Unit 5: eXtensible Markup Language 5 LHs

Introduction; Syntax; Elements and Attributes; Namespace; DTD and Schema; Introduction to XPath, XSLT, and XQuery.

Yuba Raj Devkota

Introduction to XML (eXtensible Markup Language)

XML, or eXtensible Markup Language, is a markup language designed to store and transport data in a structured format. Unlike HTML, which focuses on how data is displayed, XML focuses on what the data is and how it is organized. It allows developers to define their own tags and structure, making it highly flexible and suitable for various applications such as data interchange between systems, configuration files, and web services.

XML was developed by the World Wide Web Consortium (W3C) and is widely used in modern software systems for representing hierarchical data. Its primary purpose is to facilitate the sharing of structured data across different platforms and systems.

```xml
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Syntax of XML

The syntax of XML is straightforward and follows a set of rules that ensure the document is well-formed. A well-formed XML document adheres to the following basic rules:

1. **Declaration** : An XML document typically starts with an XML declaration that specifies the version and encoding.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

2. **Root Element** : Every XML document must have a single root element that encloses all other elements.

xml

```xml
<note>
    <!-- Other elements go here -->
</note>
```

3. **Elements** : Elements are defined using tags. Each element must have a start tag (`<tag>`) and an end tag (`</tag>`), or it can be self-closing (`<tag />`).

```xml
<to>Tove</to>
<from>Jani</from>
```

4. **Nesting** : Elements can be nested within each other, but they must not overlap.

```xml
<person>
    <name>John</name>
    <age>30</age>
</person>
```

5. **Case Sensitivity** : XML is case-sensitive, meaning `<Name>` and `<name>` are treated as different elements.

6. **Comments** : Comments can be added using `<!-- -->`.

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

7. **Attributes** : Elements can have attributes, which provide additional information about the element.

```xml
<book category="fiction">Harry Potter</book>
```

8. **CDATA Sections** : Special sections can be used to include text that should not be parsed by the XML parser.

```xml
<![CDATA[<script>alert("Hello");</script>]]>
```

Elements in XML

Elements are the building blocks of an XML document. They represent the data and its structure. Here's a breakdown of elements:

1. **Simple Elements** : These contain only text.

   xml
   ```xml
   <name>John Doe</name>
   ```

2. **Complex Elements** : These can contain other elements, attributes, or a mix of both.

   xml
   ```xml
   <person>
       <name>John Doe</name>
       <age>30</age>
   </person>
   ```

3. **Empty Elements** : These do not contain any content and are self-closing.

```xml
<br />
```

4. **Nested Elements** : Elements can be nested within one another to represent hierarchical relationships.

```xml
<library>
    <book>
        <title>XML Basics</title>
        <author>John Smith</author>
    </book>
</library>
```

Attributes in XML

Attributes provide additional information about elements. They are always placed inside the start tag of an element and consist of a name-value pair. Here are some key points about attributes:

1. **Syntax** :

   ```xml
   <element attribute="value">Content</element>
   ```

2. **Example** :

   ```xml
   <book category="fiction">Harry Potter</book>
   ```

3. **Rules** :
    - Attribute values must always be enclosed in quotes (`"` or `'`).
    - Each attribute name must be unique within an element.
4. **Use Cases** :
    - Attributes are often used to describe properties of an element, such as metadata or identifiers.

```xml
<student id="12345" major="Computer Science">
    <name>Alice</name>
</student>
```

XML is a powerful and flexible language for representing structured data. Its syntax is simple yet robust, and its features make it ideal for data storage, interchange, and configuration. By understanding the basics of XML—such as elements, attributes, and syntax—you can effectively use it in various applications, from web development to enterprise systems.

Namespace in XML

In XML, a **namespace** is a mechanism used to avoid element name conflicts. Namespaces allow you to differentiate between elements or attributes that may have the same name but come from different vocabularies or schemas. This is especially important when combining XML documents from different sources or when using multiple XML-based languages within the same document.

Namespaces are identified by a **Uniform Resource Identifier (URI)** , which is typically a URL (though it doesn't have to point to an actual web resource). The URI acts as a unique identifier for the namespace, ensuring that element names are globally unique.

Namespaces are declared using the `xmlns` attribute. Once declared, all elements and attributes within that scope are considered part of the namespace unless otherwise specified.

Syntax of Namespace Declaration

1. **Default Namespace** : A default namespace applies to all elements within the scope where it is declared.

