

# eXtensible Markup Language (XML)

Avinash Maskey

# Introduction (1) – What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to carry data, not to display data
- XML tags are not predefined. You must define your own tags
- XML is designed to be self-descriptive
- XML typically have “.xml” file extension
- XML declaration is optional, it refers to version and encoding of xml document
- XML is both human and machine readable
- XML is a software and hardware independent way of storing, transporting and sharing data.
- XML is a W3C Recommendation

# Introduction (2) – What is XML?

- XML is not a replacement for HTML.
- **XML and HTML were designed with different goals:**
  - XML was designed to **transport and store data**, with focus on **what data is**
  - HTML was designed to **display data**, with focus on **how data looks**
  - HTML is about displaying information, while XML is about carrying information.
- XML should contain **XML prolog**, **root element** and **tags**.

# Introduction (3) – How can XML be used?

- XML Separates Data from HTML
- XML Simplifies Data sharing
- XML simplifies Data Transport
- XML simplifies Platform Changes
- XML makes your data more available
- XML is used to create new internet language

# Introduction (4) – HTML and XML Examples

- **An HTML Example**

```
<h2>Nonmonotonic Reasoning: Context-Dependent Reasoning</h2>
<i>by <b>V. Marek</b> and <b>M. Truszczyński</b></i><br>
Springer 1993<br>
ISBN 0387976892
```

- **The Same Example in XML**

```
<book>
  <title>Nonmonotonic Reasoning: Context- Dependent Reasoning</title>
  <author>V. Marek</author>
  <author>M. Truszczyński</author>
  <publisher>Springer</publisher>
  <year>1993</year>
  <isbn>0387976892</isbn>
</book>
```

# Introduction (5) – HTML and XML Examples

- **In HTML**

```
<h2>Relationship force-mass</h2>  
<i> F = M × a </i>
```

- **In XML**

```
<equation>  
  <meaning>Relationship force-mass</meaning>  
  <leftside> F </leftside>  
  <rightside> M × a </rightside>  
</equation>
```

# Introduction (6) – XML Syntax Rules

- The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.
- **The XML Prolog (Syntax)**  
`<?xml version="1.0" encoding="UTF-8"?>`
  - **The XML prolog is optional.** If it exists, it must come first in the document.
  - XML documents can contain international characters, like Norwegian øæå or French èéé.
  - To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.
  - UTF-8 is the default character encoding for XML documents.

# Introduction (7) – XML Syntax Rules

- **XML Documents Must Have a Root Element**
  - XML documents must contain one root element that is the parent of all other elements.

- **Syntax:**

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

# Introduction (8) – XML Syntax Rules

- In this example `<note>` is the root element:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
    <to>Harry</to>
    <from>Marry</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</note>
```

# Introduction (9) – XML Syntax Rules

- **All XML Elements Must Have a Closing Tag**
  - In XML, it is **illegal to omit the closing tag**. All elements **must have a closing tag**
  - **Note:** The XML prolog does not have a closing tag! This is not an error. The prolog is not a part of the XML document.
- **XML Tags are Case Sensitive**
  - XML tags are **case sensitive**. The tag `<Letter>` is different from the tag `<letter>`.
  - Opening and closing tags must be written with the same case:  
`<message>This is correct</message>`
- **XML Elements Must be Properly Nested**

# Introduction (10) – XML Syntax Rules

- **XML Attribute Values Must Always be Quoted**

- XML elements can have attributes in name/value pairs just like in HTML.
- In XML, the attribute values must always be quoted:

```
<note date="12/11/2007">  
    <to>Harry</to>  
    <from>Marry</from>  
</note>
```

- **Comments in XML**

- The syntax for writing comments in XML is similar to that of HTML:

```
<!-- This is a comment -->
```

- Two dashes in the middle of a comment are not allowed:

```
<!-- This is an invalid -- comment -->
```

# Introduction (11) – XML Syntax Rules

- Entity References

- Some characters have a special meaning in XML.
- If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.
- This **will generate an XML error:**

