Free Online XML Formatter - FreeFormatter.com

🎯 Title: Understand XML Schema

✅ What is XML Schema (XSD)?

An **XML Schema** (also known as **XSD - XML Schema Definition**) defines the **structure and rules** for an XML document.
It acts like a **blueprint** or **contract** that tells:

- What **elements and attributes** are allowed
- What **order** they should appear in
- What **data types** they should have (e.g., string, number, date, etc.)
- Whether they are **optional or required**
- Any **restrictions** or **patterns** on the data

Think of it like form validation: XML Schema makes sure the XML document is **correct** and **follows rules**.

□ Purpose of XML Schema

An XML Schema is used to:

- Define **elements and attributes** that can appear in the XML
- Specify the **order** and **number** of child elements
- Assign **data types** (like string, int, date, etc.)
- Set **default or fixed values**

🔐 Why is this useful?

- Ensures **data correctness**
- Prevents **invalid data**
- Makes it easier to process and transform data

📄 Example XML Schema (XSD)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Explanation:

- `<xs:schema>`: Root element of the schema file
- `<xs:element name="note">`: Defines the root XML element `<note>`
- `<xs:complexType>`: Means that `<note>` contains other elements (not just text)
- `<xs:sequence>`: The elements inside `<note>` must appear in this **exact order**
- Each child element (like `to`, `from`, `heading`, `body`) is of type `xs:string` (text)

📝 Corresponding XML Document

```
<?xml version="1.0"?>
<note>
  <to>Ravi</to>
  <from>Raja</from>
  <heading>Reminder</heading>
  <body>Don't forget meeting this weekend!</body>
</note>
```

This XML is **valid** because:

- It has the required elements
- They appear in the correct order
- The data types match the schema (all strings)

⚙️ Extended Example with Namespace

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="https://www.w3schools.com"
           xmlns="https://www.w3schools.com"
           elementFormDefault="qualified">
  <xs:element name="note">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

What's new here?

- `targetNamespace`: Adds a unique identifier for this schema (helps when combining schemas)
- `xmlns`: Declares the namespace being used
- `elementFormDefault="qualified"`: Means all elements must be namespace-qualified in the XML document

---

🗀 Notes

- The **file extension** for an XML Schema is `.xsd`
- XML Schema files themselves use **XML syntax**
- XML Schemas are widely used in **data communication** for validating XML-based messages (like in web services)

📊 Summary: XML Document vs XML Schema

Feature		XML Document	XML Schema (XSD)
Purpose		Holds actual data	Defines structure and rules of the data
Example Root Tag		`<note>`	`<xs:schema>`
File Extension		`.xml`	`.xsd`
Validated By		Schema (XSD)	XSD is not validated; it **is** the rule

---

🗄 Why Use XML Schema?

- ✅ Enforce **structure** and **rules**
- ✅ Ensure **data integrity**
- ✅ Enable **data validation** automatically
- ✅ Support **data types**, **patterns**, and **restrictions**

- ✅ Used heavily in **web services** (SOAP, WSDL)

🔍 Deeper Concepts in XML Schema

---

1. ✅ Simple Types vs Complex Types

Simple Types

- Elements or attributes that **contain only text**
- No child elements or attributes
- Example:

```
<xs:element name="age" type="xs:integer"/>
```

Complex Types

- Elements that **contain child elements** or **attributes**
- You define the structure using `<xs:complexType>`
- Example:

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="age" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

---

2. □ Element Types

You can define:

- **Global elements** (usable anywhere in the schema)
- **Local elements** (usable only within a specific parent)

Global Element Example:

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

Local Element Example:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/> <!-- local -->
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

---

3. 🔄 Occurrence Constraints

You can control **how many times an element can occur** using:

- `minOccurs` → Minimum number of times (default: 1)
- `maxOccurs` → Maximum number of times (default: 1 or use `"unbounded"`)

Example:

```
<xs:element name="phone" type="xs:string" minOccurs="0" maxOccurs="3"/>
```

This means:

- Phone number is **optional**
- Can occur up to **3 times**

---

4. □ Data Types in XSD

XSD supports many **built-in types**, such as:

- `xs:string`
- `xs:integer`
- `xs:decimal`
- `xs:date`
- `xs:time`
- `xs:boolean`

You can also create **custom types**.

Example: Custom Type with Restriction

```
<xs:simpleType name="AgeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="18"/>
    <xs:maxInclusive value="99"/>
  </xs:restriction>
</xs:simpleType>
```

Then use it:

```
<xs:element name="age" type="AgeType"/>
```

---

5. 🔍 Facets (Restrictions on Data)

Facets are used to **limit** or **restrict** values.

