**LAB 1: Introduction to XML**

**Objective:**

* To understand the purpose and structure of XML.

**Steps:**

* Open a text editor.
* Start with an XML declaration.
* Create a basic XML structure with elements.
* Save the file with a .xml extension.
* Open the file in a browser or XML viewer.

**Source Code:**

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

<message>

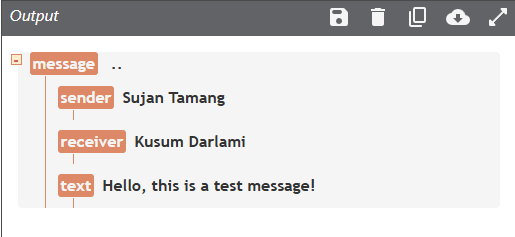
<sender>Sujan Tamang</sender>

<receiver>Kusum Darlami</receiver>

<text>Hello, this is a test message!</text>

</message>

**Output:**



**Conclusion:**

XML is used to store and transport structured data.

**LAB 2: Rules for writing XML**

**Objective:**

* To follow syntax rules for writing well-formed XML.

**Steps:**

* Ensure all tags are closed.
* Use proper nesting.
* Make tag names case-sensitive.
* Quote all attribute values.
* Include one root element.

**Source Code:**

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

<contact>

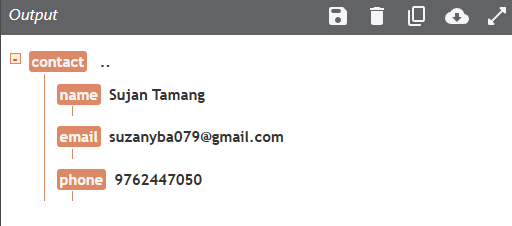
<name>Sujan Tamang</name>

<email>suzanyba079@gmail.com</email>

<phone>9762447050</phone>

</contact>

**Output:**



**Conclusion:**

XML syntax rules ensure the document is well-formed.

**LAB 3: Xml tree**

**Objective:**

* To understand the hierarchical (tree) structure of an XML document.

**Steps:**

* Write an XML file with nested elements.
* Identify the root and child elements.
* Visualize it as a tree.

**Source Code:**

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

<note>

<from>Sujan Tamang</from>

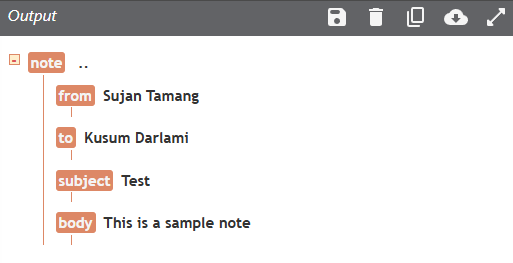
<to>Kusum Darlami</to>

<subject>Test</subject>

<body>This is a sample note</body>

</note>>

**Output:**

****

**Conclusion:**  
This lab illustrated that XML is organized in a tree structure where each element can contain sub-elements.

**Lab 4: Element**

**Objective:**

* To understand how elements are used in XML to represent structured data.

**Steps:**

* Create an XML document with nested tags representing data.
* Observe the element syntax and hierarchy.
* Check well-formedness.

**Source Code:**

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

<notes>

<noteone>

<to>Me</to>

<from>You</from>

</noteone>

<notetwo>

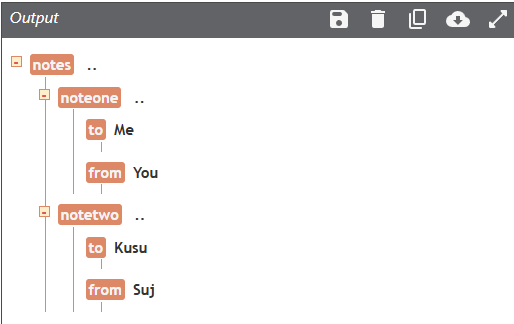
<to>Kusu</to>

<from>Suj</from>

</notetwo>

</notes>

**Output:**

****

**Conclusion:**  
This lab clarified how XML element’s structure data using tag-based syntax.

**Lab 5: Attribute**

**Objective:**

* To explore how attributes provide additional information about XML elements.