```xml
xmlns="URI"
```

2. **Prefixed Namespace** : A prefixed namespace uses a prefix to associate elements with a specific namespace.

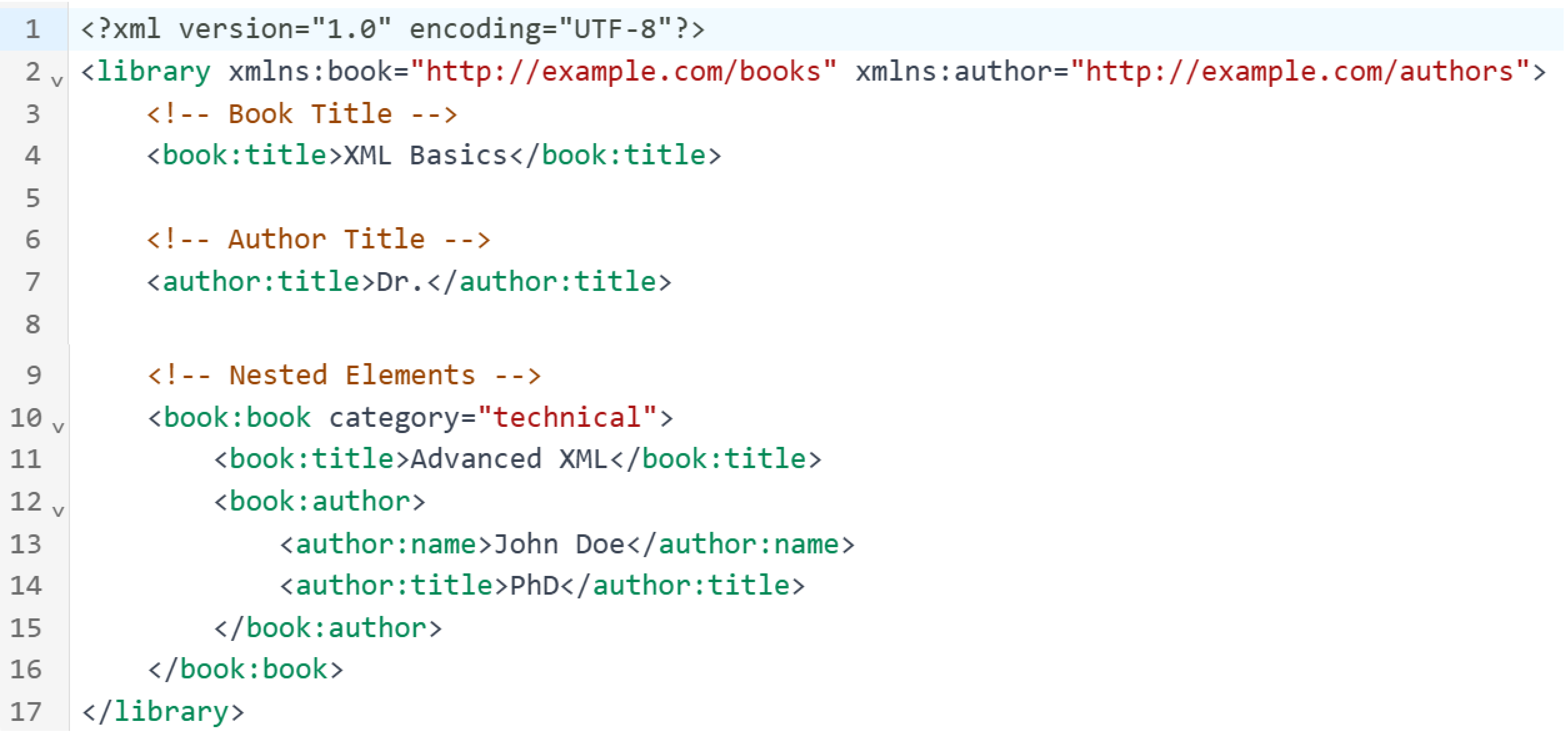
```xml
xmlns:prefix="URI"
```

Scenario:

Suppose you are working with two different XML vocabularies: one for books (`http://example.com/books`) and another for authors (`http://example.com/authors`). Both vocabularies use the `<title>` element, but they mean different things (e.g., book title vs. author's title).

To avoid confusion, you can use namespaces to distinguish between the two.

1. **Namespace Declaration** :
   - `xmlns:book="http://example.com/books"` declares a namespace for book-related elements, with the prefix `book` .
   - `xmlns:author="http://example.com/authors"` declares a namespace for author-related elements, with the prefix `author` .
2. **Element Usage** :
   - `<book:title>` refers to the `title` element in the `http://example.com/books` namespace.
   - `<author:title>` refers to the `title` element in the `http://example.com/authors` namespace.
3. **Nested Elements** :
   - The `<book:author>` element contains nested elements from the `author` namespace, such as `<author:name>` and `<author:title>` .



```xml
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:book="http://example.com/books" xmlns:author="http://example.com/authors">
    <!-- Book Title -->
    <book:title>XML Basics</book:title>

    <!-- Author Title -->
    <author:title>Dr.</author:title>

    <!-- Nested Elements -->
    <book:book category="technical">
        <book:title>Advanced XML</book:title>
        <book:author>
            <author:name>John Doe</author:name>
            <author:title>PhD</author:title>
        </book:author>
    </book:book>
</library>
```

Default Namespace Example

If you want to avoid using prefixes for a particular namespace, you can declare it as the default namespace. All elements without a prefix will belong to this namespace.

XML Document with Default Namespace:

xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://example.com/books">
    <title>XML Basics</title>
    <book category="technical">
        <title>Advanced XML</title>
        <author xmlns="http://example.com/authors">
            <name>John Doe</name>
            <title>PhD</title>
        </author>
    </book>
</library>
```

XML DTD: Document Type Definition

DTD (Document Type Definition) is a set of markup declarations that define the structure, elements, attributes, and entities allowed in an XML document. It acts as a "rulebook" to ensure that XML documents adhere to a specific format. DTDs are used for **validation** to confirm that an XML document is "valid" (conforms to the DTD rules).

Key Components of a DTD

1. **Elements** : Define the structure and hierarchy of XML elements.
2. **Attributes** : Specify properties for elements.
3. **Entities** : Define shortcuts or reusable text.
4. **Notations** : Describe external data formats (e.g., images, binary files).

Types of DTD

DTDs can be categorized into two types based on their location:

1. **Internal DTD**
   - Declared within the XML document itself, inside the `<!DOCTYPE>` declaration.
   - Useful for small, standalone documents.
   - Example:

```xml
<?xml version="1.0"?>
<!DOCTYPE note [
    <!ELEMENT note (to, from, heading, body)>
    <!ELEMENT to      (#PCDATA)>
    <!ELEMENT from    (#PCDATA)>
    <!ELEMENT heading (#PCDATA)>
    <!ELEMENT body    (#PCDATA)>
]>
<note>
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Don't forget the meeting!</body>
</note>
```

2. **External DTD**
   - Stored in a separate `.dtd` file and referenced in the XML document.
   - Ideal for reusability across multiple XML files.
   - Example:

- **XML File (`note.xml`):**

```xml
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Don't forget the meeting!</body>
</note>
```

DTD File (`note.dtd`):

```dtd
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to      (#PCDATA)>
<!ELEMENT from    (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body    (#PCDATA)>
```