Common Facets:

- `minInclusive` / `maxInclusive` – for numeric limits
- `minLength` / `maxLength` – for string lengths
- `pattern` – regex for string format
- `enumeration` – a fixed set of allowed values

Example: Enumeration

```
<xs:simpleType name="GenderType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Male"/>
    <xs:enumeration value="Female"/>
    <xs:enumeration value="Other"/>
  </xs:restriction>
</xs:simpleType>
```

---

6. Attributes in XML Schema

You can add **attributes** to elements.

Example:

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Here, `isbn` is a **required attribute** of the `<book>` element.

---

7. Groups and Reuse

If you want to **reuse a group of elements**, use `xs:group` or define named types.

Element Group:

```
<xs:group name="personInfo">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="email" type="xs:string"/>
  </xs:sequence>
</xs:group>

<!-- Use it -->
<xs:element name="employee">
  <xs:complexType>
    <xs:group ref="personInfo"/>
  </xs:complexType>
```

```
</xs:element>
```

---

8. □ Inheritance in XML Schema

XSD allows **extension** and **restriction** of types (similar to OOP).

Example: Extension

```
<xs:complexType name="Person">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Employee">
  <xs:complexContent>
    <xs:extension base="Person">
      <xs:sequence>
        <xs:element name="salary" type="xs:decimal"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Now, `Employee` inherits from `Person`.

---

9. 🧵 Namespace and elementFormDefault

- `targetNamespace`: Used to uniquely identify the schema
- `elementFormDefault="qualified"`: All elements must be namespace-qualified

---

10. □ Tools to Work with XML and XSD

- **Editors**: oXygen XML Editor, XMLSpy, VS Code with XML plugins
- **Validators**: Online tools like:
    - https://www.freeformatter.com/xml-validator-xsd.html
    - https://www.xmlvalidation.com/
- **Converters**:
    - XML to XSD generator tools
    - XSD to Java classes (with JAXB)

🧠 Summary

Feature	Description
`xs:element`	Defines an element
`xs:complexType`	Elements with child elements or attributes
`xs:simpleType`	Restrict base types
`minOccurs`, `maxOccurs`	How many times an element can appear
`xs:attribute`	Add attribute to an element
`xs:restriction`	Set rules (facets)
`xs:enumeration`	Fixed allowed values
`xs:pattern`	Regular expressions for data
`xs:extension`	Inheritance for reusability
Namespaces	Organize elements/avoid conflicts

📌 1. Visual Diagram: XML vs XSD

Here's a **conceptual diagram** (explained in text) to help you visualize the relationship:

```
XML Document                          XSD (Schema)

<note>                               <xs:element name="note">
  <to>Ravi</to>                       <xs:complexType>
  <from>Raja</from>                     <xs:sequence>
  <heading>Reminder</heading>             <xs:element name="to"
type="xs:string"/>
  <body>Don't forget...</body>            <xs:element name="from"
type="xs:string"/>
</note>                                   ...
                                        </xs:sequence>
                                      </xs:complexType>
                                    </xs:element>
```

- **Left**: the actual data (XML)
- **Right**: rules/structure (XSD)

📌 2. Real-World Use Case Example: Student Record System

XML Document (student.xml)

```
<student>
  <name>John Doe</name>
  <rollno>101</rollno>
  <department>Computer Science</department>
  <email>john@example.com</email>
</student>
```

XSD Schema (student.xsd)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="rollno" type="xs:integer"/>
        <xs:element name="department" type="xs:string"/>
        <xs:element name="email" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

✅ This structure ensures all students have:

- Name as text
- Roll number as a number
- Department as text
- Email in correct format

📌 3. Validating XML Against XSD in Code

🔗 A. Using Python (with `lxml` library)

Install:

```
pip install lxml
```

Python Code:

```
from lxml import etree

# Load XML and XSD
with open('student.xml', 'rb') as xml_file, open('student.xsd', 'rb') as
xsd_file:
    xml = etree.parse(xml_file)
    schema_doc = etree.parse(xsd_file)
    schema = etree.XMLSchema(schema_doc)

# Validate
if schema.validate(xml):
    print("XML is valid ✅")
else:
    print("XML is NOT valid ❌")
    print(schema.error_log)
```

🔗 B. Using Java (with JAXB or javax.xml)

```
import javax.xml.XMLConstants;
import javax.xml.transform.stream.StreamSource;
import javax.xml.validation.*;
```