**Steps:**

* Write an XML file with both nested elements and attributes.
* Compare attribute usage to child elements.
* Validate the XML.

**Source Code:**

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

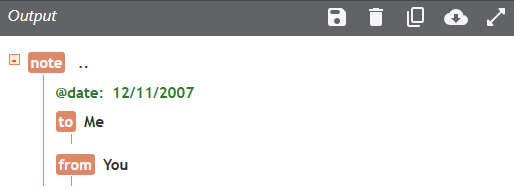
<note date="12/11/2007">

<to>Me</to>

<from>You</from>

</note>

**Output:**



**Conclusion:**  
Attributes in XML provide metadata for elements and offer an alternative to nested tags.

**LAB 6: Elements vs Attributes**

**Objective:**

* To use elements and attributes for storing data in XML.

**Steps:**

* Create a root element.
* Add child elements to store values.
* Use attributes to add metadata to elements.
* Save and validate the XML.

**Source Code:**

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

<students>

<student id="101" status="active">

<name>Sujan Tamang</name>

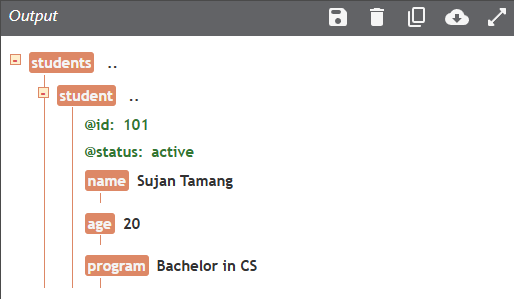
<age>20</age>

<program>Bachelor in CS</program>

</student>

</students>

**Output:**

****

**Conclusion:**

Elements hold content while attributes describe elements.

**LAB 7: XML Namespaces**

**Objective:**

* To avoid element name conflicts using XML namespaces.

**Steps:**

* Create elements with same names from different domains.
* Declare namespaces using xmlns.
* Qualify elements with prefixes.
* Validate using XML parser.

**Source Code:**

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

<root>

<h:table xmlns:h="http://www.sujanfruits.com">

<h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table xmlns:f="https://www.sujanbusiness.com/furniture">

<f:name>African Coffee Table</f:name>

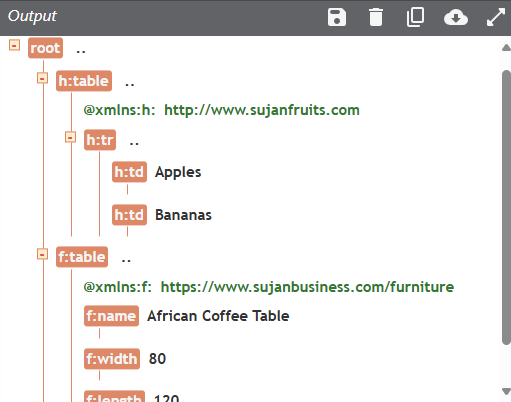
<f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

**Output:**

****

**Conclusion:**

Namespaces distinguish elements from different vocabularies.

**Lab 8: Private external DTDs**

Objective:

* To use external DTDs for validating XML documents.

Steps:

* Link an XML file to an external DTD using SYSTEM.
* Define element rules in the DTD.
* Validate the XML using a parser.

**Source Code:**

<?xml version="1.0"?>

<!DOCTYPE note SYSTEM "note.dtd">

<note>

<to>ram</to>

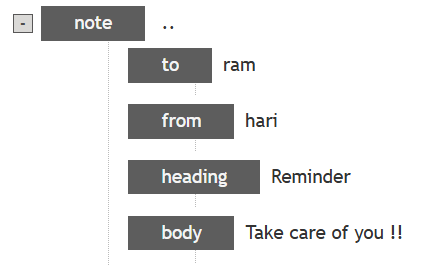
<from>hari</from>

<heading>Reminder</heading>

<body>Take care of you !!</body>

</note>

**Output:**

****

**Conclusion:**  
External DTDs allow for reusable and centralized validation rules for XML files.

**Lab 9: Public DTD**

**Objective:**

* To use public DTDs for XML validation.