DTD Syntax Examples

1. Element Declarations

- Syntax: `<!ELEMENT element-name (content-type)>`
- **Content Types** :
  - `#PCDATA` : Parsed character data (text).
  - `EMPTY` : Element cannot contain content (e.g., `<br/>`).
  - `ANY` : Element can contain any content.
  - **Nested Elements** : Define order and occurrence.

```
<!-- "author" must appear at least once, "price" is optional -->
```

```
<!ELEMENT book (title, author+, price?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (first-name, last-name)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
```

2. Attribute Declarations

- Syntax: `<!ATTLIST element-name attribute-name type default-value>`
- **Attribute Types** :
  - `CDATA` : Character data.
  - `ID` : Unique identifier.
  - `ENUMERATED` : List of allowed values.

```
<!ATTLIST book
    category    CDATA    #REQUIRED
    edition     (first | second | third) "first"
    isbn        ID       #IMPLIED
>
```

3. Entities

- Define reusable text or external resources.
- **Internal Entity** :

```dtd
<!ENTITY copyright "© 2024 Example Corp.">
```

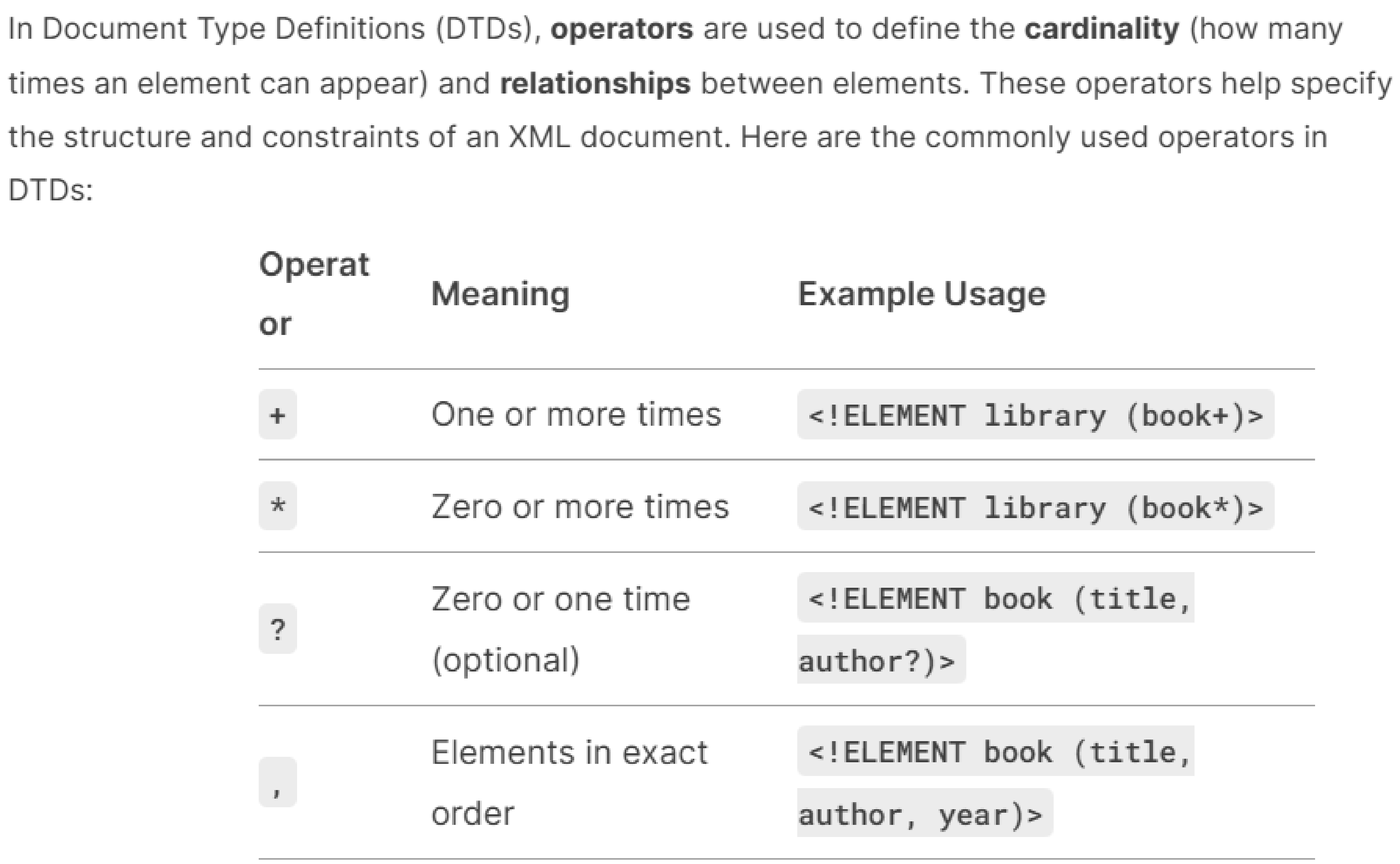
  Usage in XML: `&copyright;`
- **External Entity** :

```dtd
<!ENTITY company_logo SYSTEM "logo.jpg" NDATA JPEG>
```

When to Use DTD?

- For simple validation needs.
- When working with legacy systems (e.g., HTML).
- For small projects where XML Schema (XSD) is overkill.