```
import java.io.File;

public class XMLValidator {
    public static void main(String[] args) throws Exception {
        File schemaFile = new File("student.xsd");
        File xmlFile = new File("student.xml");

        SchemaFactory factory =
SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
        Schema schema = factory.newSchema(schemaFile);
        Validator validator = schema.newValidator();
        validator.validate(new StreamSource(xmlFile));

        System.out.println("XML is valid ✅");
    }
}
```

📌 4. Online XML & XSD Validation Tools

You can also **validate without code** using these tools:

- https://www.freeformatter.com/xml-validator-xsd.html
- https://xmlvalidation.com/

✅ Conclusion: Why Learn XSD?

Benefit	Description
Data Validation	Ensures data is correct and in expected format
Automation	Programs can auto-process valid XML
Standardization	Common schema shared across systems
Security	Rejects malformed or malicious XML
Integration	Used in web services (SOAP, WSDL, etc.)

📓 Full Guide to XSD (XML Schema Definition)

🧠 What Is XSD?

XSD defines the **structure and rules** of an XML document.
It is written in **XML syntax** and used to **validate** XML documents.

🔍 It tells:

- What **elements** and **attributes** can appear
- The **order** of elements
- The **data types** of elements/attributes
- Whether values are **required**, **optional**, or **repeated**
- Allowed **formats**, **patterns**, and **value ranges**

□ Basic Structure of XSD

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

- `xmlns:xs`: Declares the XSD namespace
- `xs:schema`: Root of the XSD

💡 Example XML + XSD

📄 XML Document (note.xml)

```
<note>
  <to>Alice</to>
  <from>Bob</from>
  <heading>Hello</heading>
  <body>This is a message.</body>
</note>
```

📜 XSD Document (note.xsd)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

✅ This ensures the XML has:

- A `<note>` element
- 4 child elements in a specific order
- All content as text (strings)

⚙️ Key XSD Concepts

✅ 1. Element Types

- **Simple**: Only text
- `<xs:element name="age" type="xs:integer"/>`
- **Complex**: Has child elements or attributes
- `<xs:complexType>...</xs:complexType>`

🔂 2. Order and Repetition

```
<xs:element name="phone" type="xs:string" minOccurs="0" maxOccurs="3"/>
```

- Can appear 0 to 3 times

🎯 3. Attributes

```
<xs:element name="book">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="isbn" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

- `isbn` is a required attribute on `<book>`

🔢 4. Data Types

Common types:

- `xs:string`
- `xs:integer`
- `xs:boolean`
- `xs:date`
- `xs:decimal`

📏 5. Restrictions (Facets)

Restrict values with:

- `minInclusive` / `maxInclusive`

- `pattern` (regex)
- `enumeration`

```
<xs:simpleType name="AgeType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="18"/>
    <xs:maxInclusive value="60"/>
  </xs:restriction>
</xs:simpleType>
```

Use it:

```
<xs:element name="age" type="AgeType"/>
```

📋 6. Enumeration Example

```
<xs:simpleType name="GenderType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Male"/>
    <xs:enumeration value="Female"/>
    <xs:enumeration value="Other"/>
  </xs:restriction>
</xs:simpleType>
```

🔗 7. Custom Types & Reuse

You can define reusable types:

```
<xs:complexType name="PersonType">
  <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="age" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="employee" type="PersonType"/>
```

🧬 8. Inheritance (Extension)

```
<xs:complexType name="EmployeeType">
  <xs:complexContent>
    <xs:extension base="PersonType">
      <xs:sequence>
        <xs:element name="salary" type="xs:decimal"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

🌐 9. Namespaces and elementFormDefault

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com"
           xmlns="http://example.com"
```

```
elementFormDefault="qualified">
```

- Ensures all elements belong to the defined namespace

📎 File Extensions

- `.xml` → XML document
- `.xsd` → XML Schema file

□ Validate XML with XSD

✅ Online Validator Tools:

- FreeFormatter
- XMLValidation

🤖 Validate XML Using Code

Python (lxml):

```
from lxml import etree
schema = etree.XMLSchema(file='schema.xsd')
xml_doc = etree.parse('data.xml')
print(schema.validate(xml_doc))  # True or False
```

Java (javax.xml.validation):

```
Validator validator = schema.newValidator();
validator.validate(new StreamSource(new File("data.xml")));
```

