

XPath steps

- Predicates are used in a path expression to filter the results to contain only nodes that meet specific criteria.
- Using a predicate, you can, for example, select only the elements that have a certain value for an attribute or child element, using a predicate like **[@dept = "ACC"]**.



Example	Meaning
`product[name = "Floppy Sun Hat"]`	All `product` elements that have a `name` child whose value is equal to `Floppy Sun Hat`
`product[number < 500]`	All `product` elements that have a `number` child whose value is less than 500
`product[@dept = "ACC"]`	All `product` elements that have a `dept` attribute whose value is `ACC`
`product[desc]`	All `product` elements that have at least one `desc` child
`product[@dept]`	All `product` elements that have a `dept` attribute
`product[@dept]/number`	All `number` children of `product` elements that have a `dept` attribute

Figure: Filtering using predicates

The XQuery Data Model

XDM

- Understanding the XQuery data model is analogous to understanding tables, columns, and rows when learning SQL.
- The basic components of XDM are:

- **Node:** An XML construct such as an element or attribute
- **Atomic value:** A simple data value with no markup associated with it.
- **Item:** A generic term that refers to either a node or an atomic value
- **Sequence:** An ordered list of zero, one, or more items.

The XQuery Data Model

The relationship among these components is depicted in the following diagram.

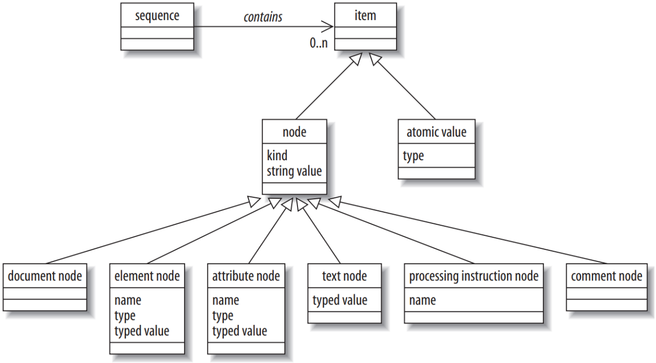
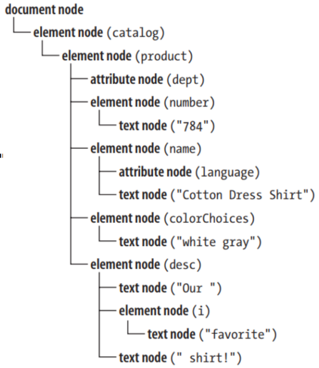


Figure: XQuery Data Model

The XQuery Data Model

Example

```
<catalog xmlns="http://datypic.com/cat">
    <product dept="MEN" xmlns="http://datypic.com/prod"
        <number>784</number>
        <name language="en">Cotton Dress Shirt</name>
        <colorChoices>white gray</colorChoices>
        <desc>Our <i>favorite</i> shirt!</desc>
    </product>
</catalog>
```



Values of nodes

String and typed values of nodes

There are two kinds of values for a node: **string** and **typed**.

- The string value of an element node is its character data content and that of all its descendant elements concatenated together.
- The string value of a node can be accessed using the **string** function.
- **Example 1:** string(doc("catalog.xml")/catalog/product[4]/number).
- **Example 2:** string(<desc>Our <i>favorite</i> shirt!</desc>).

Examples

- **Example 1:** string(doc("catalog.xml")/catalog/product[4]/number)
  Returns: 784
- **Example 2:** string(<desc>Our <i>favorite</i> shirt!</desc>)
  Returns: Our favorite shirt!

Values of nodes

Typed values of nodes

There are two kinds of values for a node: **string** and **typed**.

- The XQuery type system is based on that of XML Schema.
- XML Schema has built-in simple types representing common datatypes such as **xs:integer**, **xs:string**, and **xs:date**.
- An element or attribute might have a particular type if it has been validated with a *schema*.
- The typed value of a node can be accessed using the **data** function.

Examples

1. *data(doc("catalog.xml")/catalog/product[4]/number)*
2. it returns the integer 784, if the number element is declared in a schema to be an **integer**.
3. If it is not declared in the schema, its typed value is still 784, but the value is considered to be **untyped**.

Literals, Variables, and sequences

Literals

- Literals are simply constant values that are directly represented in a query, such as "ACC" and 29.99.
- There are two kinds of literals: *string literals* and *numeric literals*.
- They can be used in expressions anywhere a constant value is needed.

Example

The strings in the comparison expression:
if ($department = "ACC") then "accessories" else "other"

Literals, Variables, and sequences

Variables

Variables in XQuery are identified by names that are preceded by a dollar sign **($).***

Example

Declare an initialized variable **$num := 12;**

Sequences

- Sequences are ordered collections of items.
- A sequence can contain zero, one, or many items.

Example

Declare an initialized variable **$num := 12;**

FLWOR Expressions

Example 1

The following example returns ordered product names

```
for $prod in doc("catalog.xml")/catalog/product
where $prod/@dept = "ACC"
order by $prod/name
return $prod/name
```

Example 2

Wrapping results in a new element

```
<ul>{
  for $product in doc("catalog.xml")/catalog/product
  where $product/@dept='ACC'
  order by $product/name
  return $product/name
}</ul>
```

FLWOR Expressions

Example 3

Adding attributes to results

```
<ul type="square">{
  for $product in doc("catalog.xml")/catalog/product
  where $product/@dept='ACC'
  order by $product/name
  return <li class="{$product/@dept}">{data($product/name)}</li>
}</ul>
```

Joining

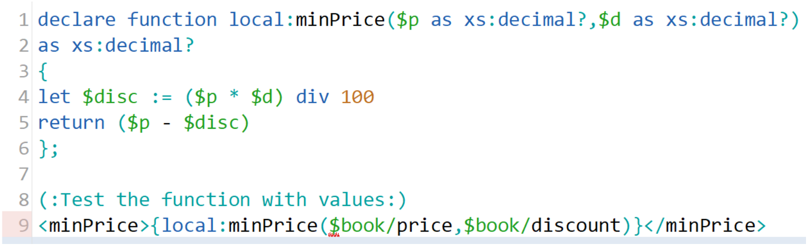
Example 4

Joining multiple input documents

Function and variables

Example 5

Example of a User-defined Function declared in the XQuery



```
declare function local:minPrice($p as xs:decimal?,$d as xs:decimal?)
as xs:decimal?
{
let $disc := ($p * $d) div 100
return ($p - $disc)
};

(:Test the function with values:)
<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>
```