**Steps:**

* Include a public DTD reference in the XML declaration.
* Write compliant XML.
* Validate using a parser.

**Source Code:**

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

<!DOCTYPE note PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<note>

<to>you</to>

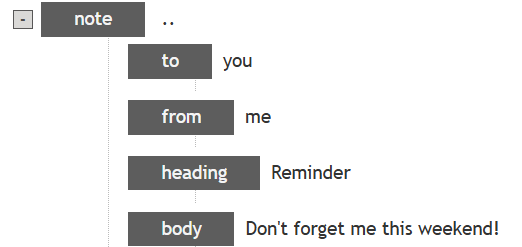
<from>me</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

**Output:**

****

**Conclusion:**  
Public DTDs provide a standard set of rules for XML validation that can be reused by different systems.

**Lab 10: Restrictions on Values**

**Objective:**

* To restrict the range of acceptable values in XML schema using minInclusive and maxInclusive.

**Steps:**

* Create an XSD schema with restrictions on numeric values.
* Validate the XML against the schema.
* Check for boundary value compliance.

**Source Code:**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="age">

<xs:simpleType>

<xs:restriction base="xs:integer">

<xs:minInclusive value="0"/>

<xs:maxInclusive value="120"/>

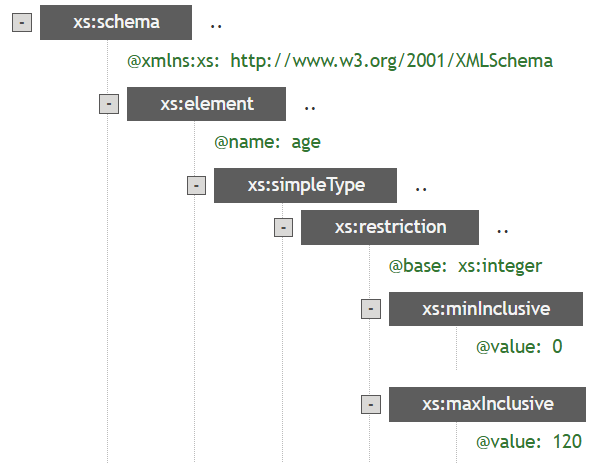
</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

**Output:**



**Conclusion:**  
This lab taught how to enforce numeric constraints within XML schemas.

**Lab 11: Restrictions on a Set of Values**

**Objective:**

* To restrict element values to a predefined list using enumeration in XSD.

**Steps:**

* Define an element in XSD with an enumeration restriction.
* List allowed values.
* Test XML documents for validation.

**Source Code:**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="car">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="Audi"/>

<xs:enumeration value="Golf"/>

<xs:enumeration value="BMW"/>

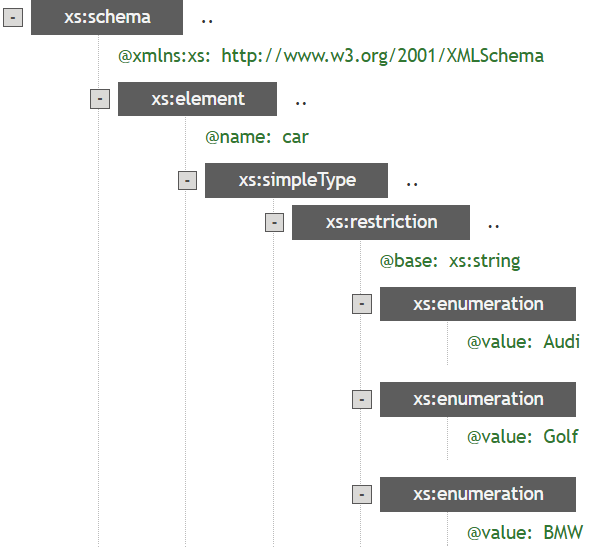
</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

Output:



**Conclusion:**  
Enumerations are useful for restricting input values to a fixed set, ensuring data consistency.

**Lab 12: Restrictions on a Series of Values**

Objective:

* To enforce string format rules using patterns in XML schema.