✅ Summary Table

Feature	Purpose
`xs:element`	Defines an element
`xs:complexType`	Element with children or attributes
`xs:sequence`	Defines the order of child elements
`xs:attribute`	Adds attributes to elements
`xs:simpleType`	Restricts base types
`enumeration`	Limits values to fixed list
`pattern`	Regex-based restriction
`extension`	Reuse types via inheritance
`minOccurs`	Minimum occurrences
`maxOccurs`	Maximum occurrences

Below is a **complete real-world example** that includes:

- Complex and simple types
- Attributes
- Enumerations
- Restrictions (facets)
- Occurrence rules
- Custom types
- Inheritance (extension)
- Namespaces

XML File: `student-record.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<student xmlns="http://example.com/student">
  <personalInfo gender="Male">
    <name>John Doe</name>
    <age>21</age>
    <email>john.doe@example.com</email>
  </personalInfo>
  <academicInfo>
    <department>Computer Science</department>
    <gpa>3.75</gpa>
  </academicInfo>
  <skills>
    <skill>Python</skill>
    <skill>Java</skill>
  </skills>
</student>
```

XSD File: `student-record.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/student"
           xmlns="http://example.com/student"
           elementFormDefault="qualified">

  <!-- Gender Enumeration -->
  <xs:simpleType name="GenderType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Male"/>
      <xs:enumeration value="Female"/>
      <xs:enumeration value="Other"/>
    </xs:restriction>
  </xs:simpleType>
```

```
  <!-- Age Restriction -->
  <xs:simpleType name="AgeType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="18"/>
      <xs:maxInclusive value="30"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Personal Info Complex Type -->
  <xs:complexType name="PersonalInfoType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="age" type="AgeType"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="gender" type="GenderType" use="required"/>
  </xs:complexType>

  <!-- Academic Info Complex Type -->
  <xs:complexType name="AcademicInfoType">
    <xs:sequence>
      <xs:element name="department" type="xs:string"/>
      <xs:element name="gpa" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>

  <!-- Skills List -->
  <xs:complexType name="SkillsType">
    <xs:sequence>
      <xs:element name="skill" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <!-- Root Element -->
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="personalInfo" type="PersonalInfoType"/>
        <xs:element name="academicInfo" type="AcademicInfoType"/>
        <xs:element name="skills" type="SkillsType" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

✅ What's Included in This Example?

Feature	Demonstrated In
`xs:element`	All elements
`xs:complexType`	`personalInfo`, `academicInfo`, `skills`, and `student`
`xs:simpleType`	`AgeType`, `GenderType`

`xs:restriction`	Age range, gender options
`xs:enumeration`	Gender values
`xs:attribute`	`gender` in `personalInfo`
`minOccurs`, `maxOccurs`	`skills` list
`targetNamespace`	Set to `http://example.com/student`
`elementFormDefault`	`qualified` for namespacing

🛠 Want to Test It?

Use a free online validator:
📎 https://www.freeformatter.com/xml-validator-xsd.html

Just:

1. Paste the XML
2. Upload or paste the XSD
3. Click “Validate”

📄 Title: Understand XML Schema (XSD)

🧠 What is an XML Schema?

An **XML Schema** (also called **XSD** – XML Schema Definition) defines the **structure** and **rules** for an XML document.

It helps to ensure the XML is:

- Well-formed (correct structure)
- Valid (matches the schema rules)

✅ Why Use XML Schema?

XML Schema allows you to:

- ✅ Define **which elements and attributes** must appear
- ✅ Specify the **data type** of each element (string, number, date, etc.)
- ✅ Set **rules** like how many times an element appears (e.g., once, multiple times)

- ✅ Ensure **correct format** using patterns, ranges, etc.
- ✅ Validate the **order of elements**

🖇️ Relationship: XML Document vs XML Schema

XML Document	XML Schema (XSD)
Contains actual data	Contains rules/structure of XML
Example: `<to>Ravi</to>`	Says: `<to>` must be a string
Can be processed or read	Used to validate if XML is correct
Human- or machine-readable	Mostly for validation and data contracts

---

💡 XML Document Example