```
<message>salary < 1000</message>
```

- To avoid this error, replace the "<" character with an entity reference:

```
<message>salary &lt; 1000</message>
```

There are 5 pre-defined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

# Introduction (12) – XML Syntax Rules

- **White-space is Preserved in XML**
  - XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	Hello	Tove
HTML:	Hello Tove	

- **Creating a multiword tags in XML (or Naming Conventions for XML Elements)**
  - **Using Underscore** – e.g., <first\_name></first\_name>
  - **Using Hyphen** – e.g., <first-name></first-name>
  - **Using Camel Case** – e.g., <firstName></firstName>

# XML Elements (1)

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

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

- An element can contain: **text, attributes, other elements or a mix of both.**
- **Example:**

```
<bookstore>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

- **In the example above:** `<title>`, `<author>`, `<year>`, and `<price>` have **text content** because they contain text (like Learning XML, 39.95 etc). `<bookstore>` and `<book>` have **element contents**, because they contain elements. `<book>` has an **attribute** (`category="web"`).

# XML Elements (2)

- **Empty XML Elements**
  - An element with no content is said to be empty.
  - In XML, you can indicate an empty element like : <element></element>
  - You can also use a so called self-closing tag: <element />
  - **Note:** Empty elements can have attributes.
- **XML Naming Rules**
  - XML elements must follow these naming rules:
    - ❖ Element names are case-sensitive
    - ❖ Element names must start with a letter or underscore
    - ❖ Element names cannot start with the letters xml (or XML, or Xml, etc)
    - ❖ Element names can contain letters, digits, hyphens, underscores, and periods
    - ❖ Element names cannot contain spaces
    - ❖ **Any name can be used, no words are reserved (except xml).**

# XML Elements (3)

- Best Naming Practices
  - Create descriptive names, like this: <person>, <firstname>, <lastname>.
  - Create short and simple names, like this: <book\_title> not like this: <the\_title\_of\_the\_book>.
  - Avoid “-”: If you name something "first-name", some software may think you want to subtract "name" from "first".
  - Avoid “.”: If you name something "first.name", some software may think that "name" is a property of the object "first".
  - Avoid “:”: Colons are reserved for namespaces (more later).
  - Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them!

# XML Elements (4) – An Example

**Q. Create a simple XML file to keep contact's detail.**

```
<?xml version="1.0" encoding="UTF-8"?>
<contactsList>
    <contact>
        <name>Angel Dimaria</name>
        <address>Rosario, Argentina</address>
        <email>dimaria11@gmail.com</email>
        <phoneNumber>9999999999</phoneNumber>
    </contact>
    <contact>
        <name>Cristiano Ronaldo</name>
        <address>Funchal, Portugal</address>
        <email>cr7@gmail.com</email>
        <phoneNumber>1111111111</phoneNumber>
    </contact>
</contactsList>
```

# XML Attributes (1)

- An **empty element is not necessarily meaningless**. It may have some properties in terms of attributes.
- An attribute is a name-value pair inside the opening tag of an element.
- **Attribute values must always be quoted. Either single or double quotes can be used.**
- **XML (Without Attribute):**

```
<lecturer>
    <name>David Billington</name>
    <phone> +61 – 7 – 3875 507 </phone>
</lecturer>
```
- **XML (With Attribute):**

```
<lecturer name="David Billington" phone="+61 – 7 – 3875 507"/>
```

**Or,**

```
<lecturer name="David Billington" phone="+61 – 7 – 3875 507"></lecturer>
```

# XML Attributes (2) – An Example

- **XML (Without Attribute):**

```
<order>
  <orderNo>23456</orderNo>
  <customerName>John Smith</customer>
  <date>October 15, 2002</date>
  <item>
    <itemNo>a528</itemNo>
    <quantity>1</quantity>
  </item>
  <item>
    <itemNo>c817</itemNo>
    <quantity>3</quantity>
  </item>
</order>
```