Steps:

* Define a pattern using regex in XSD.
* Apply it to an element.
* Validate the XML data.

**Source Code:**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="letter">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:pattern value="[a-z]"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

**Output:**

****

**Conclusion:**  
Patterns help validate string formats such as IDs, names, or codes.

**Lab 13: Restrictions on Whitespace Characters**

**Objective:**

* To control whitespace behavior in XML schema using whiteSpace facet.

**Steps:**

* Add whitespace restriction in XSD.
* Observe how parser treats spaces or line breaks.
* Validate results.

**Source Code:**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="person">

<xs:simpleType>

<xs:restriction base="xs:integer">

<xs:whiteSpace value="pre"/>

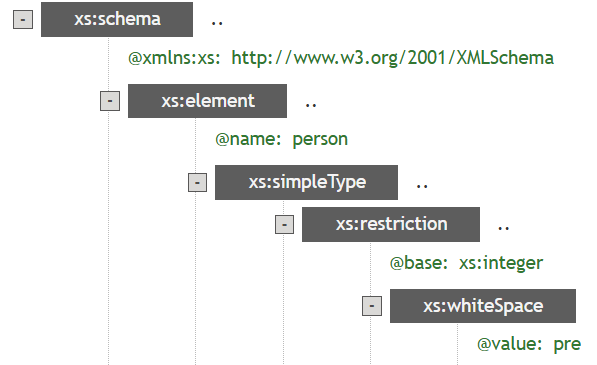
</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

**Output:**

****

**Conclusion:**  
Whitespace control is important for preserving or normalizing textual content.

**Lab 14: Restrictions on Length**

**Objective:**

* To enforce data length constraints in XML schema.

**Steps:**

* Use length, minLength or maxLength in schema definition.
* Validate with XML inputs.
* Observe validation errors if lengths mismatch.

**Source Code:**

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="code">

<xs:simpleType>

<xs:restriction base="xs:float">

<xs:length value="9"/>

</xs:restriction>

</xs:simpleType>

</xs:element>

</xs:schema>

**Output:**

****

**Conclusion:**  
This ensures data fields adhere to fixed-length requirements like codes, PINs, etc.

**Lab 15: Order Indicators**

**Objective:**

* To learn the use of order indicators in XSD: <all>, <choice>, and <sequence>.

**Steps:**

* Define elements using all indicators.
* Create example XML.
* Validate how they affect ordering.

**Source Code:**

a) All Indicator

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="person">

<xs:complexType>

<xs:all>

<xs:element name="name" type="xs:string"/>

<xs:element name="surname" type="xs:string"/>

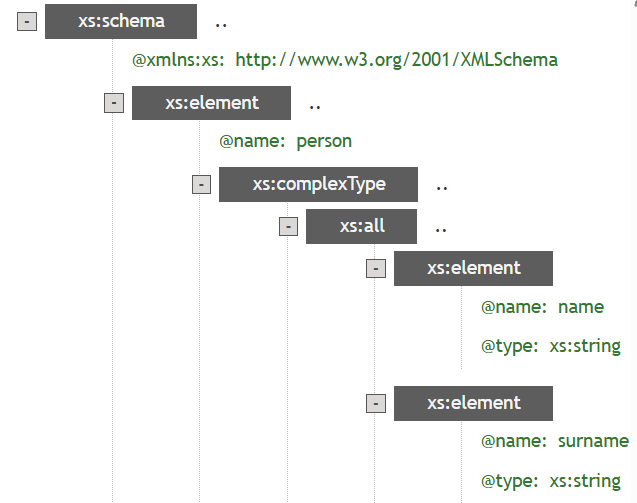
</xs:all>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



b) Choice Indicator

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="employ" type="xs:string"/>

