Modern Markup for Data Interchange

Modern Markup for Data Interchange

Trainer's Guide

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2016

Aptech®
Unleash your potential

Aptech™
COMPUTER EDUCATION
Unleash your potential

“A little learning is a dangerous thing,
but a lot of ignorance is just as bad.”

Preface

The book 'Modern Markup for Data Interchange' Trainer's Guide aims to teach the fundamentals of XML and JSON. The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the module and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

Table of Contents

Sessions

“The future depends on what we do in the present.”

Session 1 - Introduction to XML

1.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth.

1.1.1 Objectives

By the end of this session, the learners will be able to:

- Introduction to XML
- Exploring XML
- Working with XML
- XML Syntax

1.1.2 Teaching Skills

To teach this session, you should be well-versed with XML and drawbacks of earlier markup languages that led to the development of XML. You should also aware yourself with the structure and lifecycle of the XML document, the XML syntax, and the various parts of the XML document.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

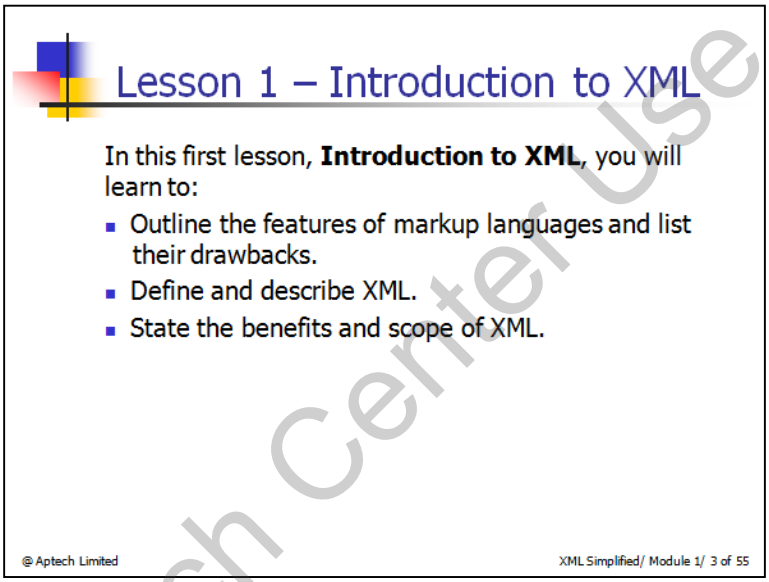
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces them to XML. They will learn about the drawbacks of earlier markup language that led to the development of XML and the structure and lifecycle of the XML document. They will also learn about the XML syntax and the various parts of the XML document.

1.2 In-Class Explanations

Slide 3

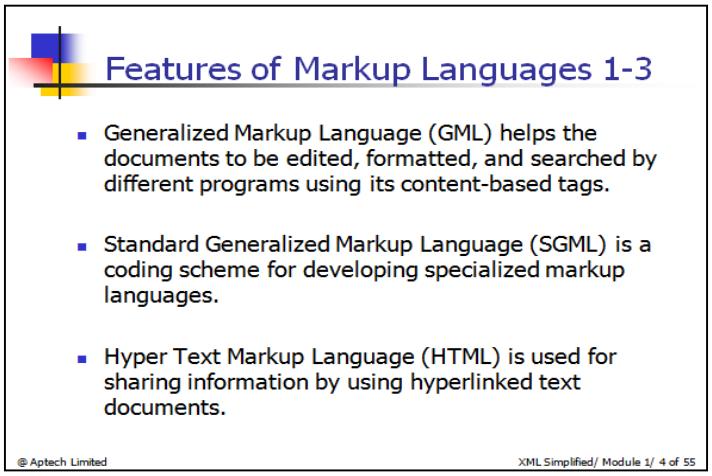
Let us understand the objectives of lesson 1.

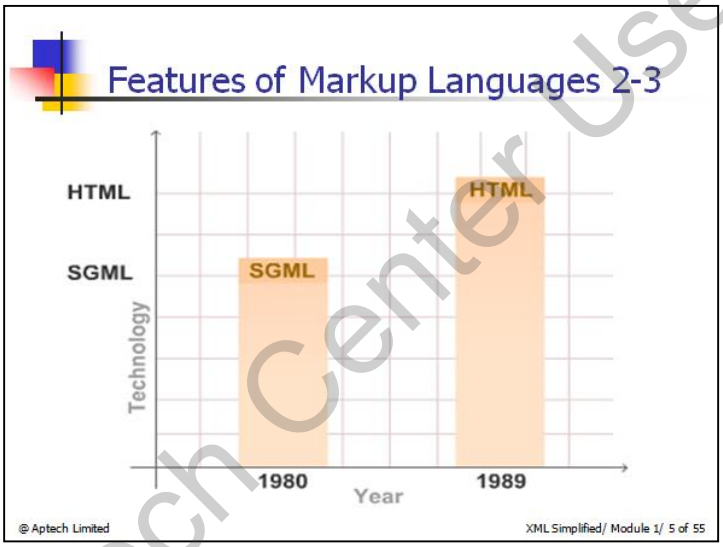


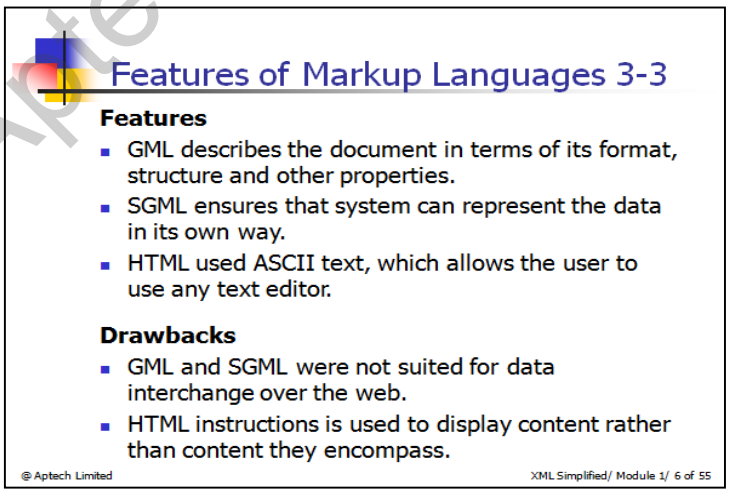
Using slide 3, describe the topics to be covered in the lesson.

Slides 4 to 6

Let us understand the features of markup languages.







Using slides 4 to 6, explain the features of markup languages.

Explain them that markup languages are used to define the meaning and organize the structure of a text document. Markup languages help the documents by giving them a new

look and formatting the text. Markup languages are expressed with the help of tags. They instruct the software that interprets the text to carry out some action on it.

Earlier markup languages were used in publishing practices which involved handwritten annotations in the form of symbols. Some early examples of computer markup languages available outside the publishing industry can be found in typesetting tools on UNIX systems such as troff and TEX.
In these systems, formatting commands were inserted into the document text so that typesetting software could format the text according to the editor's specifications.

Tips:
Markup was also commonly applied by editors, proofreaders, publishers, graphic designers, and indeed by document authors.

Explain them that based on the action taken on the text represented by markup languages; they are categorized into following categories:

- **Presentational Markup** – Focuses on the structure of the document. Examples include traditional word-processing systems that produce the What You See is What You Get (WYSIWYG) effect within the document text.

- **Procedural Markup** – Similar to presentation markup. In addition, they help the users to arrange the text. Examples of procedural markup systems include the creation of macros that can be defined and invoked by name.

- **Descriptive Markup** – Determines the content of the document. This means they are used to label the markup text, rather than providing instruction on how text should be processed. Example of such markup languages includes use of certain tags such as <cite> used in HTML language to label the text.

Creation of SGML Markup Language

In 1980, American National Standards Institute (ANSI) committee created Standard Generalized Markup Language (SGML), an all-encompassing coding scheme and flexible toolkit for developing specialized markup languages.

SGML is the successor to Generalized Markup Language (GML). In 1986, International Organization for Standardization (ISO) acquired it as a standard. SGML is a meta language as other languages are created from it. SGML has a syntax to include markup in documents as well as the tags describing what markup is allowed to do in the document.

SGML application consists of SGML declaration and SGML Document Type Definition (DTD). SGML found wide acceptance and use in fields with very large-scale documentation requirements.