```
<?xml version="1.0"?>
<note>
  <to>Ravi</to>
  <from>Raja</from>
  <heading>Reminder</heading>
  <body>Don't forget meeting this weekend!</body>
</note>
```

This is a sample XML document that has:

- Root element: `<note>`
- Child elements: `<to>`, `<from>`, `<heading>`, and `<body>`

📜 XML Schema (XSD) for the Above Example

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

🔍 Explanation:

- `<xs:schema>`: Root element of every XSD file.

- `xmlns:xs="http://www.w3.org/2001/XMLSchema"`: Declares the XML Schema namespace.
- `<xs:element name="note">`: Defines an element `<note>` in the XML.
- `<xs:complexType>`: Means `<note>` contains other elements (not just text).
- `<xs:sequence>`: Defines that the child elements must appear in the exact order listed.
- `<xs:element name="to" type="xs:string"/>`: `<to>` must contain text (string).

Schema with Namespace Example

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="https://www.w3schools.com"
           xmlns="https://www.w3schools.com"
           elementFormDefault="qualified">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="heading" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Additional Concepts:

- `targetNamespace`: Specifies the namespace the schema applies to.
- `elementFormDefault="qualified"`: Means all elements must be namespace-qualified.

File Extension

- XML Schema files use the extension: `.xsd`

Example:

- `note.xml` → Your XML data
- `note.xsd` → The rules that validate `note.xml`

Benefits of Using XML Schema

Feature	Benefit
Data types support	Validates strings, numbers, dates, etc.

Custom rules	Set formats, min/max values, etc.
Structure enforcement	Controls the order and count of elements
Reusability	Define and reuse types across elements
Automation	Tools can auto-generate forms/code

✅ Summary

- XML Schema (XSD) **defines the structure** of XML documents.
- It ensures **data consistency**, **accuracy**, and **validation**.
- XML uses **real data**, XSD defines **rules for the data**.
- You use `.xsd` files to check if your `.xml` files are valid.

✅ Use Case: Student Information System

We want to store student information in XML, and validate it using an XSD.

📄 `student.xml` — The XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<student xmlns="http://example.com/student">
  <id>101</id>
  <name>John Doe</name>
  <age>21</age>
  <email>john.doe@example.com</email>
  <gender>Male</gender>
  <course>Computer Science</course>
  <grade>A</grade>
</student>
```

📜 `student.xsd` — The XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/student"
           xmlns="http://example.com/student"
           elementFormDefault="qualified">

  <!-- Root element -->
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="id" type="xs:integer"/>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="age" type="xs:integer"/>
        <xs:element name="email" type="xs:string"/>
        <xs:element name="gender" type="GenderType"/>
```

```
        <xs:element name="course" type="xs:string"/>
        <xs:element name="grade" type="GradeType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Gender must be one of the listed values -->
  <xs:simpleType name="GenderType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Male"/>
      <xs:enumeration value="Female"/>
      <xs:enumeration value="Other"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Grade must be A, B, C, D, or F -->
  <xs:simpleType name="GradeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="A"/>
      <xs:enumeration value="B"/>
      <xs:enumeration value="C"/>
      <xs:enumeration value="D"/>
      <xs:enumeration value="F"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

🔍 Explanation

Element	Data Type	Description
`<id>`	`xs:integer`	Student ID, must be a number
`<name>`	`xs:string`	Full name of the student
`<age>`	`xs:integer`	Age of the student (must be a number)
`<email>`	`xs:string`	Student email address
`<gender>`	`GenderType`	Must be "Male", "Female", or "Other"
`<course>`	`xs:string`	Student's course name
`<grade>`	`GradeType`	Must be one of "A", "B", "C", "D", or "F"

□ How to Validate the XML Against XSD

You can validate your XML using:

✅ Online tool:

- https://www.freeformatter.com/xml-validator-xsd.html

Steps:

1. Copy-paste the **XML** in the first box

2. Copy-paste the **XSD** in the second box
3. Click "Validate XML"

✅ Using Python:

```
from lxml import etree

xml_file = 'student.xml'
xsd_file = 'student.xsd'

xml_doc = etree.parse(xml_file)
xsd_doc = etree.parse(xsd_file)

schema = etree.XMLSchema(xsd_doc)
print("Valid?" , schema.validate(xml_doc))
```

🗁 File Extensions

File	Purpose
`student.xml`	The XML data
`student.xsd`	The schema (validation)

✅ What You Learned

- How to create a real XML structure
- How to define rules in XSD
- How to validate XML against XSD
- Data types, enumerations, and complex types in action

✅ Use Case: Bookstore Inventory

We want to store data about books in a bookstore, including details like title, author, price, category, and optional information like discount.