<xs:complexType name="employ">

<xs:sequence>

<xs:element name="identity" type="xs:int"/>

<xs:element name="dep" type="xs:string"/>

</xs:sequence>

</xs:complexType>

<xs:element name="person">

<xs:complexType>

<xs:sequence>

<xs:element ref="h3" minOccurs="0"/>

<xs:choice>

<xs:element name="staff" type="employ"/>

<xs:element name="worker" type="worker"/>

</xs:choice>

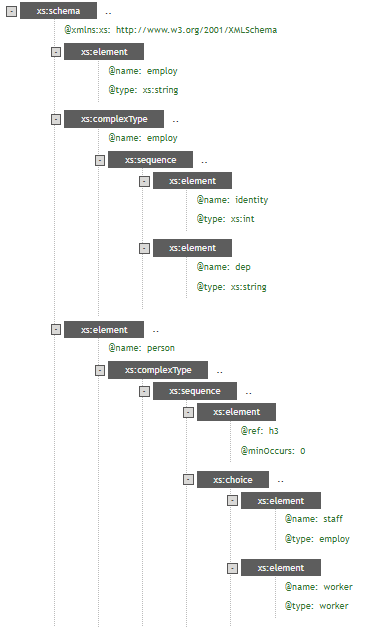
</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**

****

c) Sequence Indicator

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="person">

<xs:complexType>

<xs:sequence>

<xs:element name="name" type="xs:string"/>

<xs:element name="surname" type="xs:string"/>

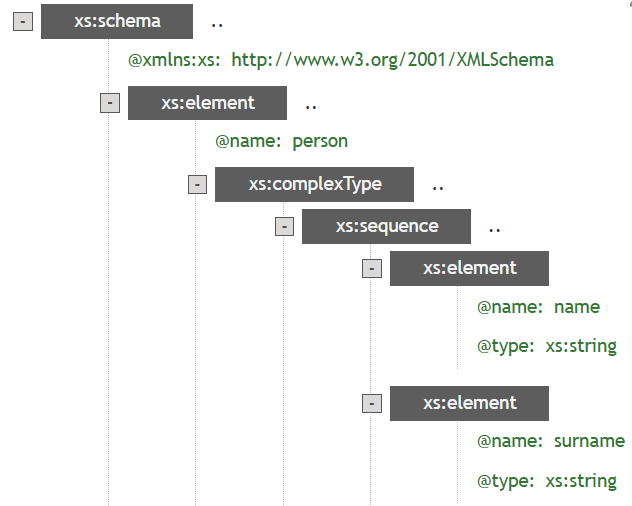
</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



**Conclusion:**  
Order indicators determine how child elements appear — fixed, optional, or unordered.

**Lab 16: Occurrence Indicators**

**Objective:**

* To control element repetition using minOccurs and maxOccurs attributes in XSD.

**Steps:**

* Create an XSD schema with these indicators.
* Write XML with repeated elements.
* Validate how parser enforces occurrence limits.

**Source Code:**

a) maxOccurs Indicator

<?xml version="1.0" encoding="ISO-8859-1"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

elementFormDefault="qualified">

<xs:element name="persons">

<xs:complexType>

<xs:sequence>

<xs:element name="person" maxOccurs="unbounded">

<xs:complexType>

<xs:sequence>

<xs:element name="full\_name" type="xs:string"/>

<xs:element name="child\_name" type="xs:string" minOccurs="0"