Creation of HTML Markup Language

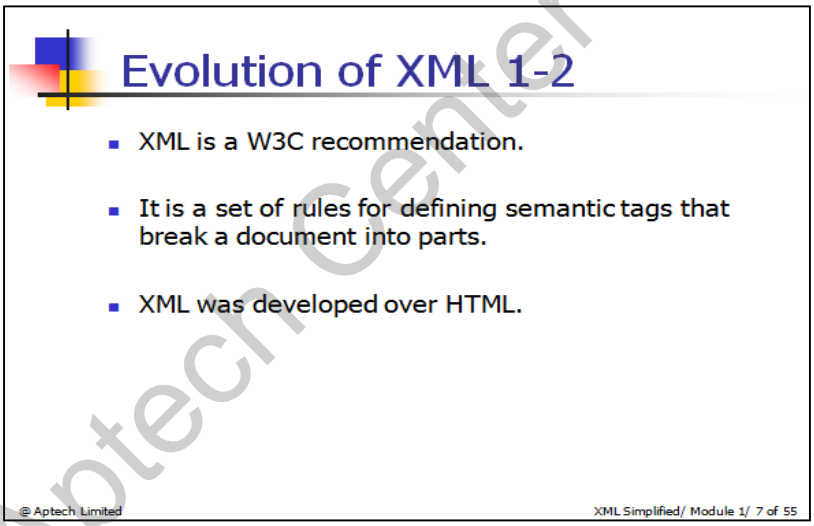
In 1989, HyperText Markup Language (HTML), a technology for sharing information by using hyperlinked text documents was developed. HTML was created from SGML. In the early years, it was extensively accessed by scientists and technicians. HTML was originally created to mark up technical papers, so that they could be transferred across different platforms. However, with time, it began to be used for marking up non-technical documents too. As the use of the Internet became popular, browser manufacturers started developing different tags to display documents with more creativity.

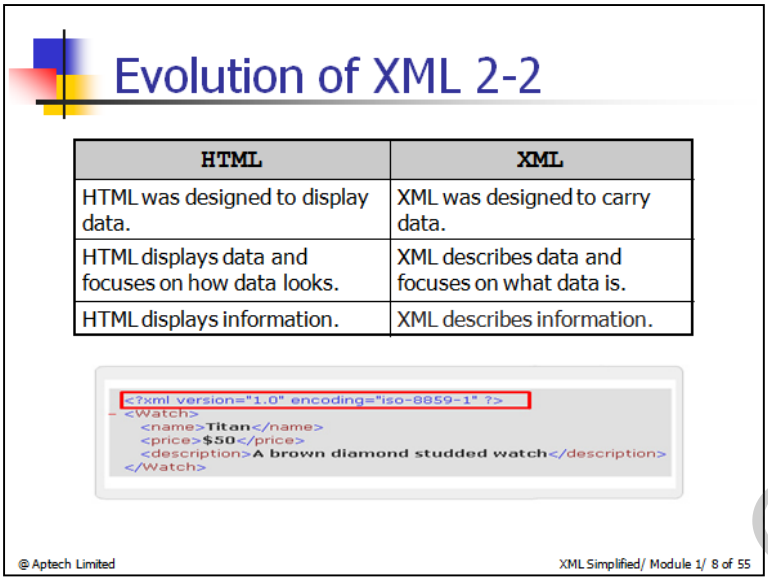
HTML became the main markup language for creating Web pages and other information that can be displayed in a Web browser, and is quite likely the most widely used markup language in the world today.

Explain the evolution of markup languages displayed on slide 5 and then explain the features and drawbacks of earlier Markup languages as described on slide 6.

Slides 7 and 8

Let us understand the evolution of XML.





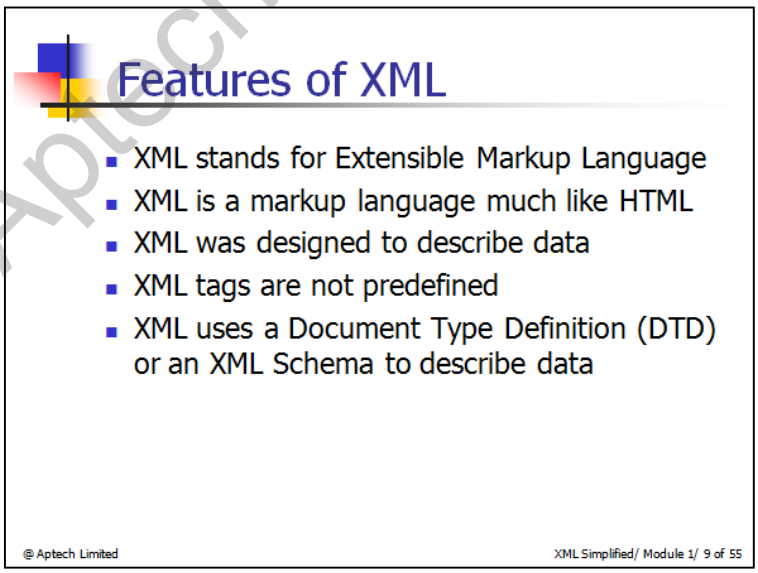
Using slides 7 and 8, explain the evolution of XML.

eXtensible Markup Language (XML) is a meta markup language developed by the World Wide Web Consortium. The main purpose of creating XML language was its use in exchanging the information between the applications running in different business environments. XML is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document.

Explain them that the basic differences between HTML and XML markup languages as listed in the table on slide 8. Also, show the basic XML document.

Slide 9

Let us understand the features of XML.



Using slide 9, explain the features of XML.

Explain them though the writing style of XML is similar to HTML language, but it was created to structure, store, and transport information.

For example, book publishing software wants to describe the book details using XML. The generated XML document will be as follows:

```
<book>
<title> Learning XML </title>
<author> Jon Bosak </author>
<price> 200 </price>
</book>
```

This XML file is very much self-descriptive. The information wrapped in tags of XML document is received by software that can read, process, and use the information further in the application.

Tell them that XML is designed to describe data. For example, <title> tag describes the name of the book in the discussed XML. Similarly, <author> and <price> tag describe the name of the author and number of pages in the book.

Tell them that XML tags are not predefined like HTML, rather the tags are described by the author of the XML document. This is because XML tags describes the document, whereas HTML tags are used for presentation.

Tell them that XML uses a DTD or an XML Schema to describe the data.

In-Class Question:

After you finish the feature of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

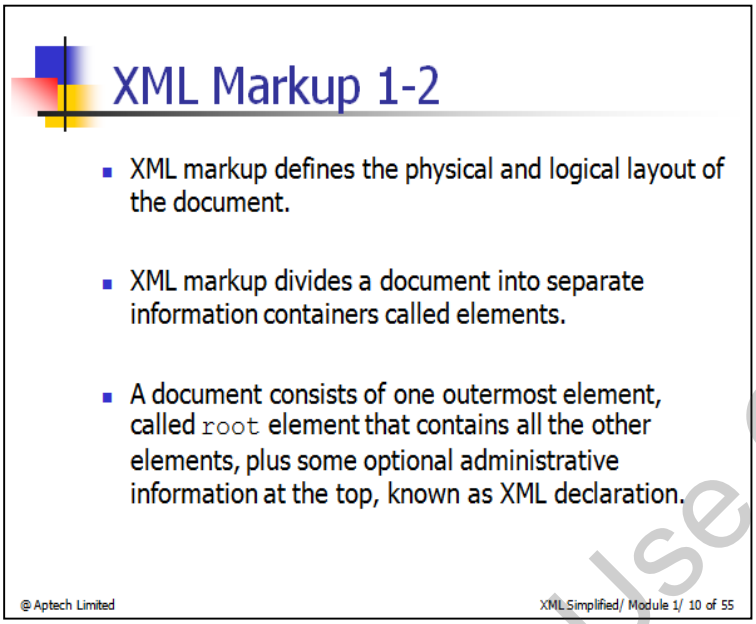
How can you use an XML document in application development?