Operators used in DTD



In Document Type Definitions (DTDs), **operators** are used to define the **cardinality** (how many times an element can appear) and **relationships** between elements. These operators help specify the structure and constraints of an XML document. Here are the commonly used operators in DTDs:

Operator	Meaning	Example Usage
`+`	One or more times	`<!ELEMENT library (book+)>`
`*`	Zero or more times	`<!ELEMENT library (book*)>`
`?`	Zero or one time (optional)	`<!ELEMENT book (title, author?)>`
`,`	Elements in exact order	`<!ELEMENT book (title, author, year)>`

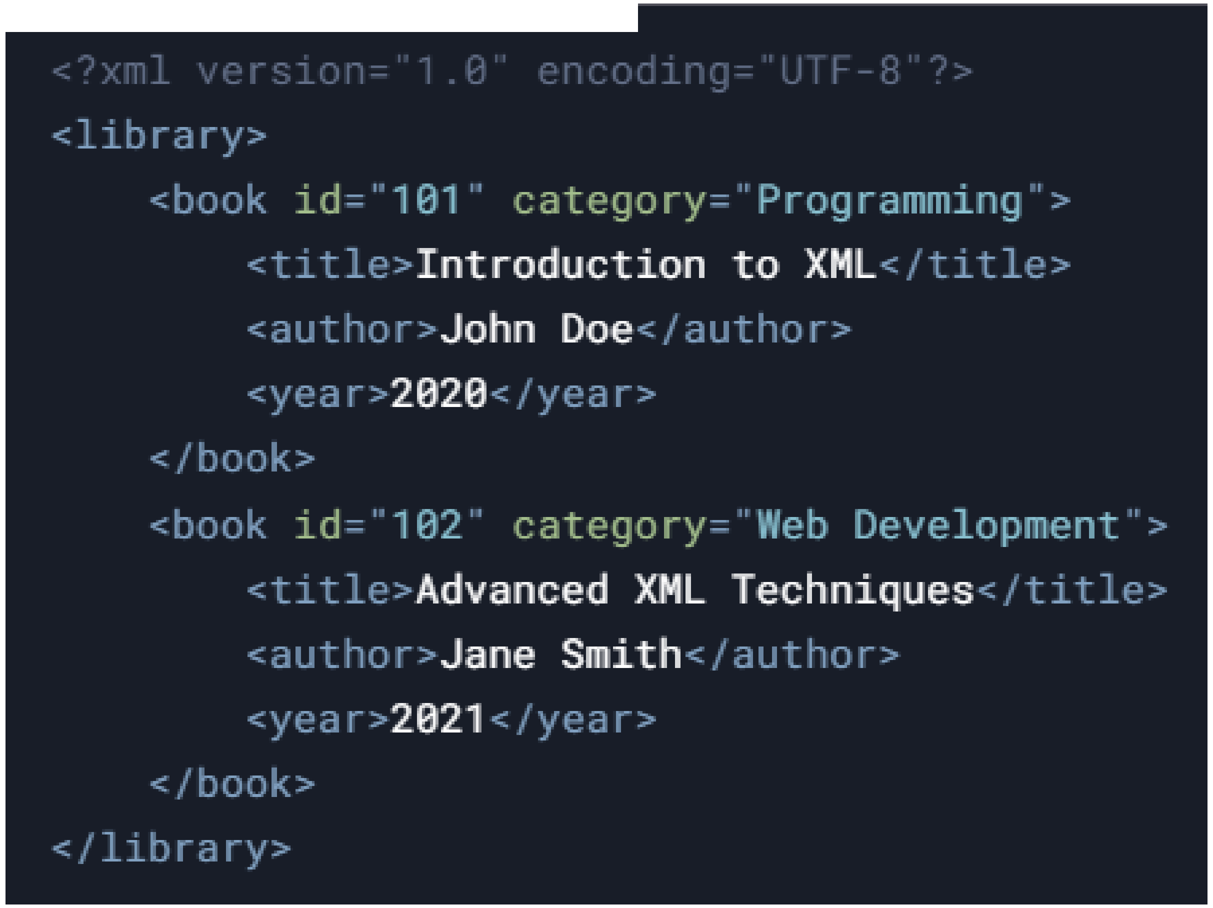
Example Combining Operators:

```xml
<!ELEMENT library (book+, (magazine | journal)*)>
```

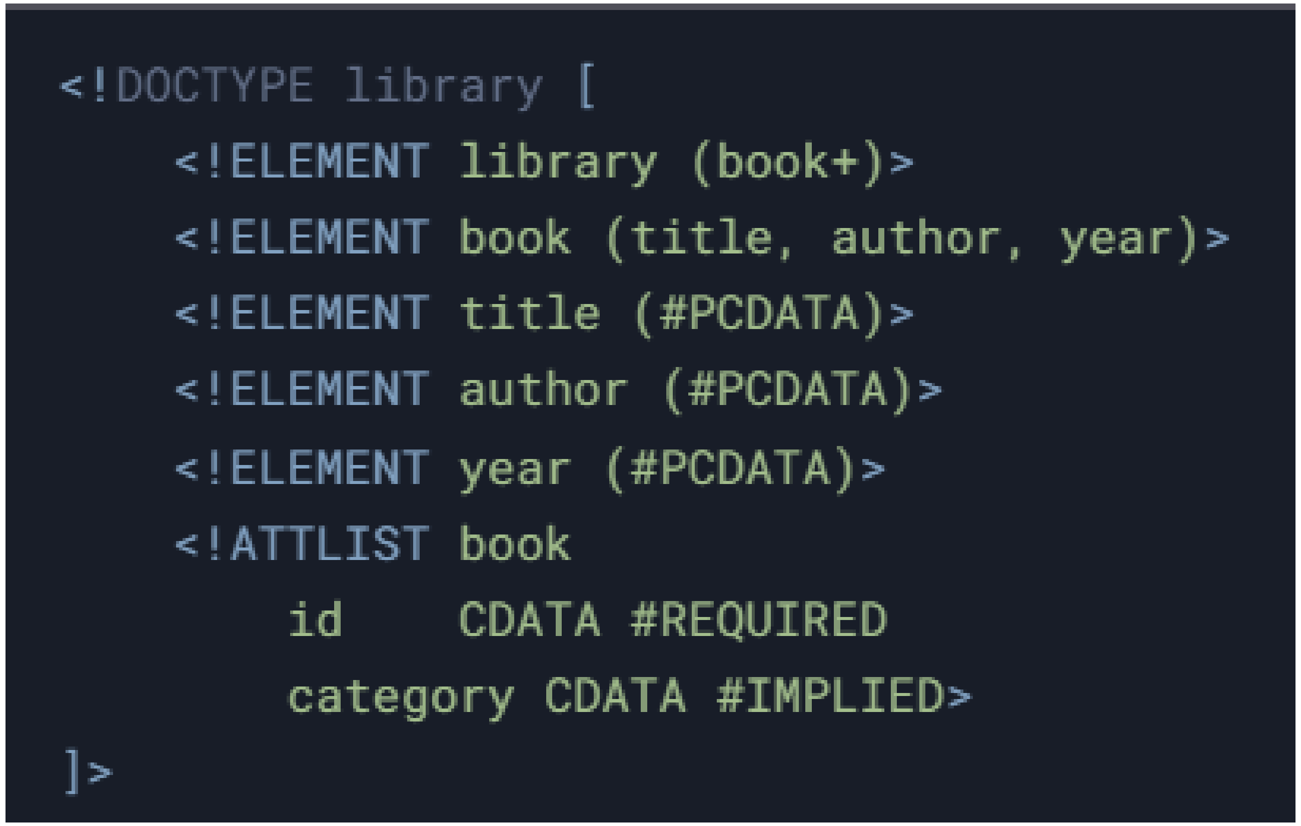
- This means:
    - The `library` element must contain **one or more** `book` elements.
    - It can also contain **zero or more** `magazine` or `journal` elements (in any order).