maxOccurs="5"/>

</xs:sequence>

</xs:complexType>

</xs:element>

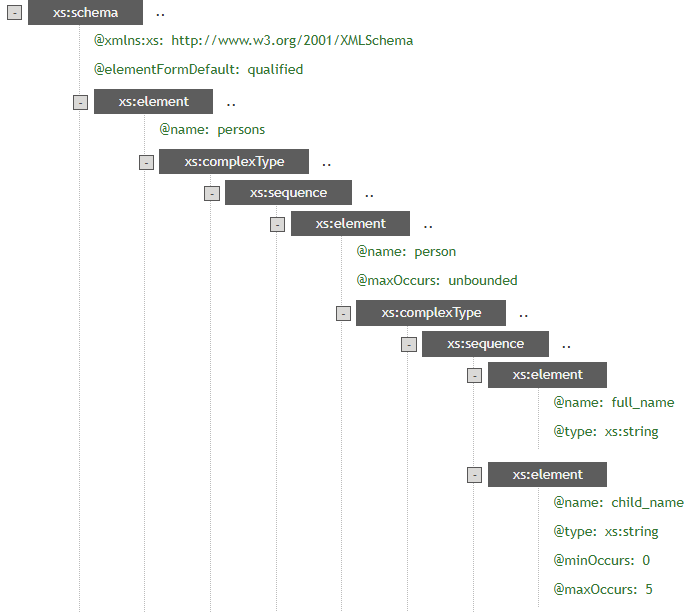
</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



b) minOccurs Indicator

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="person">

<xs:complexType>

<xs:sequence>

<xs:element name="name" type="xs:string"/>

<xs:element name="childrens name" type="xs:string" maxOccurs="8" minOccurs="0"/>

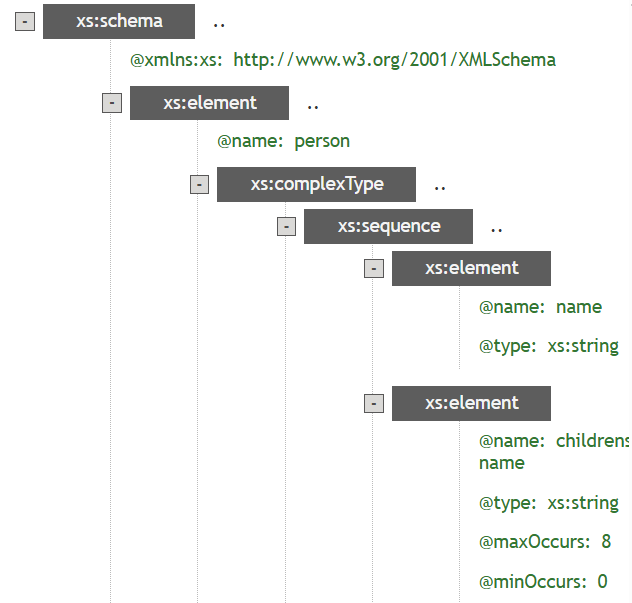
</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



**Conclusion:**  
These indicators help control how many times elements can appear in XML.

**Lab 17: Group Indicators**

**Objective:**

* To group related XML elements and attributes using elementGroup and attributeGroup in XSD.

**Steps:**

* Define groups in schema.
* Reference them inside complex types.
* Observe reusability and structure clarity.

**Source Code:**

a) Element Groups:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:group name="persongroup">

<xs:sequence>

<xs:element name="name" type="xs:string"/>

<xs:element name="lastname" type="xs:string"/>

<xs:element name="birth day" type="xs:date"/>

</xs:sequence>

</xs:group>

<xs:element name="person">

<xs:complexType>

<xs:sequence>

<xs:group ref="persons group"/>

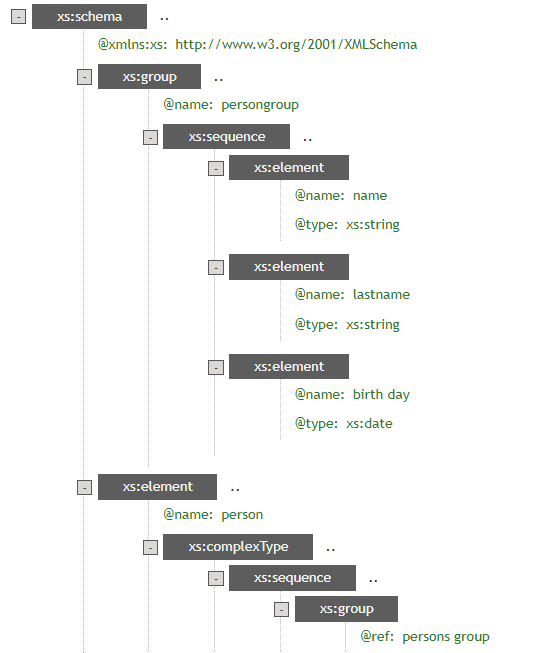
</xs:sequence>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



b) Attribute Groups:

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:attributeGroup name="person attrgroup">

<xs:attribute name="name" type="xs:string"/>

<xs:attribute name="surname" type="xs:string"/>

<xs:attribute name="birth day" type="xs:date"/>

</xs:attributeGroup>

<xs:element name="person">

<xs:complexType>

<xs:attributeGroup ref="person attrgroup"/>

</xs:complexType>

</xs:element>

</xs:schema>

**Output:**



**Conclusion:**  
Grouping simplifies schema design by modularizing similar elements or attributes.

**Lab 18: XSLT Code**

**Objective:**

* To transform XML data into HTML using XSLT.

**Steps:**

* Write an XSLT stylesheet with templates.
* Apply it to an XML file.
* Generate an HTML table from XML data.

**Source Code:**

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

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">

<html>

<body>

<h2>My collection</h2>

<table border="1">

<tr bgcolor="#9acd32">

<th style="text-align:left">Title</th>

<th style="text-align:left">Artist</th>

</tr> <xsl:for-each select="catalog/cd"> <tr>

<td><xsl:value-of select="title"/></td>

<td><xsl:value-of select="artist"/></td>

</tr>

</xsl:for-each>

</table>

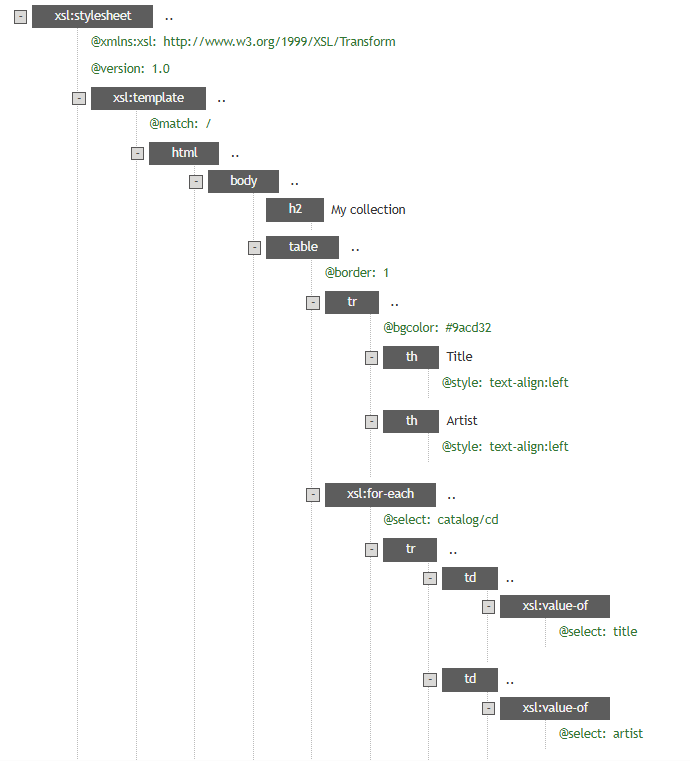
</body>

</html>

</xsl:template>

</xsl:stylesheet>

**Output:**



**Conclusion:**  
XSLT allows converting XML into user-friendly formats like HTML.

**Lab 19: XPATH Functions**

**Objective:**

* To use XPath to navigate through elements and attributes in XML.

**Steps:**

* Write an XML document.
* Use XPath queries to select specific nodes.
* Display queried data.

**Source Code:**

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

<bookstore>

<book>

<title lang="en">lord of rings</title>

<author>J.R.R. Tolkien</author>

<year>1892</year>

<price>19.99</price>

</book>

</bookstore>

**Output:**



**Conclusion:**  
XPath enables precise querying of XML content based on paths and conditions.

**Lab 20: XQuery Functions**

**Objective:**

* To perform querying and sorting on XML data using XQuery.