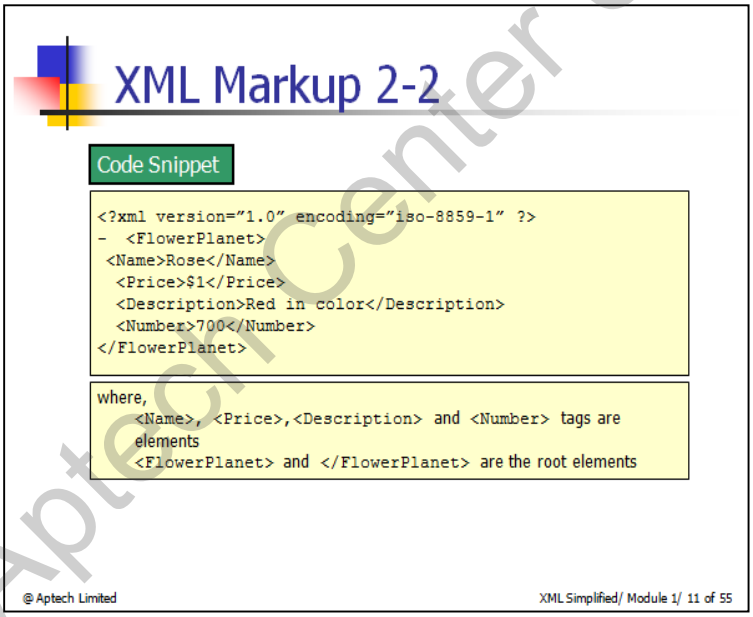
Answer:

XML can be used for data storage and sharing the data by different applications.

Slides 10 and 11

Let us understand the representation of XML markup.





Using slides 10 and 11, explain the representation of XML markup.

XML markup defines the physical and logical layout of the document. XML can be considered as an information container. It employs a tree-based structure to represent a document. The markup divides the information into a hierarchy of character data and container-like elements and its attributes. All elements can have text content and attributes similar to HTML.

XML documents must contain a **root element**. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements) as shown in the XML template file:

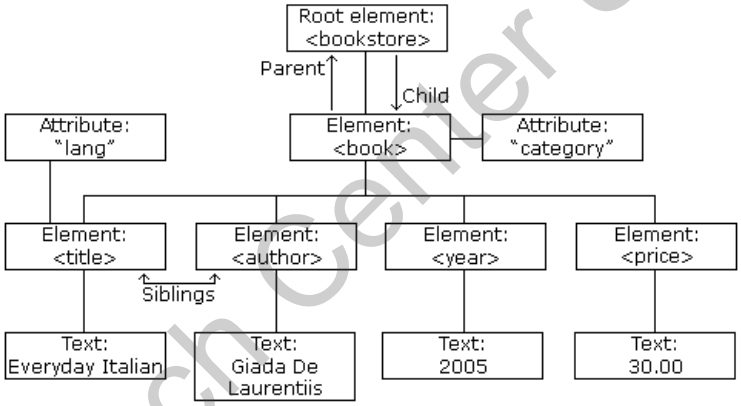
```
<root>
  <child>
    <sub-child>.....</sub-child>

    <sub-child>.....</sub-child>

  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

Following figure shows the tree structure representation of XML containing book information:



The usage of XML can be observed in many real-life scenarios. It can be used in the fields of information sharing, single application usage, content delivery, re-use of data, separation of data and presentation, semantics, and so forth. News agencies are a common place where XML is used. News producers and news consumers often use a standard specification named XMLNews to produce, retrieve, and relay information across different systems in the world.

Then, explain the XML code provided on slide 11.

In-Class Question:

After you finish the representation of XML markup, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Consider that a bookstore application wants to store the details of books available in the stock. Now, what can be the root element for storing books information?

Answer:

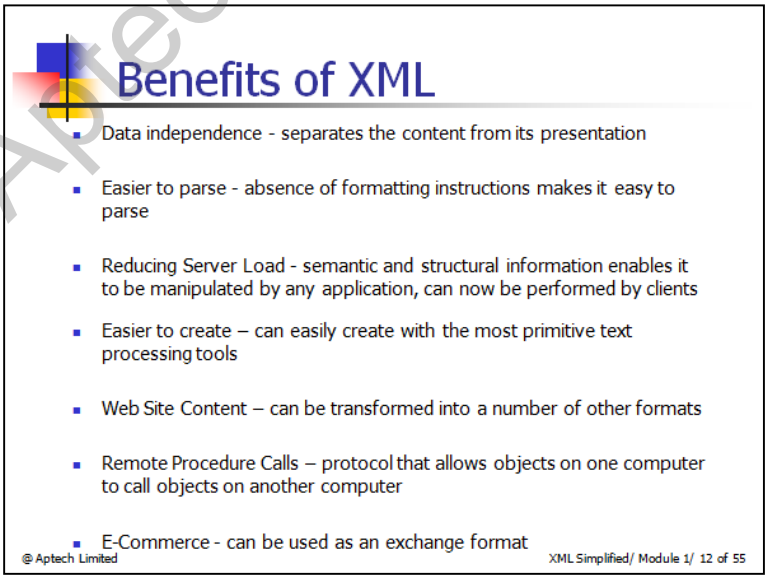
To create a bookstore information file in XML, the root element can be named as:

```
<bookstore>
        <book>
            . . .
        </book>
        <book>
           .   . .
       </book>
</bookstore>
```

XML can be used for data storage and sharing the data by different applications.

Slide 12

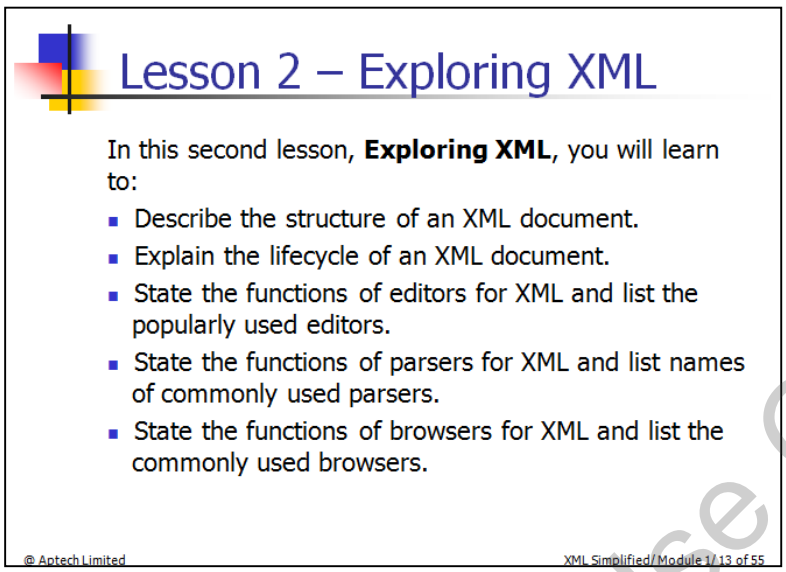
Let us understand the benefits of XML.



Using slide 12, explain the benefits of XML.

Slide 13

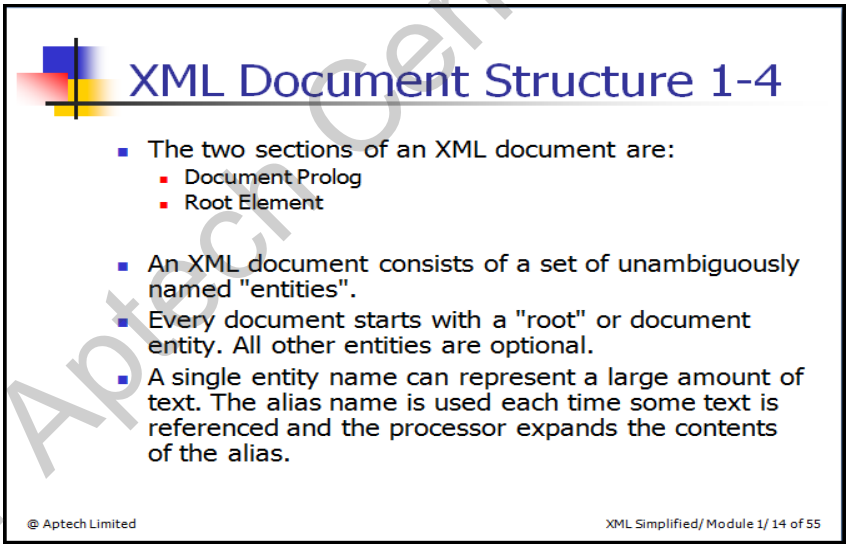
Let us understand the objectives of lesson 2.



Using slide 13, describe the topics covered in lesson 2.

Slides 14 to 17

Let us understand the XML document structure.



XML Document Structure 2-4

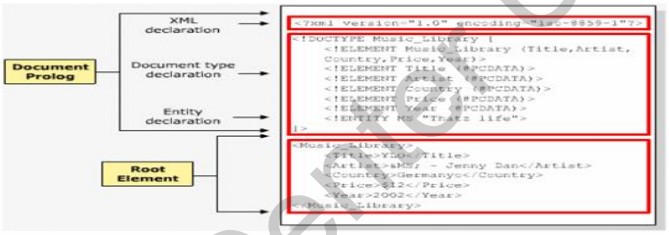
Document Prolog

- Document prolog contains metadata and consists of two parts - XML Declaration and Document Type Declaration.
- XML Declaration specifies the version of XML being used.
- Document Type Declaration defines entities' or attributes' values and checks grammar and vocabulary of markup.

Root Element

- Also called a document element.
- It must contain all the other elements and content in the document.
- An XML element has a start tag and end tag.

XML Document Structure 3-4



XML Document Structure 4-4

Code Snippet