# XML Attributes (3) – An Example

- **XML (With Attribute):**

```
<order orderNo="23456" customerName="John Smith" date="October 15, 2002">
    <item itemNo="a528" quantity="1"/>
    <item itemNo="c817" quantity="3"/>
</order>
```

# XML Attributes (4) – Alternative Ways to Write XML

- The following three XML documents contain exactly the same information:

- A date attribute is used in the first example:

```
<note date="2008-01-10">
    <to>Harry</to>
    <from>Marry</from>
</note>
```

- A <date> element is used in the second example:

```
<note>
    <date>2008-01-10</date>
    <to>Harry</to>
    <from>Marry</from>
</note>
```

# XML Attributes (5) – Alternative Ways to Write XML

- An expanded `<date>` element is used in the third example:

```
<note>
  <date>
    <year>2008</year>
    <month>01</month>
    <day>10</day>
  </date>
  <to>Harry</to>
  <from>Marry</from>
</note>
```

# XML Attributes (6)

- **Avoid XML Attributes?**
  - *Some things to consider when using attributes are:*
    - ❖ attributes cannot contain multiple values (elements can)
    - ❖ attributes cannot contain tree structures (elements can)
    - ❖ attributes are not easily expandable (for future changes)
  - *Don't end up like this:*

```
<note    day="10"    month="01"    year="2008"    to="Harry"    from="Marry"
heading="Reminder" body="Don't forget me this weekend!"> </note>
```

# XML Namespaces (1)

- **XML Namespaces** provide a method to **avoid element name conflicts**.
- **Name Conflicts**
  - In XML, element names are defined by the developer. This often results in a conflict when trying to mix XML documents from different XML applications.
  - **This XML carries HTML table information:**

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

# XML Namespaces (2)

- **This XML carries information about a table (a piece of furniture):**

```
<table>
    <name>African Coffee Table</name>
    <width>80</width>
    <length>120</length>
</table>
```

- **If these XML fragments were added (or combined) together, there would be a name conflict.** Both contain a `<table>` element, but the elements have different content and meaning.
- A user or an XML application will not know how to handle these differences.

# XML Namespaces (3)

- **Solving the Name Conflict Using a Prefix**

- Name conflicts in XML **can easily be avoided using a name prefix.**
- This XML carries information about an HTML table, and a piece of furniture:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

- In the example above, there will be no conflict because the two `<table>` elements have different names.

# XML Namespaces (4) – The xmlns Attribute

- XML namespaces are used to **avoid conflicts in element and attribute names** when XML documents are combined from multiple sources.
- They provide a way to uniquely identify elements and attributes by associating them with a specific namespace.
- This ensures that elements or attributes with the same name but from different sources or domains can coexist without any naming clashes.
- The namespace declaration has the **following syntax:**

```
xmlns:prefix="URI"
```

- XML namespaces are declared using the **xmlns attribute** in the XML document.
- The xmlns attribute is usually placed in the root element to define the default namespace for that element and its descendants.

# XML Namespaces (5) – The xmlns Attribute

- **Note:** It's good practice to choose prefixes that are relevant to the namespace they represent. For example, if you have a namespace for books, you might use "book" as the prefix, or for images, you might use "img"
- For well-known XML namespaces (e.g., XML Schema, XHTML, SVG), it's common to use standard prefixes defined in their respective specifications.
- **For example,** the "xs" prefix is commonly used for the XML Schema namespace, and the "svg" prefix is often used for the SVG namespace.
- **Avoid using "xml" as a prefix:** The "xml" prefix is reserved for core XML namespaces and should not be used for custom namespaces.

# XML Namespaces (6) – An Example

- Let's look at an example to illustrate XML namespaces: Suppose we have two XML documents, each defining a `<book>` element, but they represent different types of books, one for fiction and another for non-fiction.
  - **XML Document 1 (Fiction Book):**

```
<book xmlns="http://example.com/fiction">
    <title>Harry Potter and the Sorcerer's Stone</title>
    <author>J.K. Rowling</author>
</book>
```
  - **XML Document 2 (Non-Fiction Book):**

```
<book xmlns="http://example.com/nonfiction">
    <title>The Art of War</title>
    <author>Sun Tzu</author>
</book>
```
- **Note:** In the example above, the value is a URI (Uniform Resource Identifier), which serves as a unique identifier for the namespace.
- The URI can be any valid string, but it is usually a URL pointing to a resource that provides more information about the namespace (though it doesn't have to be an actual web resource).