**Steps:**

* Load an XML file.
* Use XQuery to extract and order elements.
* Return results in XML or HTML.

**Source Code:**

<ul>{

for $x in doc("books.xml")/bookstore/book/title

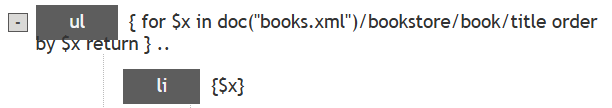
order by $x

return <li>{$x}</li>

}

</ul>

**Output:**



**Conclusion:**  
XQuery is powerful for data extraction and restructuring from XML sources.

**Lab 21: XQuery Conditional Expressions**

**Objective:**

* To use XQuery conditional expressions to filter and transform XML data.

**Steps:**

* Load an XML document.
* Write a conditional XQuery expression.
* Display elements based on their category.

**Source Code:**

<movies>{

for $x in doc("movies.xml")/moviestore/movie

return

if ($x/@category eq "children") then

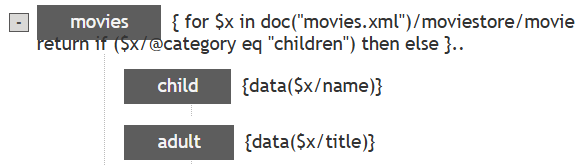
<child>{data($x/name)}</child>

else

<adult>{data($x/title)}</adult>

}</movies>

**Output:**



**Conclusion:**  
XQuery allows condition-based selection and transformation of XML data.

**LAB 22: XML DOM**

**Objective:**

* To parse XML using the Document Object Model (DOM) in JavaScript.

**Steps:**

* Create an XML string or file.
* Use DOM methods to read and manipulate elements.
* Output the desired information from the document.

**Source Code:**

<?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>

</bookstore>

**Output:**



**Conclusion:**  
This lab helped understand how the DOM API is used to navigate and extract data from XML documents.

**Lab 23: XML Parser**

**Objective:**

* To parse XML content dynamically using JavaScript and DOMParser.

**Steps:**

* Write JavaScript code to parse a string of XML.
* Access elements via DOM methods.
* Display selected content.

**Source Code:**

<!DOCTYPE html>

<html>

<body>

<p id="dome"></p>

<script>

var parser, xmlDoc;

var text = "<bookstore><book>" +

"<title>Everyday Italian</title>" +

"<author>Giada De Laurentiis</author>" +

"<year>2005</year>" +

"</book></bookstore>";

parser = new DOMParser();

xmlDoc = parser.parseFromString(text, "text/xml");

document.getElementById("demo").innerHTML =

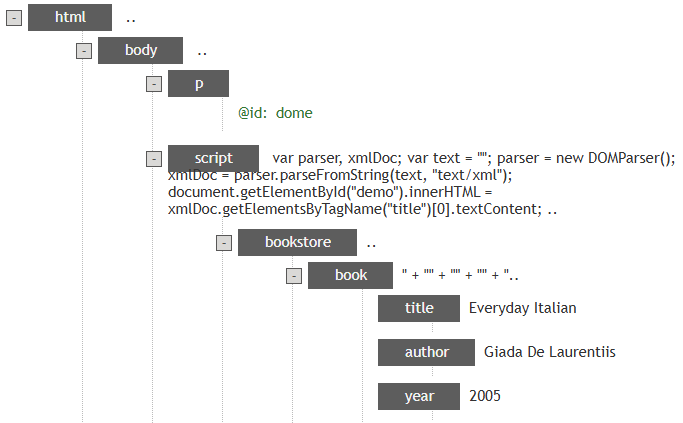
xmlDoc.getElementsByTagName("title")[0].textContent;

</script>

</body>

</html>

**Output:**

****

**Conclusion:**  
Client-side XML parsing lets web apps manipulate and render XML data.