```
<?xml version="1.0" encoding="iso-8859-1"?>
 <!DOCTYPE Music_Library [
     <!ELEMENT Music_Library (Title,Artist,Country,Price,Year)>
     <!ELEMENT Title (#PCDATA)>
     <!ELEMENT Artist (#PCDATA)>
     <!ELEMENT Country (#PCDATA)>
     <!ELEMENT Price (#PCDATA)>
     <!ELEMENT Year (#PCDATA)>
     <!ENTITY MS "Thatz life">
 ]>
 <Music_Library>
    <Title>YLO</Title>
    <Artist>&MS; - Jenny Dan</Artist>
    <Country>Germanyo</Country>
    <Price>$12</Price>
    <Year>2002</Year>
</Music_Library>
```

where,
The first block indicates xml declaration and document type declaration. `Music_Library` is the root element.

Using slides 14 to 17, explain the XML document structure.

XML documents are commonly stored in text files with extension `.xml`. The two sections of an XML document are as follows:

- **Document Prolog**: XML parser gets information about the content in the document with the help of document prolog. Document prolog contains metadata and consists of two parts - XML Declaration and DTD. XML Declaration specifies the version of XML being used. DTD defines entities' or attributes' values and checks grammar and vocabulary of markup.

- **Root Element**: The second is an element called the root element. The root element is also called a document element. It must contain all the other elements and content in the document. An XML element has a start tag and end tag.

Tell them that an XML document contains XML elements. An XML element is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain:

- other elements
- text
- attributes
- or a mix of all

Figure on slide 16 shows the XML document structure. Explain the code which demonstrates the XML document structure using slide 17.

Tell them that the first block indicates xml declaration. The second block shows the DTD of XML document and finally, the third block shows XML documents with `Music_Library` as the root element.

In-Class Question:

After you finish the XML documents structure, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

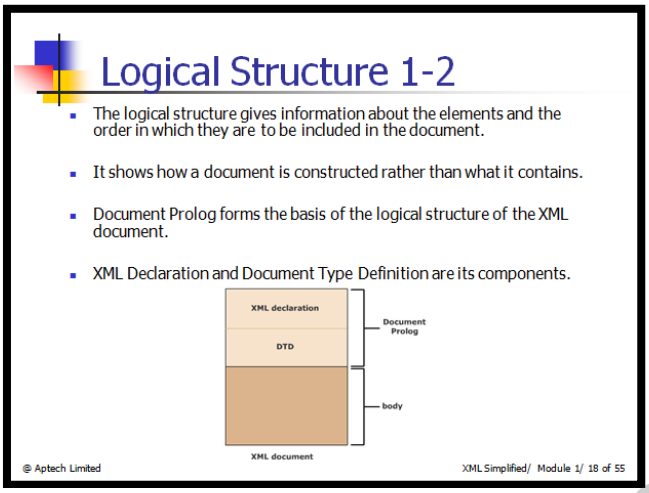
Why you should use DTD for an XML document?

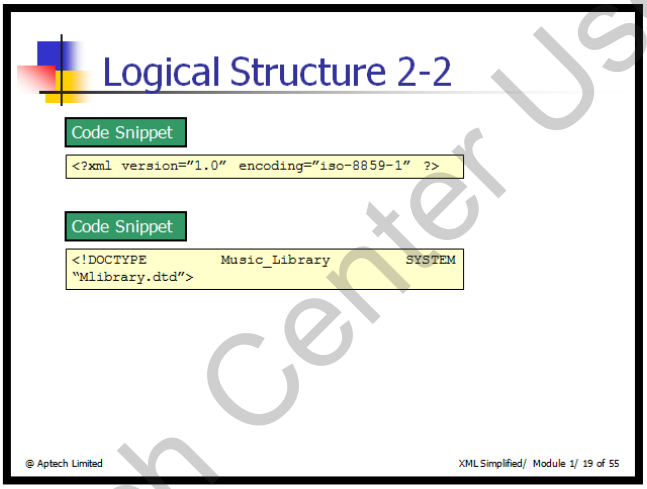
Answer:

1. To follow or agree on the standards for interchanging data.
2. To verify that the XML data received from the other application is valid.

Slides 18 and 19

Let us understand the logical structure of XML document.





Using slides 18 and 19, explain the logical structure of XML document.

The logical structure of an XML document gives information about the elements and the order in which they are to be included in the document. It shows how a document is constructed rather than what it contains.

Document Prolog forms the basis of the logical structure of the XML document. It contains the following components:

- **XML Declaration** - Gives the version of the XML specification and also identifies the character encoding scheme.
  Example: `<?xml version="1.0" encoding="iso-8859-1" ?>`