🗎 `bookstore.xml` — The XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore xmlns="http://example.com/bookstore">
  <book category="Fiction" language="English">
    <title>The Alchemist</title>
    <author>Paulo Coelho</author>
    <price>12.99</price>
    <discount>10</discount>
```

```
  </book>

  <book category="Programming" language="English">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <price>39.95</price>
  </book>
</bookstore>
```

---

bookstore.xsd — The XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/bookstore"
           xmlns="http://example.com/bookstore"
           elementFormDefault="qualified">

  <!-- Root element -->
  <xs:element name="bookstore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" type="BookType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Complex type for a single book -->
  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
      <xs:element name="discount" type="DiscountType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="category" type="CategoryType" use="required"/>
    <xs:attribute name="language" type="xs:string" use="optional"/>
  </xs:complexType>

  <!-- Discount: Must be between 0 and 100 -->
  <xs:simpleType name="DiscountType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>

  <!-- Category: Must be one of these -->
  <xs:simpleType name="CategoryType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fiction"/>
      <xs:enumeration value="Non-Fiction"/>
      <xs:enumeration value="Programming"/>
      <xs:enumeration value="Science"/>
      <xs:enumeration value="History"/>
    </xs:restriction>
```

```
  </xs:simpleType>

</xs:schema>
```

📝 Breakdown of Concepts Used

Feature	Where It Appears	Description
Attributes	`category`, `language` on `<book>`	Extra info inside the tag
Complex Type	`BookType`, `bookstore`	Elements that contain other elements
Simple Type (Enum)	`CategoryType`	Limits values (like choices in dropdown)
Restrictions	`DiscountType`	Min/Max values allowed
Optional Element	`<discount>` with `minOccurs="0"`	Not required
Multiple Items	`<book>` with `maxOccurs="unbounded"`	Allows many books

✅ How It Looks Together

🗂️ Files

- `bookstore.xml` — Your actual data
- `bookstore.xsd` — The rules/schema

📄 XML Snippet

```
<book category="Programming" language="English">
  <title>Learning XML</title>
  <author>Erik T. Ray</author>
  <price>39.95</price>
</book>
```

🔍 It passes validation because:

- `category="Programming"` is a valid option
- `language` is optional
- `discount` is optional (not included here)
- All required child elements (`title`, `author`, `price`) are present

□ Try It Yourself

Validate the XML using:

✅ Online:

📎 https://www.freeformatter.com/xml-validator-xsd.html

Or

✅ Python:

```
from lxml import etree

xml = etree.parse("bookstore.xml")
xsd = etree.parse("bookstore.xsd")
schema = etree.XMLSchema(xsd)

print("Valid XML?" , schema.validate(xml))
```

✅ Option 1: Using Python (Recommended)

🔧 Step 1: Install Required Library

Open your terminal (or Command Prompt) and run:

```
pip install lxml
```

`lxml` is a powerful library for parsing XML and validating against XSD.

📝 Step 2: Save These Two Files

✅ `bookstore.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore xmlns="http://example.com/bookstore">
  <book category="Fiction" language="English">
    <title>The Alchemist</title>
    <author>Paulo Coelho</author>
    <price>12.99</price>
    <discount>10</discount>
  </book>

  <book category="Programming" language="English">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <price>39.95</price>
  </book>
</bookstore>
```

✅ `bookstore.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/bookstore"
           xmlns="http://example.com/bookstore"
           elementFormDefault="qualified">
```

```
  <xs:element name="bookstore">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="book" type="BookType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="author" type="xs:string"/>
      <xs:element name="price" type="xs:decimal"/>
      <xs:element name="discount" type="DiscountType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="category" type="CategoryType" use="required"/>
    <xs:attribute name="language" type="xs:string" use="optional"/>
  </xs:complexType>

  <xs:simpleType name="DiscountType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="100"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="CategoryType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Fiction"/>
      <xs:enumeration value="Non-Fiction"/>
      <xs:enumeration value="Programming"/>
      <xs:enumeration value="Science"/>
      <xs:enumeration value="History"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

□ Step 3: Create the Python Validation Script

✅ **validate.py**

```
from lxml import etree

# Load XML and XSD files
xml_file = "bookstore.xml"
xsd_file = "bookstore.xsd"

# Parse XML and XSD
xml_doc = etree.parse(xml_file)
xsd_doc = etree.parse(xsd_file)

# Create XML Schema object
schema = etree.XMLSchema(xsd_doc)
```

```
# Validate XML
is_valid = schema.validate(xml_doc)

print("Is XML valid?", is_valid)

# If invalid, print error log
if not is_valid:
    for error in schema.error_log:
        print(f"Line {error.line}: {error.message}")
```

▶️ Step 4: Run the Script

In your terminal, run:

```
python validate.py
```

You will see:

```
Is XML valid? True
```

If it's not valid, you'll get detailed errors with line numbers.

✅ Option 2: Use an XML Editor with XSD Support

You can also use editors like:

💻 VS Code (with XML extension)

1. Install the "XML Language Support by Red Hat" extension.
2. Open your XML file.
3. Add schema location like this:

```
<bookstore xmlns="http://example.com/bookstore"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://example.com/bookstore bookstore.xsd">
```

4. Save both `bookstore.xml` and `bookstore.xsd` in the same folder.
5. The editor will show validation results.

✅ Use Case: Employee Directory

We want to describe employee records in an organization, with data such as:

- Employee ID (as attribute)
- Name

- Department
- Email
- Phone number (optional)
- Employment type (Full-Time / Part-Time / Contract)
- Join date (must be in date format)

`employees.xml` — The XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<employees xmlns="http://example.com/employees">
  <employee empId="E101">
    <name>Jane Smith</name>
    <department>HR</department>
    <email>jane.smith@example.com</email>
    <employmentType>Full-Time</employmentType>
    <joinDate>2021-03-15</joinDate>
  </employee>

  <employee empId="E102">
    <name>Mark Johnson</name>
    <department>IT</department>
    <email>mark.johnson@example.com</email>
    <phone>1234567890</phone>
    <employmentType>Contract</employmentType>
    <joinDate>2022-01-10</joinDate>
  </employee>
</employees>
```

`employees.xsd` — The XML Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/employees"
           xmlns="http://example.com/employees"
           elementFormDefault="qualified">

  <!-- Root Element -->
  <xs:element name="employees">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="employee" type="EmployeeType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- Complex Type for Each Employee -->
  <xs:complexType name="EmployeeType">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
```

```
      <xs:element name="department" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
      <xs:element name="phone" type="xs:string" minOccurs="0"/>
      <xs:element name="employmentType" type="EmploymentType"/>
      <xs:element name="joinDate" type="xs:date"/>
    </xs:sequence>
    <xs:attribute name="empId" type="xs:string" use="required"/>
  </xs:complexType>

  <!-- Enumeration for Employment Type -->
  <xs:simpleType name="EmploymentType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Full-Time"/>
      <xs:enumeration value="Part-Time"/>
      <xs:enumeration value="Contract"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

□ Explanation

Element/Attribute	Type	Notes
`empId`	`xs:string`	Required attribute
`name`	`xs:string`	Employee's full name
`department`	`xs:string`	Department name
`email`	`xs:string`	Email address
`phone`	`xs:string`	Optional
`employmentType`	Enum	Only one of: Full-Time, Part-Time, Contract
`joinDate`	`xs:date`	Must be in `YYYY-MM-DD` format

🗁 Folder Structure

```
your-folder/
├── employees.xml
├── employees.xsd
├── validate.py (optional for Python)
```

☑ Want to Validate?

Use the same Python script (`validate.py`) I gave you earlier, just update file names:

```
xml_file = "employees.xml"
xsd_file = "employees.xsd"
```

Then run:

```
python validate.py
```

Great! You've shared a simple XML structure:

```
<employee>
    <firstname>Priya </firstname>
    <salary>45000.50</salary>
    <hiredata>2020-11-01</hiredata>
</employee>
```

Now let's create a **complete working example** to validate this XML using an **XSD (Schema)** and a `validate.py` script.

✅ Step 1: XML File — `employee.xml`

Save this as `employee.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee xmlns="http://example.com/employee">
    <firstname>Priya</firstname>
    <salary>45000.50</salary>
    <hiredata>2020-11-01</hiredata>
</employee>
```

✅ Step 2: XSD Schema — `employee.xsd`

Save this as `employee.xsd` in the same folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/employee"
           xmlns="http://example.com/employee"
           elementFormDefault="qualified">

  <xs:element name="employee">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstname" type="xs:string"/>
        <xs:element name="salary" type="xs:decimal"/>
        <xs:element name="hiredata" type="xs:date"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

✅ Step 3: Python Validation Script — `validate.py`

Save this as `validate.py`:

```
from lxml import etree

xml_file = "employee.xml"
xsd_file = "employee.xsd"

try:
```

```
    # Load and parse XSD
    with open(xsd_file, 'rb') as f:
        schema_doc = etree.parse(f)
        schema = etree.XMLSchema(schema_doc)