# XML Namespaces (7) – An Example

- In previous example, the `xmlns` attribute in each `<book>` element is used to declare its namespace.
- The namespace URIs ("`http://example.com/fiction`" and "`http://example.com/nonfiction`") are arbitrary but serve as unique identifiers for their respective elements.
- Now, if we want to combine these two XML documents into one, we can do it using a root element that has different namespace declarations for each type of book.
- **Combined XML Document:**

```
<library xmlns:fiction="http://example.com/fiction" xmlns:nonfiction="http://example.com/nonfiction">
    <fiction:book>
        <fiction:title>Harry Potter and the Sorcerer's Stone</fiction:title>
        <fiction:author>J.K. Rowling</fiction:author>
    </fiction:book>
    <nonfiction:book>
        <nonfiction:title>The Art of War</nonfiction:title>
        <nonfiction:author>Sun Tzu</nonfiction:author>
    </nonfiction:book>
</library>
```

# Structuring XML Documents

- Define all the element and attribute names that may be used
- **Define the structure**
  - what values an attribute may take
  - which elements may or must occur within other elements, etc.
- If such structuring information exists, the document can be validated
- **An XML document is valid if**
  - it is well-formed
  - respects the structuring information it uses
- **There are two ways of defining the structure of XML documents: (Grammar for XML)**
  - DTDs (the older and more restricted way)
  - XML Schema (offers extended possibilities)

# Document Type Definition (1)

- **DTD** stands for Document Type Definition.
- A DTD is a set of rules that allow us to specify our own set of elements and attributes.
- DTD is grammar to indicate what tags are legal in XML documents.
- XML Document is valid if it has an attached DTD and document is structured according to rules defined in DTD.
- DTD have “.dtd” as file extension.
- **DTDs can be defined as: (Types of DTD)**
  - Inline DTD - inline within the XML document
  - External DTD - in a separate external file

# Document Type Definition (2) – Notations/Attributes

- **Notations in DTD:**
  - **<!ELEMENT>** : Defines the structure of an XML element.
  - **<!ATTLIST>** : Defines the attributes of an element.
  - **(#PCDATA)** : Represents parsed character data, such as plain text.
  - **\*** : Zero or more occurrences of the specified element.
  - **+** : One or more occurrences of the specified element.
  - **?** : Zero or one occurrence of the specified element.
- **Attribute Type:**
  - The data type of the attribute's value. It can be one of the following:
    - ❖ **CDATA** : Character data, which can contain any characters, including special characters.
    - ❖ **ID** : Unique identifier for the attribute value within the document.
- **Attribute Default Value:**
  - The default value for the attribute, if any. It can be one of the following:
    - ❖ **#REQUIRED** : The attribute must be specified and have a value in the XML instance.
    - ❖ **#IMPLIED** : The attribute is optional and doesn't need to be specified in the XML instance.

# Document Type Definition (3) – Types of DTD

- **Syntax of an inline DTD:**

```
<!DOCTYPE root_element [  
    <!-- DTD declarations go here -->  
]>
```

- **Syntax of an external DTD:**

```
<!DOCTYPE root_element SYSTEM "path/to/dtd_file.dtd">
```

- Where,

- **<!DOCTYPE>** : declaration is used at the beginning of an XML document to define the document type and the location of the associated Document Type Definition (DTD).
- **root\_element** : This is the name of the root element of the XML document. It specifies the starting point of the document's structure.
- **SYSTEM** : This keyword indicates that the DTD is referenced using a system identifier (a file path or URL).
- **"path/to/dtd\_file.dtd"** : This is the system identifier that points to the location of the DTD file. It can be either a local file path or a URL.