- **DTD** - Defines a set of rules to which the XML document conforms. A DTD can be either external or internal. In this case, the DTD is an external file.
  Example: `<!DOCTYPE Music_Library SYSTEM "Mlibrary.dtd">`

Tips:

An XML document validated against a DTD is "Well Formed" and "Valid".

In-Class Question:

After you finish the logical structure of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

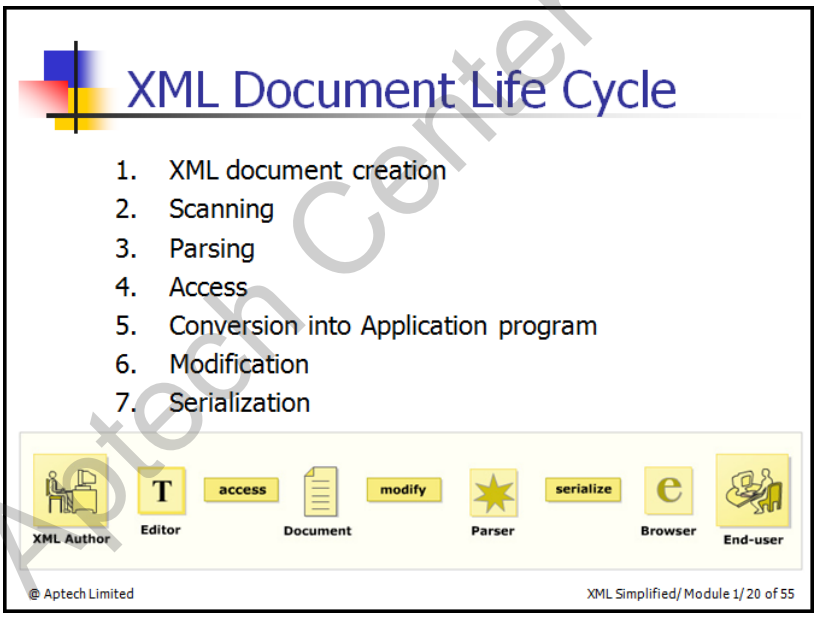
If the XML document contains international characters, then which part of the declaration must be specified to avoid errors?

Answer:

You can specify the encoding used in the XML document.

Slide 20

Let us understand the XML document life cycle.



Using slide 20, explain the XML document life cycle.

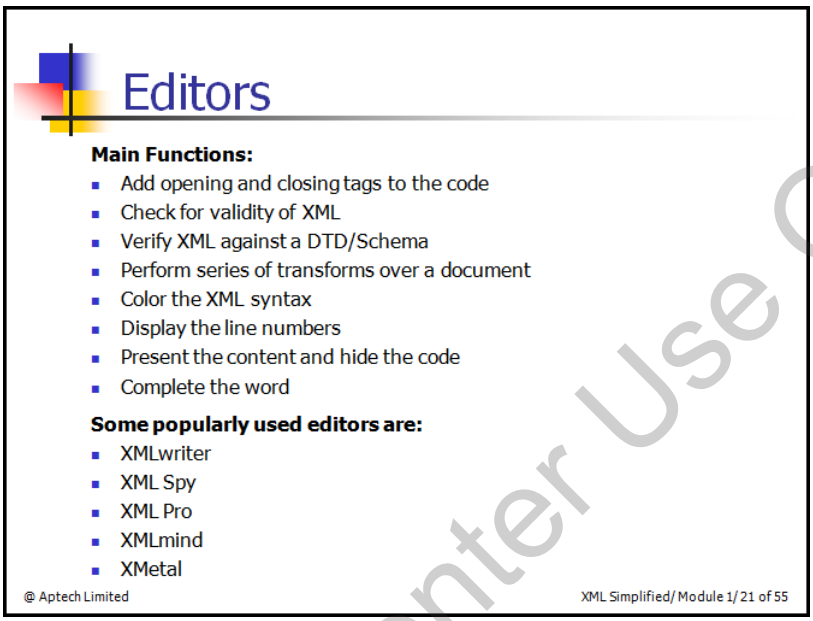
An XML editor refers to the DTD and creates an XML document. After the document is created, the document is scanned for elements and attributes in it. This stage is known as **scanning**. The XML parser builds a complete data structure after parsing. The data is then extracted from elements and attributes of the document. This stage is known as **access**. It is then converted into the application program. The document structure can also be modified during the process by inserting or deleting elements or by changing textual content of element or attribute. This stage is known as **modification**. The data is then serialized to a

textual form and is passed to a browser or any other application that can display it. This stage is known as **serialization**.

Figure shown on the slide displays the XML document life cycle.

Slide 21

Let us understand the editors.



Using slide 21, explain the editors.

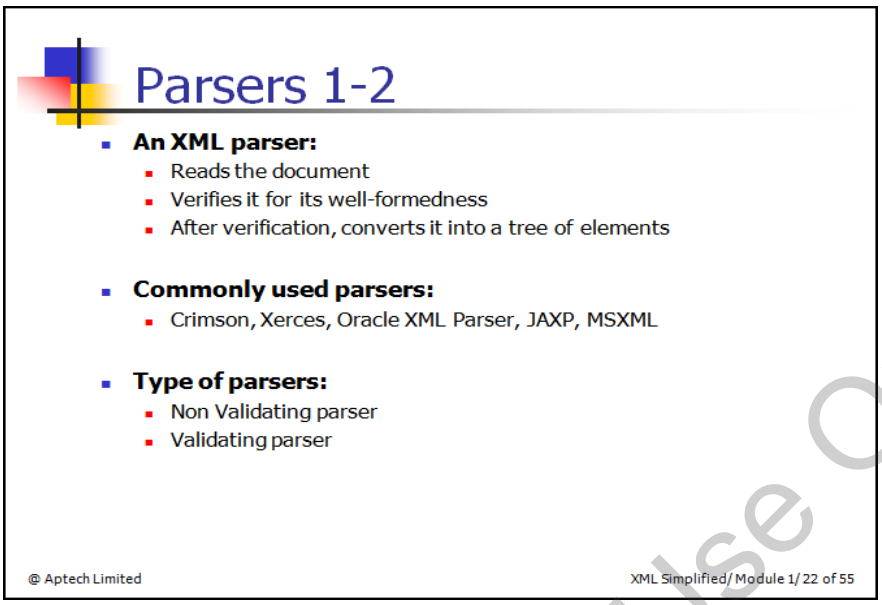
An XML Editor is used to create and edit XML documents. Any application can be used as an editor in XML. Since all XML documents are text-based markup languages, a standard Windows Notepad or WordPad can also be used. However, for various reasons, Notepad should not be used for writing professional XML. Notepad does not know that the text written in it is XML code, and thus, can create problem.

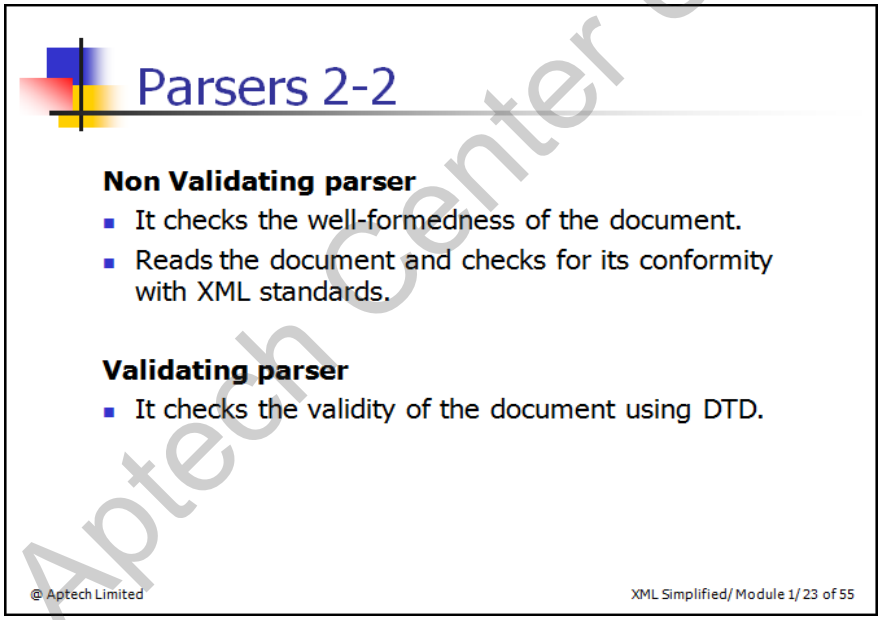
For an XML document to be error-free and possess XML-specific features such as the ability to edit elements and attributes, a professional XML editor should be used.

Explain the main functions that XML editors must provide and also list the names of some of the popularly used XML editors.

Slides 22 and 23

Let us understand the parsers.





Using slides 22 and 23, explain the parsers.

An XML parser or XML processor reads the document and verifies it for its well-formedness.

An XML parser keeps the entire representation of XML data in memory with the help of Document Object Model (DOM). The in-built parser used in IE 5.0 is also known as Microsoft XML Parser (MSXML). It is a component, which is available once IE 5.0 is installed. Microsoft's XML parser goes through the entire data structure and accesses the values of the attributes in the elements. The parser also creates or deletes the elements and converts the tree structure into XML.