Write a Document Type Definition (DTD) that defines the structure of this XML document. The DTD should enforce the following rules:

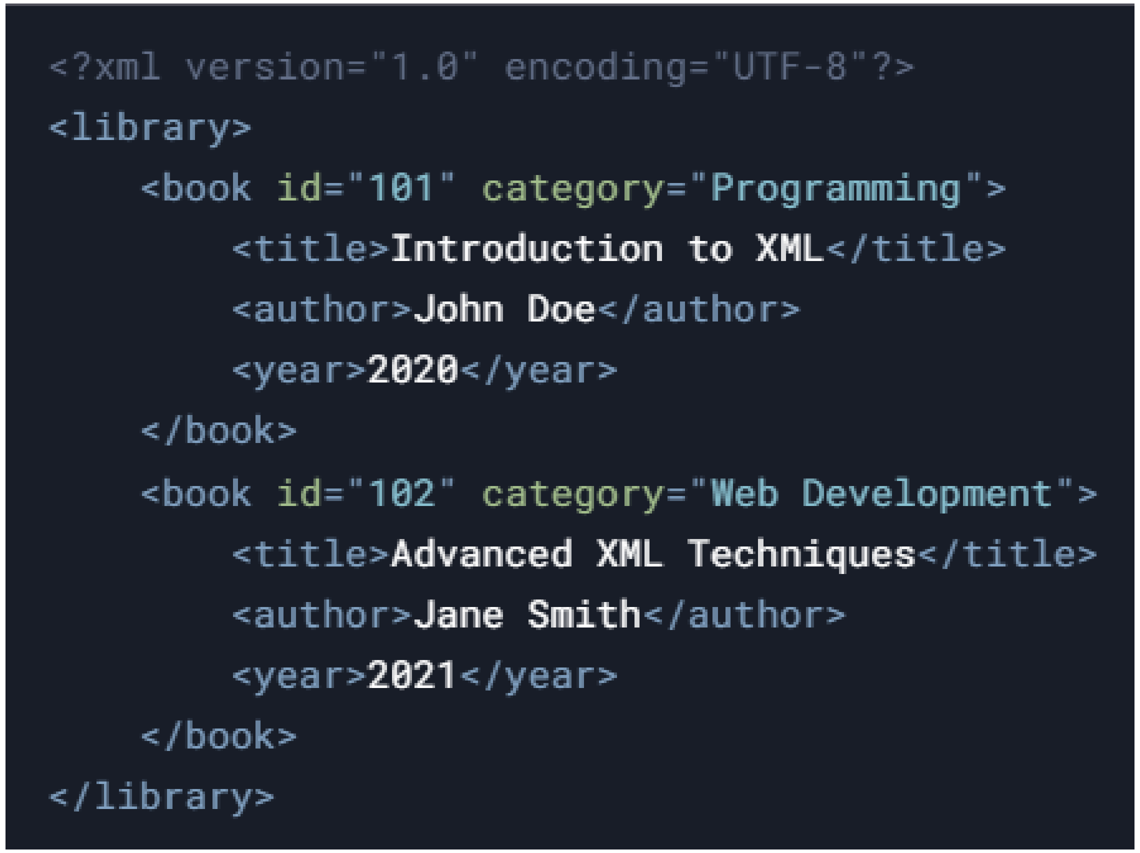
1. The root element is `library`.
2. The `library` element contains one or more `book` elements.
3. Each `book` element contains exactly one `title`, one `author`, and one `year` element, in that order.
4. The `title` and `author` elements contain only text.
5. The `year` element contains only numbers.
6. Each `book` element has two attributes:
   - `id` (required, must be a number).
   - `category` (optional, must be a string).



```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="101" category="Programming">
        <title>Introduction to XML</title>
        <author>John Doe</author>
        <year>2020</year>
    </book>
    <book id="102" category="Web Development">
        <title>Advanced XML Techniques</title>
        <author>Jane Smith</author>
        <year>2021</year>
    </book>
</library>
```



```
<!DOCTYPE library [
    <!ELEMENT library (book+)>
    <!ELEMENT book (title, author, year)>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT year (#PCDATA)>
    <!ATTLIST book
        id     CDATA #REQUIRED
        category CDATA #IMPLIED>
]>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<library>
    <book id="101" category="Programming">
        <title>Introduction to XML</title>
        <author>John Doe</author>
        <year>2020</year>
    </book>
    <book id="102" category="Web Development">
        <title>Advanced XML Techniques</title>
        <author>Jane Smith</author>
        <year>2021</year>
    </book>
</library>
```

XML Schema (XSD):

XML Schema Definition (XSD) is a powerful language used to describe the structure, data types, and constraints of XML documents. Unlike DTDs, XSD is written in XML syntax, supports namespaces, and provides rich data typing (e.g., strings, numbers, dates). XSD ensures that XML documents are **valid** (compliant with the schema rules).