# Document Type Definition (4) – Inline DTD Example

- **Inline DTD** is defined within **the same XML document** using the `<!DOCTYPE>` declaration

```
<?xml version="1.0" encoding="UTF-8"?>          <library>
<!DOCTYPE library [                         <book genre="Fantasy">
    <!ELEMENT library (book+)>             <title>Harry Potter – The Stone</title>
    <!ELEMENT book (title, author)>         <author>J.K. Rowling</author>
    <!ATTLIST book
        genre CDATA #IMPLIED >           </book>
        <!ELEMENT title (#PCDATA)>          <book genre="Fantasy">
        <!ELEMENT author (#PCDATA)>         <title>The Hobbit</title>
                                            <author>J.R.R. Tolkien</author>
    ]>                                         </book>
                                              </library>
```

## Example: Library Books

# Document Type Definition (5) – External DTD Example

- **External DTD** is stored in a separate file and referenced in the XML document using the `<!DOCTYPE>` declaration with a system identifier.
- Let us take same example of ‘Library Books’:

- **XML Document (books.xml):**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library SYSTEM "books.dtd">
<library>
    <book genre="Fantasy">
        <title>Harry Potter and the Sorcerer's Stone</title>
        <author>J.K. Rowling</author>
    </book>
    <book genre="Fantasy">
        <title>The Hobbit</title>
        <author>J.R.R. Tolkien</author>
    </book>
</library>
```

# Document Type Definition (6) – External DTD Example

- **External DTD File (books.dtd):**

```
<!ELEMENT library (book+)>
<!ELEMENT book (title, author)>
<!ATTLIST book genre CDATA #IMPLIED >
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
```

- **Task: Convert this XML document to DTD document**

```
<lecturer>
  <name>David Billington</name>
  <subject>Discrete Mathematics</name>
  <phone>+61 - 7 - 3875 507</phone>
</lecturer>
```

# XML Schema (1) - Introduction

- **XML Schema**, often referred to as **XSD** (XML Schema Definition), is a language for defining the structure, data types, and constraints of XML documents.
- XSD provides a more powerful and flexible way to validate and describe XML documents compared to DTD (Document Type Definition).
- XML Schema files use the ".xsd" extension.
- It allows you to define *complex data types*, *specify element relationships*, and *enforce constraints* more precisely.
- **Syntax of XML Schema:**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Schema definitions go here -->
</xs:schema>
```

# XML Schema (2) – Syntax and Notations

- XML Schema is written in XML syntax and consists of several components, including elements, attributes, complex types, simple types, and more. Here are some key notations used in XML Schema:

- **Defining Elements: (Syntax)**

```
<xs:element name="element_name" type="data_type"/>
```

- ❖ **xs:element** : The XML Schema namespace prefix for the "element" element.
    - ❖ **name** : Specifies the name of the element.
    - ❖ **type** : Specifies the data type of the element's content.

- **Defining Attributes: (Syntax)**

```
<xs:attribute name="attribute_name" type="data_type"/>
```

- ❖ **xs:attribute** : The XML Schema namespace prefix for the "attribute" element.
    - ❖ **name** : Specifies the name of the attribute.
    - ❖ **type** : Specifies the data type of the attribute's value.

# XML Schema (3) – Syntax and Notations

- **Defining Simple Types:**
  - simple type specifies the data format and allowed values for elements that hold simple information like names, string, numbers, or dates.
  - **Syntax:**

```
<xs:simpleType name="simple_type_name">
    <!-- Define the constraints of the simple type here -->
</xs:simpleType>
```

    - **xs:simpleType** : The XML Schema namespace prefix for the "simpleType" element.
    - **name** : Specifies the name of the simple type.

# XML Schema (4) – Syntax and Notations

- **Defining Complex Types:**

- XML Schema complex type defines the structure and composition of elements with child elements and attributes.