MSXML supports some of the COM objects for backward interoperability. Some of them are XSLProcessor objects, XSLTemplate objects, XMLDOMSelection objects, XMLSchemaCache objects, and XMLDOMDocument2 objects.

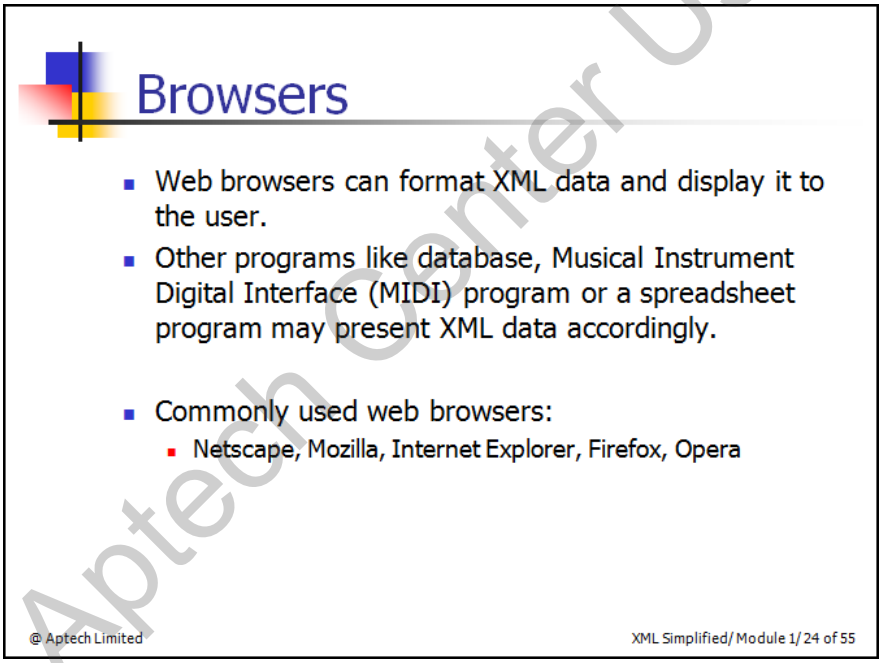
XML parsing in Mozilla performs functions such as going through the elements, accessing their values, and so on. Using JavaScript, an instance of XML parser can be created in Mozilla browsers. The XML parsing in Firefox automatically parses the data into DOM. Opera uses only those XML parsers that do not validate DTDs by default.

Speed and performance are the criteria against which XML parsers are selected. Then, mention the commonly used XML parsers.

Also, explain the types of parsers in details using slide 23.

Slide 24

Let us understand the browsers.



Using slide 24, explain the browsers.

After the XML document is read, the parser passes the data structure to the client application. The application can be a Web browser. The browser then formats the data and displays it to the user. Other programs such as database, Musical Instrument Digital Interface (MIDI) program, or a spreadsheet program may also receive the data and present it accordingly.

The final output of XML data is viewed in a browser. XML is not supported by all browsers, for example, Netscape Navigator 4.0 does not support XML but later versions of the browser like Netscape 6.0 do support it. Only browsers like IE 5.0 or greater give full support for XML

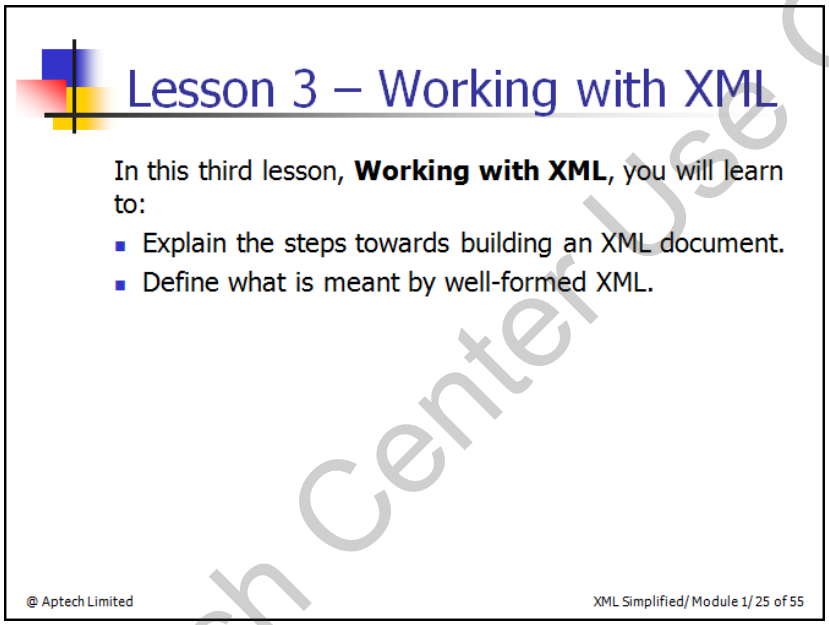
specifications. In IE, XML can be directly viewed using style sheets. It gives support to namespaces and handles a mechanism known as **data islands** where XML is embedded into HTML.

Mozilla 5.0 uses an interface to the XML DOM via JavaScript and plug-ins. It also supports elements from the HTML namespace.

Mention the commonly used browsers.

Slide 25

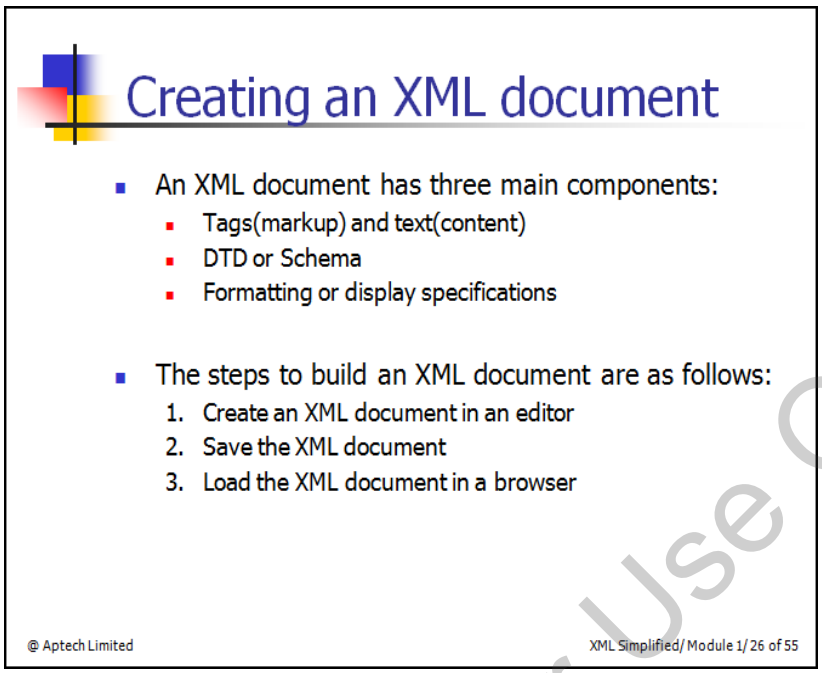
Let us understand the objectives of lesson 3.



Using slide 25, describe the topics covered in lesson 3.

Slide 26

Let us understand how to create a XML document.



Using slide 26, explain how to create a XML document.

The steps to build an XML document are as follows:

- **Create an XML document in an editor**
  To create a simple XML document, you type XML code in any text or XML editor.

  Explain the following code snippet in demonstration:

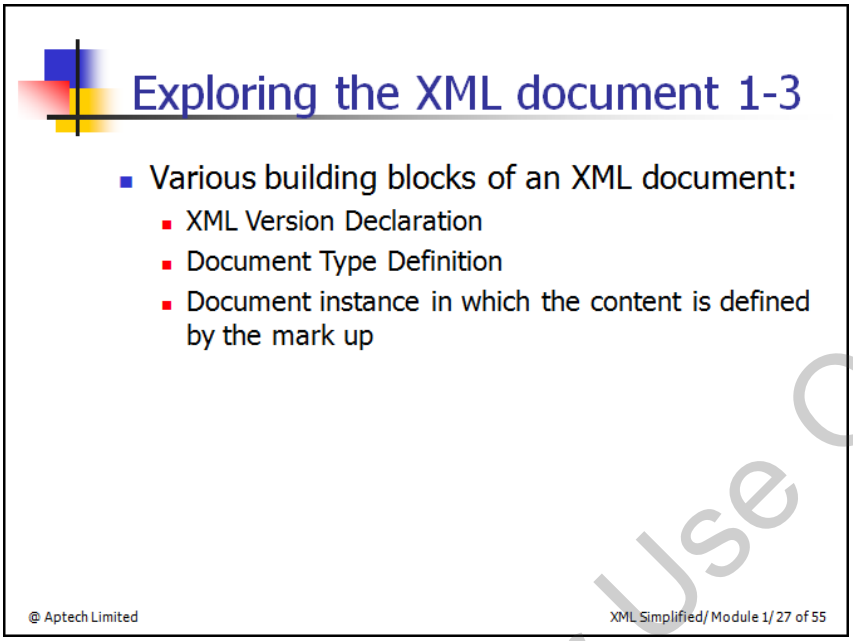
```
<?xml version="1.0" encoding="iso-8859-1"?>
<Soft_drink>
<Name>Refresh</Name>
<Price>$1</Price>
</Soft_drink>
```

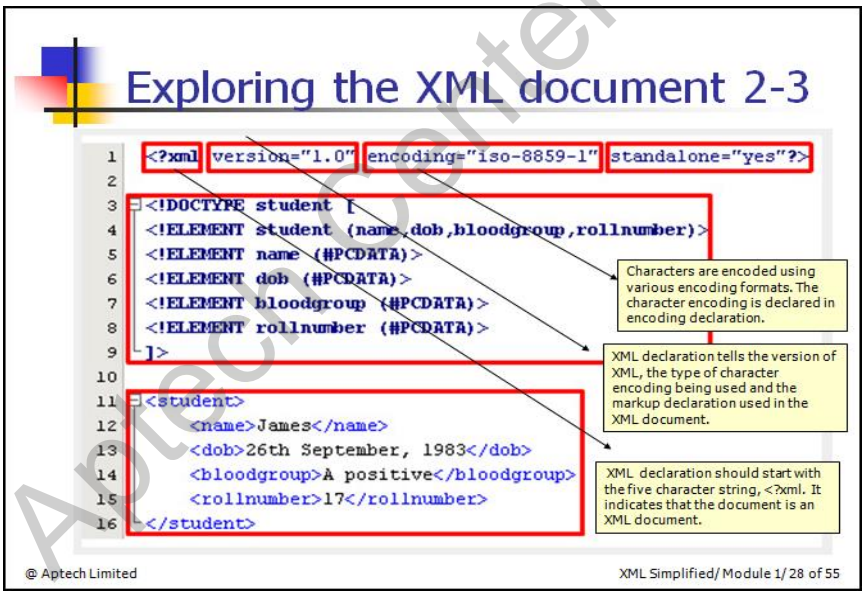
- **Saving the XML file**
  After the typing is done, you save the contents of the file. To save the file, you click the **File** menu, click the **Save** option and provide the file name such as, `SoftDrink.xml`. The extension `.xml` is compulsory if you are typing the code in Notepad or any kind of word processor such as Microsoft Word.