Advantages of XSD Over DTD

1. **Data Types** : XSD supports rich data types (e.g., dates, numbers).
2. **Namespaces** : XSD fully supports XML namespaces.
3. **Extensibility** : Define reusable components (e.g., groups, attribute groups).
4. **Validation** : More precise validation rules (e.g., regex patterns).

XML Document (`employee.xml`):

xml

```xml
<?xml version="1.0"?>
<Employee xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="employee.xsd">
    <Name>John Doe</Name>
    <Age>30</Age>
    <Salary>60000.50</Salary>
</Employee>
```

XSD Schema (`employee.xsd`):

xml

```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Define the root element "Employee" -->
    <xs:element name="Employee">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Name" type="xs:string"/>
                <xs:element name="Age" type="xs:integer"/>
                <xs:element name="Salary" type="xs:decimal"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Common XSD Elements and Structures

1. **Simple Types** :
    - Define custom data types with restrictions.

xml

```xml
<xs:simpleType name="SKU">
    <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z]{2}\d{3}"/>
    </xs:restriction>
</xs:simpleType>
```

2. **Complex Types** :
    - Define elements with nested structure.

xml

```xml
<xs:complexType name="AddressType">
    <xs:sequence>
        <xs:element name="Street" type="xs:string"/>
        <xs:element name="City" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
```

3. **Choices and Sequences** :
   - `<xs:choice>` : Only one child element can appear.
   - `<xs:all>` : Child elements can appear in any order.

```xml
<xs:choice>
    <xs:element name="HomePhone" type="xs:string"/>
    <xs:element name="MobilePhone" type="xs:string"/>
</xs:choice>
```

4. **Occurrence Constraints** :
   - `minOccurs` and `maxOccurs` control how many times an element can appear.

```xml
<xs:element name="Item" minOccurs="1" maxOccurs="10"/>
```

Summary

- XSD is the standard for defining XML document structure and validation rules.
- Key features include data typing, namespaces, and complex validation constraints.
- Use XSD to ensure XML documents are well-formed, consistent, and error-free.

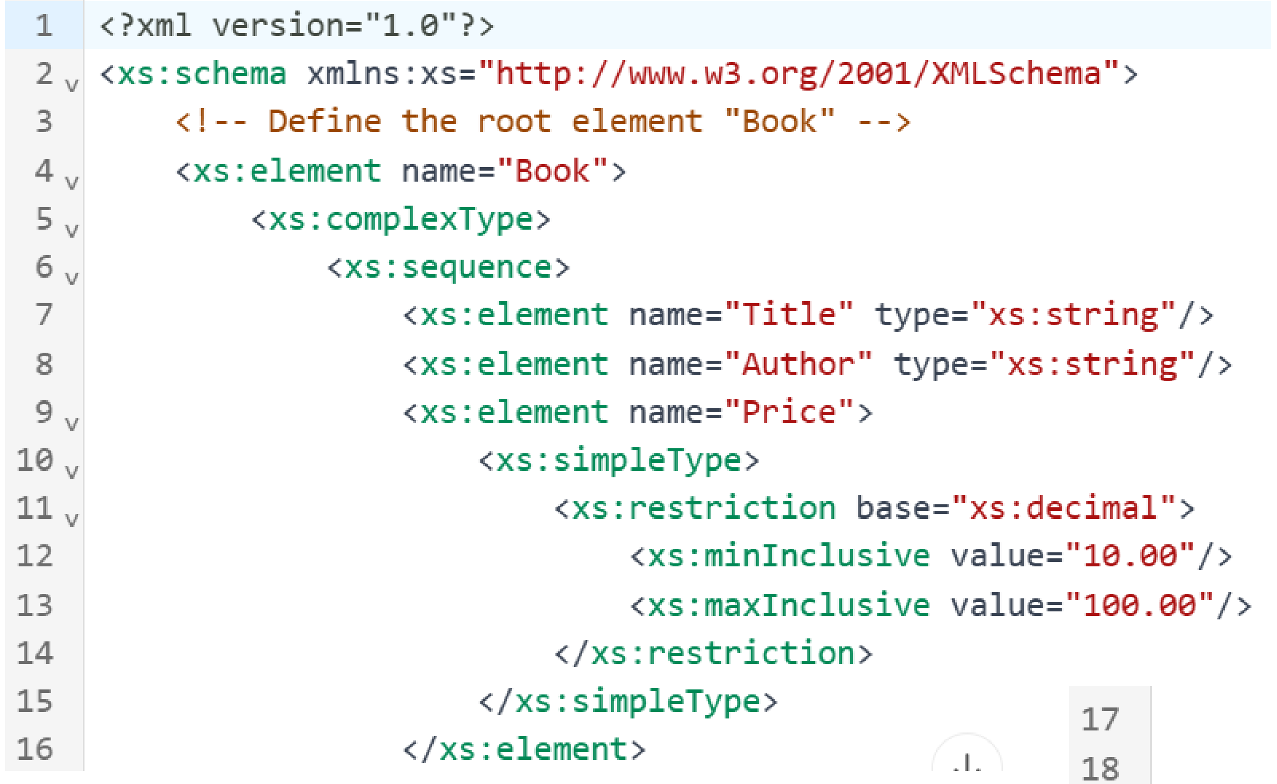
Example 2: XSD with Attributes and Restrictions

This example includes attributes, custom data types, and validation rules.

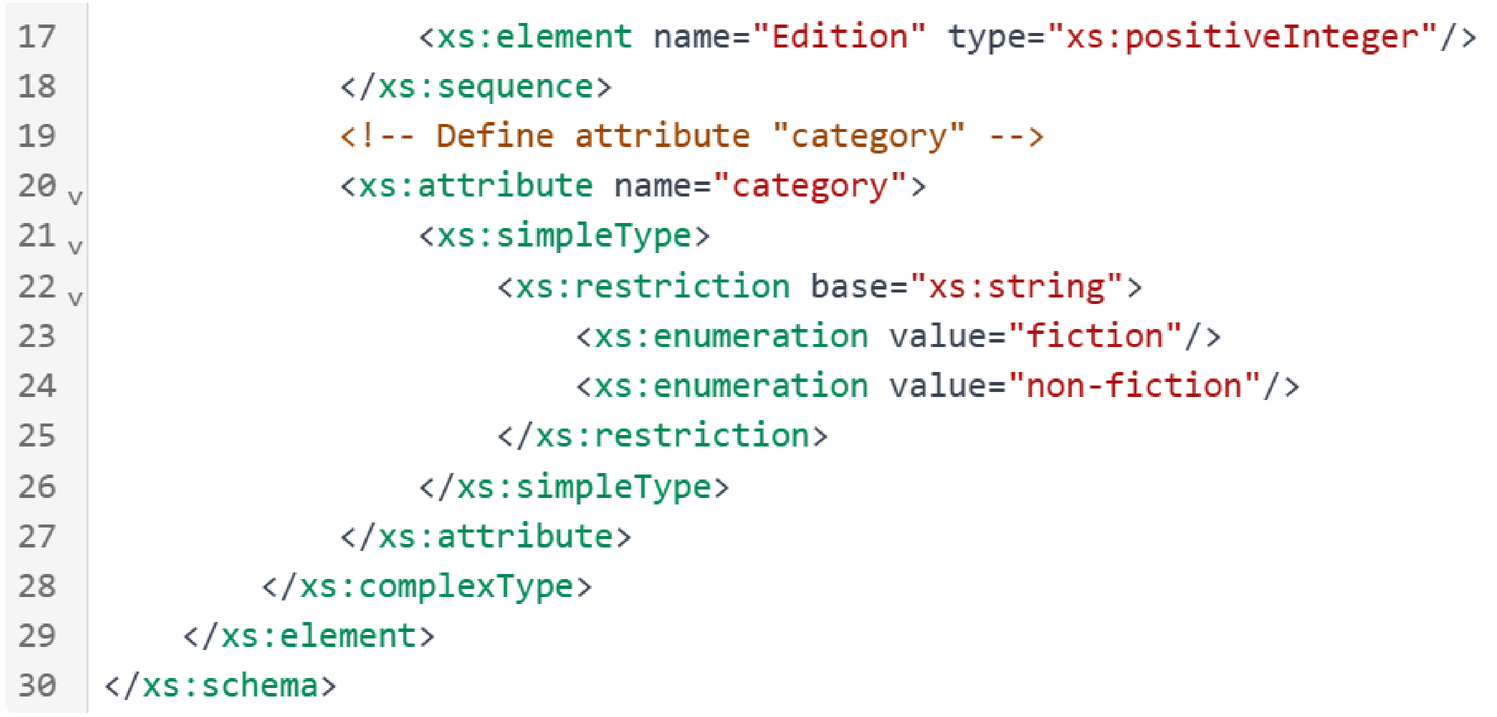
XML Document (`book.xml`):