    # Load and parse XML
    with open(xml_file, 'rb') as f:
        xml_doc = etree.parse(f)

    # Validate XML
    is_valid = schema.validate(xml_doc)
    print("✅ XML is valid!" if is_valid else "❌ XML is invalid.")

    # Show detailed errors if invalid
    if not is_valid:
        for error in schema.error_log:
            print(f"Line {error.line}: {error.message}")

except Exception as e:
    print(f"Error: {e}")
```

▶️ Step 4: Run the Script

1. Open your terminal or PowerShell.
2. Navigate to the folder where these three files are saved.
3. Run the command:

```
python validate.py
```

You should see:

```
✅ XML is valid!
```

✅ Recap of Files

File	Purpose
`employee.xml`	The data
`employee.xsd`	Schema to validate the structure
`validate.py`	Python script to perform validation

working with a very simple XML that holds just a single value:

```
<?xml version="1.0" encoding="UTF-8"?>
<SCORE xmlns="http://example.com/score">85</SCORE>
```

And you want the **XSD Schema** to validate that:

- `SCORE` must be a **number**

- It must be **between 0 and 100**, inclusive

✅ Step 1: XML File — `score.xml`

Save this as `score.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<SCORE xmlns="http://example.com/score">85</SCORE>
```

✅ Step 2: XSD Schema File — `score.xsd`

Save this as `score.xsd`:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/score"
           xmlns="http://example.com/score"
           elementFormDefault="qualified">

  <xs:element name="SCORE">
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="100"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

</xs:schema>
```

✔ This schema ensures the value must be an **integer between 0 and 100**.

✅ Step 3: Python Validation Script — `validate.py`

If you don't already have it, here's the script:

```
from lxml import etree

xml_file = "score.xml"
xsd_file = "score.xsd"

try:
    # Load XSD
    with open(xsd_file, 'rb') as f:
        schema_doc = etree.parse(f)
        schema = etree.XMLSchema(schema_doc)

    # Load XML
    with open(xml_file, 'rb') as f:
        xml_doc = etree.parse(f)

    # Validate
```

```
    is_valid = schema.validate(xml_doc)
    print("✅ XML is valid!" if is_valid else "❌ XML is invalid.")

    if not is_valid:
        for error in schema.error_log:
            print(f"Line {error.line}: {error.message}")

except Exception as e:
    print(f"Error: {e}")
```

✅ Step 4: Test with Valid and Invalid Examples

✅ Valid Example

```
<SCORE xmlns="http://example.com/score">85</SCORE>  ✅ Passes
```

❌ Invalid Example (above 100)

```
<SCORE xmlns="http://example.com/score">120</SCORE>  ❌ Fails
```

❌ Invalid Example (below 0)

```
<SCORE xmlns="http://example.com/score">-5</SCORE>  ❌ Fails
```

validate an XML where the `<department>` element must only contain one of the following **specific values**:

- `HR`
- `IT`
- `Finance`

This is a perfect use case for an **enumeration** in XSD.

✅ Step 1: XML File — `department.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<department xmlns="http://example.com/department">HR</department>
```

✅ Step 2: XSD Schema File — `department.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://example.com/department"
           xmlns="http://example.com/department"
           elementFormDefault="qualified">

  <xs:element name="department">
    <xs:simpleType>
```

```
      <xs:restriction base="xs:string">
        <xs:enumeration value="HR"/>
        <xs:enumeration value="IT"/>
        <xs:enumeration value="Finance"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

</xs:schema>
```

✅ This ensures only `"HR"`, `"IT"`, or `"Finance"` are allowed.

✅ Step 3: Python Validation Script — `validate.py`

If you need the validation script too, here it is:

```
from lxml import etree

xml_file = "department.xml"
xsd_file = "department.xsd"

try:
    schema_doc = etree.parse(xsd_file)
    schema = etree.XMLSchema(schema_doc)
    xml_doc = etree.parse(xml_file)

    is_valid = schema.validate(xml_doc)
    print("✅ XML is valid!" if is_valid else "❌ XML is invalid.")

    if not is_valid:
        for error in schema.error_log:
            print(f"Line {error.line}: {error.message}")

except Exception as e:
    print(f"Error: {e}")
```

□ Try Invalid Examples:

❌ Invalid Value

```
<department xmlns="http://example.com/department">Marketing</department>
```

Will give output:

```
❌ XML is invalid.
Line 2: Element 'department': [facet 'enumeration'] The value 'Marketing' is not
an element of the set {'HR', 'IT', 'Finance'}.
```

✅ Summary

File	Purpose
`department.xml`	Contains department value
`department.xsd`	Ensures value is one of HR/IT/Finance

`validate.py`	Validates the XML against the XSD