- **Syntax:**

```
<xs:complexType name="complex_type_name">  
    <!-- Define the structure of the complex type here -->  
</xs:complexType>
```

- **xs:complexType** : The XML Schema namespace prefix for the "complexType" element.
  - **name** : Specifies the name of the complex type.

# XML Schema (5) – Simple Type Example

- **XML Document (person.xml):**

```
<?xml version="1.0" encoding="UTF-8"?>
<age xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="person.xsd">25</age>
```

- **XML Schema (person.xsd):**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="age" type="xs:int"/>
</xs:schema>
```

# XML Schema (6) – Complex Type Example

- **XML Document (books.xml):**

```
<?xml version="1.0" encoding="UTF-8"?>
<book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="bookstore.xsd" genre="Fantasy">
    <title>Harry Potter and the Sorcerer's Stone</title>
    <author>J.K. Rowling</author>
</book>
```

# XML Schema (7) – Complex Type Example

- **XML Schema (bookstore.xsd):**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="book">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="author" type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="genre" type="xs:string"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# Introduction to XPath (1)

- **XPath (XML Path Language)** is a language used to navigate and select elements and attributes in XML documents.
- Imagine an XML document as a tree-like structure, with elements (nodes) and attributes (properties).
- XPath is like a map that helps you navigate this tree and find specific elements or attributes you're interested in.
- It uses a concise syntax to describe paths to reach nodes, similar to directories in a file system.
- **For example**, given an XML document representing books, the XPath expression **“/books/book[1]/title”** points to the title of the first book in the "books" element.
- **Note:** In XPath, the index for nodes or elements start from 1 not 0.

# Introduction to XPath (2)

- Example

```
<books>
  <book id="1">
    <title>Harry Potter</title>
    <author>J.K. Rowling</author>
  </book>
  <book id="2">
    <title>The Alchemist</title>
    <author>Paulo Coelho</author>
  </book>
</books>
```

- Selecting Elements

- **/books/book[1]/title** : Harry Potter
- **/books/book[2]/author** : Paulo Coelho

- Selecting Attributes

- **/books/book[1]/@id** : 1
- **/books/book[2]/@id** : 2

# Introduction to XSLT (1)

- **eXtensible Stylesheet Language Transformations (XSLT)** is like a set of rules or templates that instruct how to transform XML data into a different format.
- You define these templates to specify how to display, format, or extract data from the XML document.
- Think of **XSLT as a recipe** that takes an XML input and turns it into another XML, HTML, or even plain text output.
- **For instance**, you can create an XSLT to convert an XML file containing weather data into a nicely formatted HTML table showing the forecast.

# Introduction to XSLT (2)

- Simple XSLT Example

- XML Example:

```
<books>
  <book>
    <title>Harry Potter</title>
    <author>J.K. Rowling</author>
  </book>
</books>
```

- Convert XML to HTML

```
<xsl:template match="/books">
  <html>
    <body>
      <h2><xsl:value-of select="book/title"/></h2>
    </body>
  </html>
</xsl:template>
```

- Output:

Displays **Harry Potter** as HTML heading

# Introduction to XQuery (1)

- **XQuery (XML Query)** is a language designed for querying and extracting data from XML documents.
- If **XPath is a navigation tool**, **XQuery is a querying tool** for XML documents.
- It allows you to ask questions and retrieve specific pieces of data from XML documents.
- You can use XQuery to filter, sort, and extract information based on certain conditions or criteria.
- **For example**, in a large XML database of products, XQuery can help you find all products with prices below a certain value.

# Introduction to XQuery (2)

- Simple XQuery Example

**XML Example:**

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>500</price>
  </book>
  <book>
    <title>The Alchemist</title>
    <price>300</price>
  </book>
</books>
```

- **XQuery Code:**

```
for $b in /books/book
return $b/title
```

- **Output:**

```
<title>Harry Potter</title>
<title>The Alchemist</title>
```

**Note:** *It does NOT return plain text unless you use data().*

- Hence, if you want only text use:

```
for $b in /books/book
return data($b/title)
```

- **Output:**

```
Harry Potter
The Alchemist
```

# **THANK YOU!**