```xml
<?xml version="1.0"?>
<Book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="book.xsd"
      category="fiction">
    <Title>XML Essentials</Title>
    <Author>John Smith</Author>
    <Price>49.99</Price>
    <Edition>2</Edition>
</Book>
```

- The `<Price>` element is restricted to values between `10.00` and `100.00`
- The `category` attribute is restricted to `fiction` or `non-fiction` .
- `<Edition>` must be a positive integer.



```xml
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Define the root element "Book" -->
    <xs:element name="Book">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Title" type="xs:string"/>
                <xs:element name="Author" type="xs:string"/>
                <xs:element name="Price">
                    <xs:simpleType>
                        <xs:restriction base="xs:decimal">
                            <xs:minInclusive value="10.00"/>
                            <xs:maxInclusive value="100.00"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:element>
```



```xml
                <xs:element name="Edition" type="xs:positiveInteger"/>
            </xs:sequence>
            <!-- Define attribute "category" -->
            <xs:attribute name="category">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="fiction"/>
                        <xs:enumeration value="non-fiction"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Write an XML Schema Definition (XSD) that defines the structure of this XML document. The XSD should enforce the following validation rules:

1. The root element is `library`.
2. The `library` element contains one or more `book` elements.
3. Each `book` element has the following attributes:
   - `id` (required, must be a positive integer).
   - `category` (required, must be a string with a maximum length of 50 characters).
   - `language` (optional, must be a string with a maximum length of 20 characters).
4. Each `book` element contains the following child elements in order:
   - `title` (must contain text, maximum length of 100 characters).
   - `author` (must contain text, maximum length of 50 characters).
   - `year` (must be a positive integer between 2000 and 2100).
   - `price` (must be a decimal value, with a required `currency` attribute that must be either "USD" or "EUR").

XPath (XML Path Language)

Definition :

XPath is a query language used to navigate and select nodes (elements, attributes, text) in an XML document. It uses path expressions to traverse the XML tree structure. XPath is often used in conjunction with XSLT and XQuery.

Key Features :

- Selects nodes using path-like syntax (e.g., `/library/book/title`).
- Supports predicates (filters) to refine selections (e.g., `book[@category='fiction']`).
- Works with axes (e.g., `child::`, `parent::`, `ancestor::`).

Example XML Document (`books.xml`):

```xml
<library>
    <book category="fiction">
        <title>XML Basics</title>
        <author>Jane Smith</author>
        <price>29.99</price>
    </book>
    <book category="non-fiction">
        <title>XML Cookbook</title>
        <author>John Doe</author>
        <price>45.50</price>
    </book>
</library>
```

1. Select all `<title>` elements:

   ```xpath
   /library/book/title
   ```

   Result :

   ```text
   XML Basics
   XML Cookbook
   ```

2. Select the price of the first book:

   ```xpath
   /library/book[1]/price
   ```

   Result : `29.99`

3. Select books with `category="fiction"` :

   ```xpath
   //book[@category='fiction']
   ```

   Result : The first `<book>` element.

4. Select the author of the second book:

   ```xpath
   //book[2]/author/text()
   ```

   Result : `John Doe`

XSLT (Extensible Stylesheet Language Transformations)

Definition :

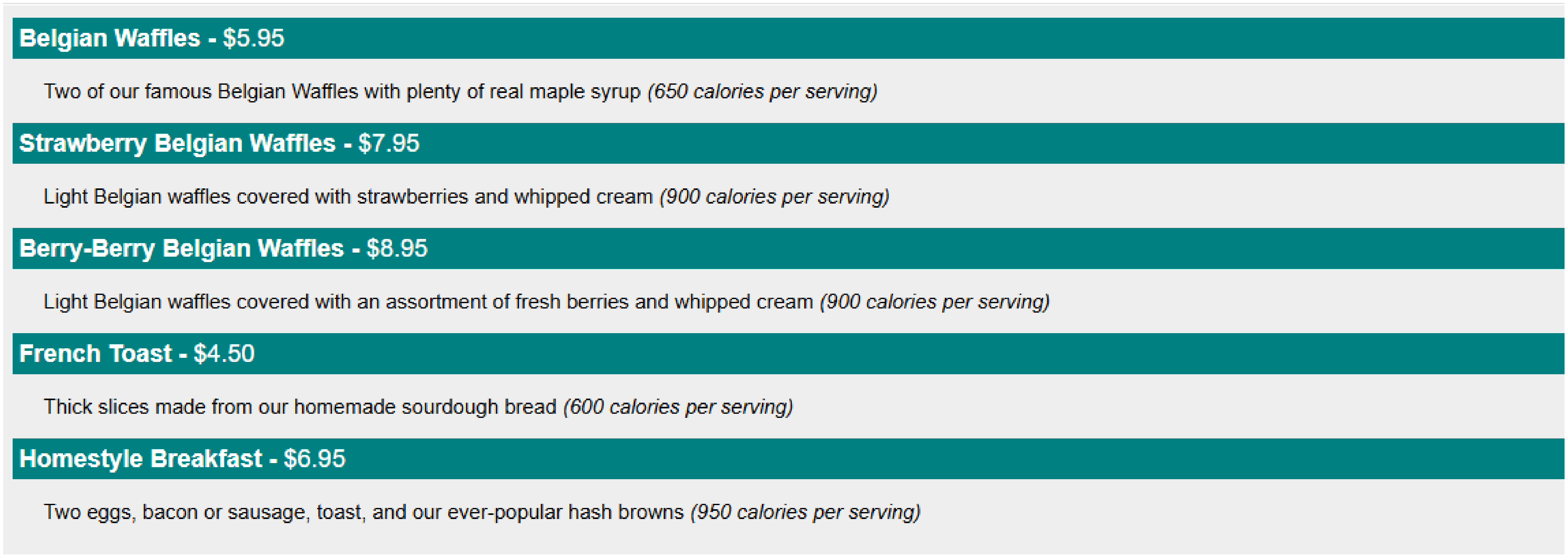
XSLT is a language for transforming XML documents into other formats (e.g., HTML, plain text, or another XML structure). It uses **templates** and **XPath** to match and process nodes.

Key Features :

- Converts XML into other formats (e.g., HTML, JSON).
- Uses templates with `<xsl:template>` to define rules.
- Leverages XPath to select and filter data.

```xml
<?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>

<food>
<name>French Toast</name>
<price>$4.50</price>
<description>Thick slices made from our homemade sourdough bread</description>
<calories>600</calories>
</food>
```

```
<?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>
```



Belgian Waffles - $5.95

Two of our famous Belgian Waffles with plenty of real maple syrup *(650 calories per serving)*

Strawberry Belgian Waffles - $7.95

Light Belgian waffles covered with strawberries and whipped cream *(900 calories per serving)*

Berry-Berry Belgian Waffles - $8.95

Light Belgian waffles covered with an assortment of fresh berries and whipped cream *(900 calories per serving)*

French Toast - $4.50

Thick slices made from our homemade sourdough bread *(600 calories per serving)*

Homestyle Breakfast - $6.95

Two eggs, bacon or sausage, toast, and our ever-popular hash browns *(950 calories per serving)*

XQuery (XML Query Language)

Definition :
XQuery is a query language designed to extract and manipulate data from XML documents. It is similar to SQL but tailored for XML. XQuery uses **FLWOR** expressions (For, Let, Where, Order by, Return) to process data.

Key Features :

- Query XML databases or documents.
- Supports complex transformations and aggregations.
- Combines XPath-like syntax with procedural logic.

```xquery
xquery version "3.1";

for $book in doc("books.xml")//book
where $book/@category = "fiction"
order by $book/price descending
return
    <result>
        <title>{$book/title}</title>
        <price>{$book/price}</price>
    </result>
```

```xml
<result>
    <title>XML Basics</title>
    <price>29.99</price>
</result>
```

Key Differences

TECHNOLOGY	PURPOSE	USED FOR
XPath	Node selection and navigation	Querying specific parts of an XML document.
XSLT	XML transformation	Converting XML to HTML, PDF, or other formats.
XQuery	Data extraction and manipulation	Querying XML databases or complex XML datasets.

Summary

- **XPath** : Navigate and select XML nodes (e.g., `/library/book/title`).
- **XSLT** : Transform XML into other formats using templates (e.g., HTML tables).
- **XQuery** : Query XML data with SQL-like syntax (e.g., filtering and sorting books).