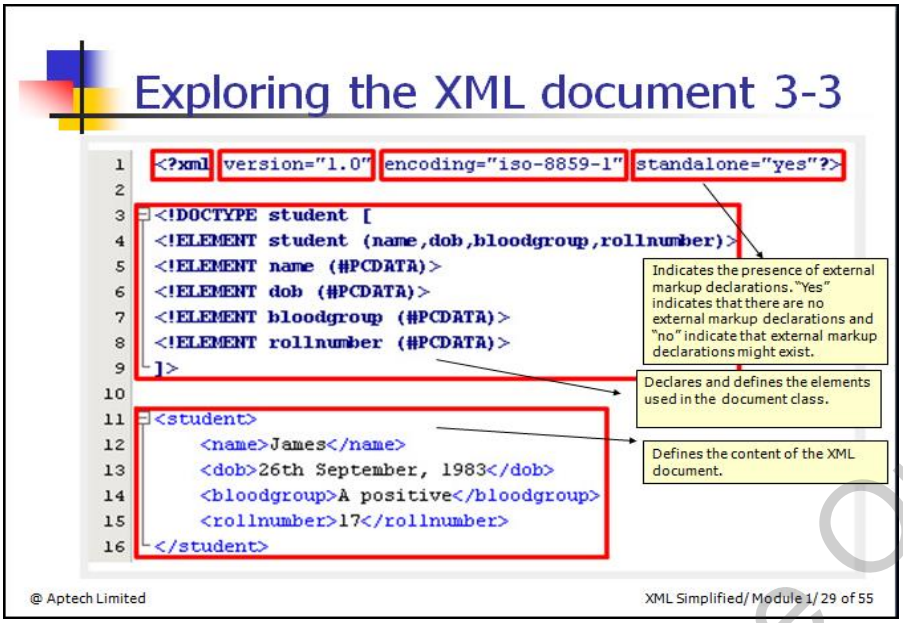
- **Load XML document in a browser**
  After the document is saved, you can open the file directly in a browser that supports XML. A common browser used is Internet Explorer 5.0 or later.

Slides 27 to 29

Let us understand the building blocks of XML document.







Using slides 27 to 29, explain the the building blocks of XML document.

Figure on slide 28 depicts a XML document. Explain the code in the figure with each block in details.

- `<?xml`

  The XML declaration should start with the five character string, `<?xml`. It indicates that the document is an XML document.

- `version="1.0"`

  The XML declaration tells the version of XML, the type of character encoding being used and the markup declaration used in the XML document.

- `encoding="iso-8859-1"`

  Characters are encoded using various encoding formats. The character encoding is declared in encoding declaration.

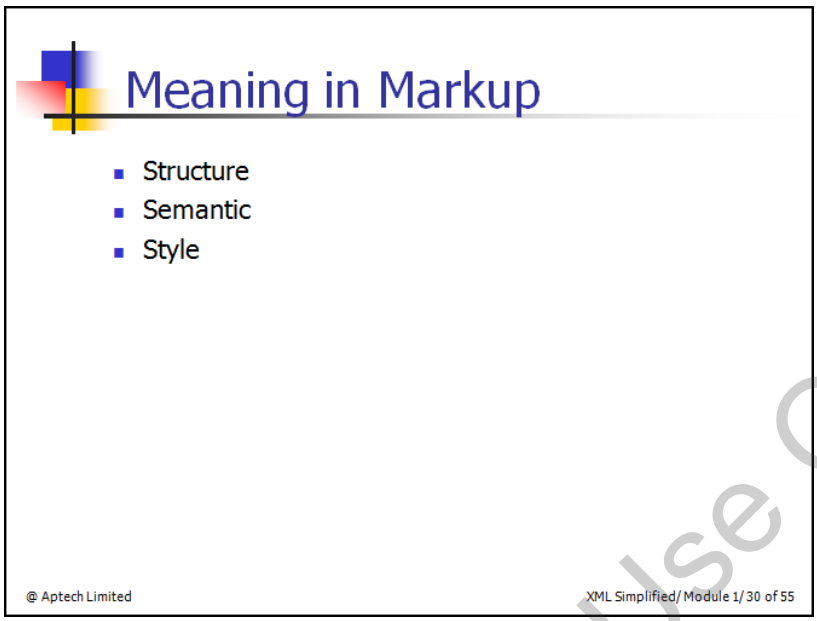
- `standalone="yes"`

  The standalone declaration indicates the presence of external markup declarations. "`Yes`" indicates that there are no external markup declarations and "`no`" indicates that external markup declarations might exist.

DTD for the XML document can be declared internally or externally.

Slide 30

Let us understand the meaning of components in markup.



Using slide 30, explain the meaning of components in markup.

Markup can be divided into:

- **Structure**
  It describes the form of the document by specifying the relationship between different elements in the document. It emphasizes to specify a single non-empty, root element that contains other elements, and the content.

- **Semantic**
  Semantics describes how each element is specified to the outside world of the document. For example, an HTML enabled Web browser assigns "paragraph" to the tags `<P>` and `</P>`, but not to the tags `<Message>` and `</Message>`.

- **Style**
  It specifies how the content of the tag or element is displayed. It indicates whether the tag is bold, normal, and pink in color or with the font size 10.

In-Class Question:

After you finish the components of XML, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

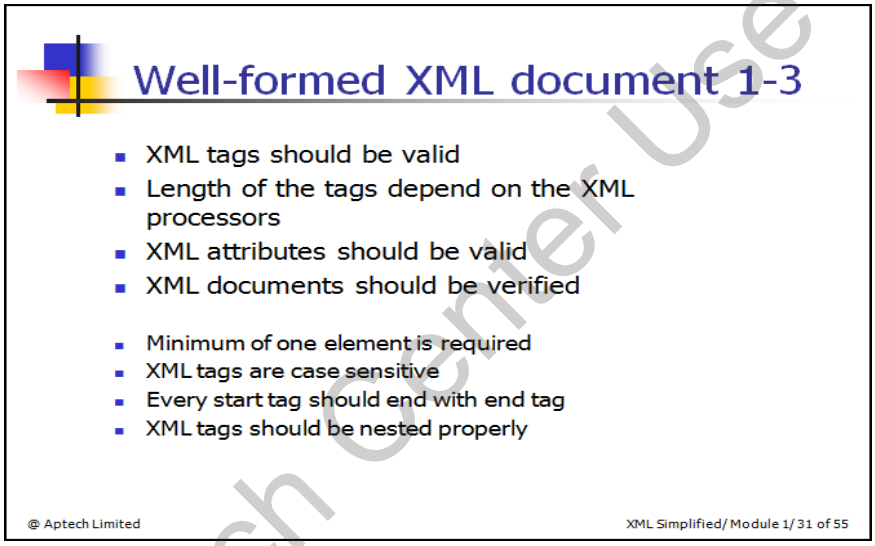
An XML document with correct syntax is referred to as _____.

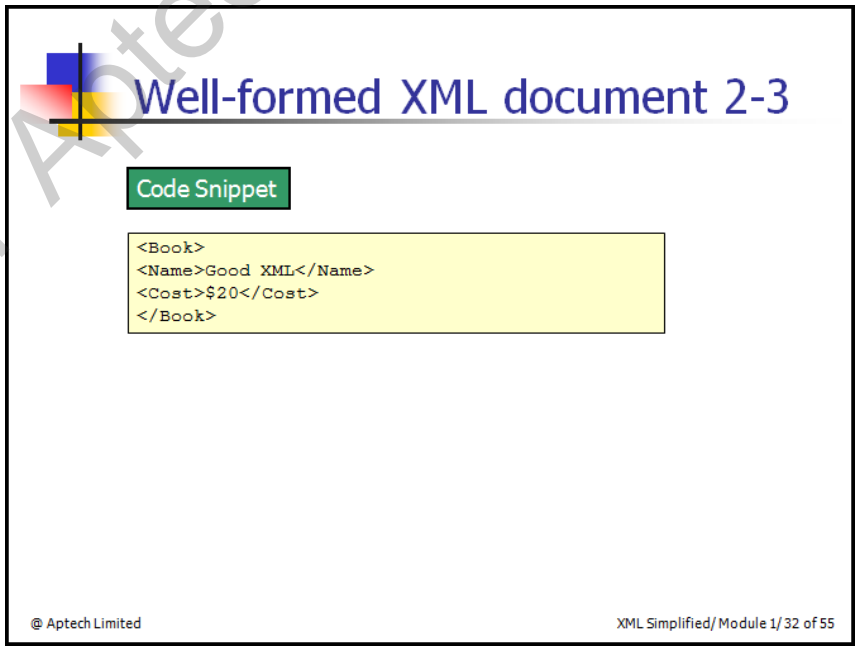
Answer:

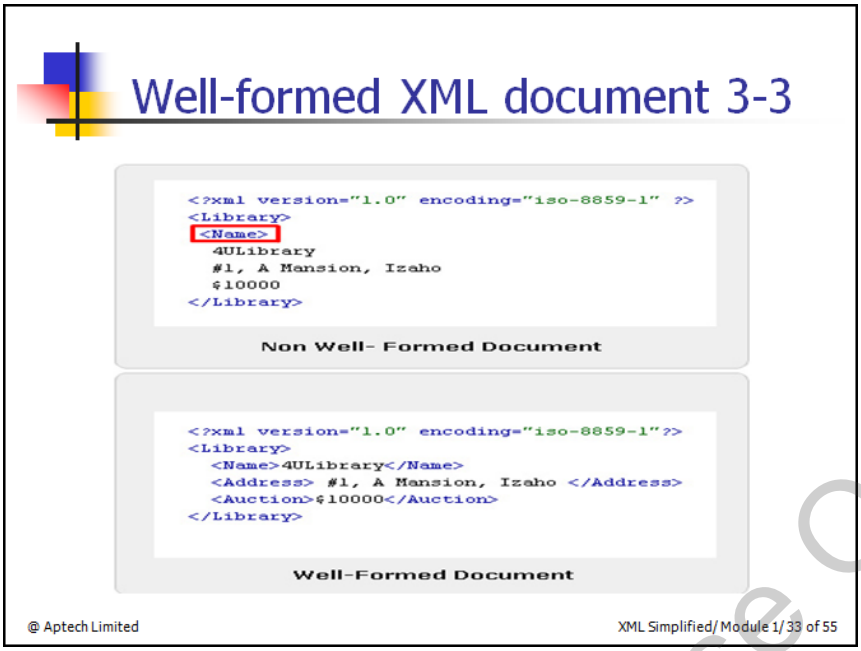
Well-formed

Slides 31 to 33

Let us understand the well-formed XML document.







Using slides 31 to 33, explain the well-formed XML document.

Well-formedness refers to the standards that are to be followed by the XML documents. Well-formedness makes XML processors and browsers read XML documents.

Minimum of One Element is Required

Every well-formed XML document should consist of a minimum of one element.

All XML Elements Must Have a Closing Tag

In HTML, some elements do not have a closing tag. For example, `<br>`. In XML, all elements must have a closing tag.

For example, consider the following XML syntax:

```
<p> This is a paragraph. </p>
<br />
```

XML Tags are Case Sensitive

XML tags are case sensitive. The tag `<author>` is different from the tag `<Author>`. Opening and closing tags must be written with the same case.

For example,

```
<author>This is incorrect</Author>
<author>This is correct</author>
```

XML Elements Must be Properly Nested

In HTML, you might see improperly nested elements. For example, `<b><i>This text is bold and italic</b></i>`. In XML, all elements must be properly nested within each other. For example, `<b><i>This text is bold and italic</i></b>`.

XML Documents Must Have a Root Element

XML documents must contain one element that is the **parent** of all other elements. This element is called the **root** element.

XML Attribute Values Must be Quoted

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

For example,
```
<Person age="27”>
  <first_name>John</first_name>
  <last_name>Jani</last_name>
</Person>
```

Comments in XML

The syntax for writing comments in XML is similar to that of HTML. For example, `<!--This is a comment -->`

White-space is Preserved in XML

In HTML, multiple white-space characters are truncated to one single white-space. In XML, the white-spaces in a document is not truncated.

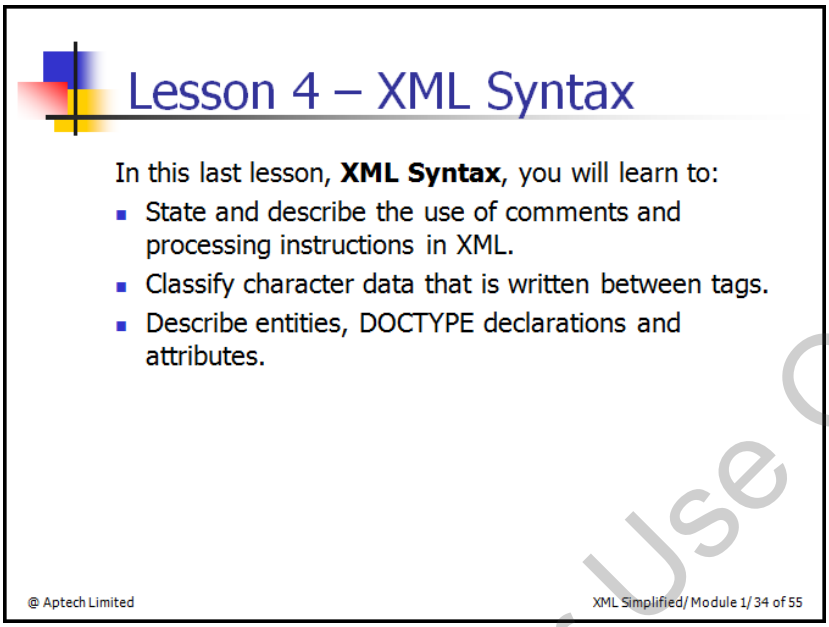
XML Tags should be Valid

- Tags should begin with letter, an underscore (_), or a colon (**:**).
- Tags should contain combination of letters, numbers, periods (**.**), colons, underscores, or hyphens (−).
- Tags should not contain white space.
- Tags should not start with reserved words like ’xml’.

If an XML documents that conform to the syntax rules are said to be ’Well formed‘ XML documents.

Slide 34

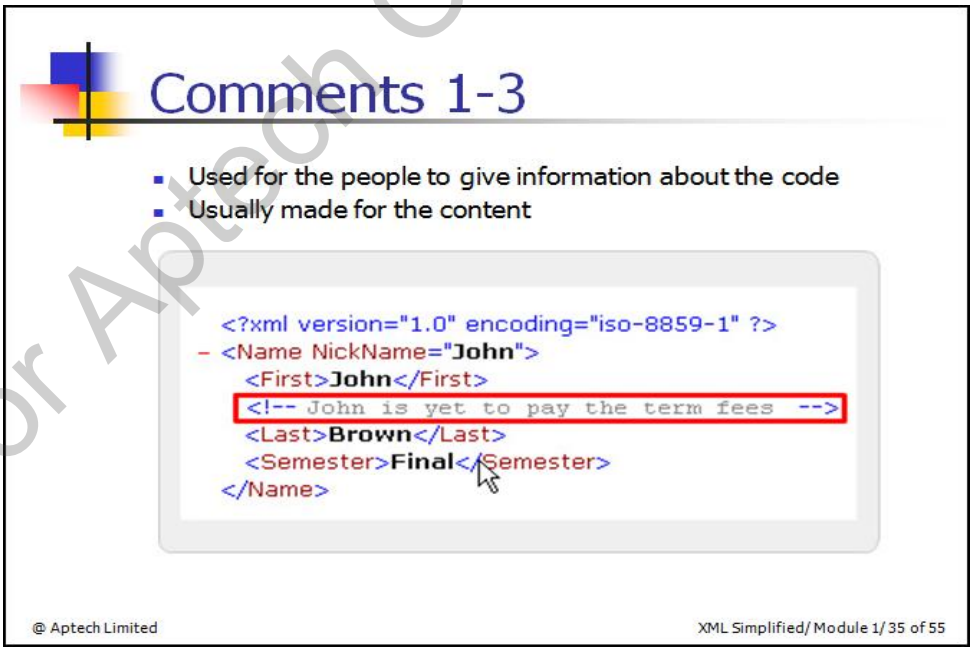
Let us understand the objectives of lesson 4.

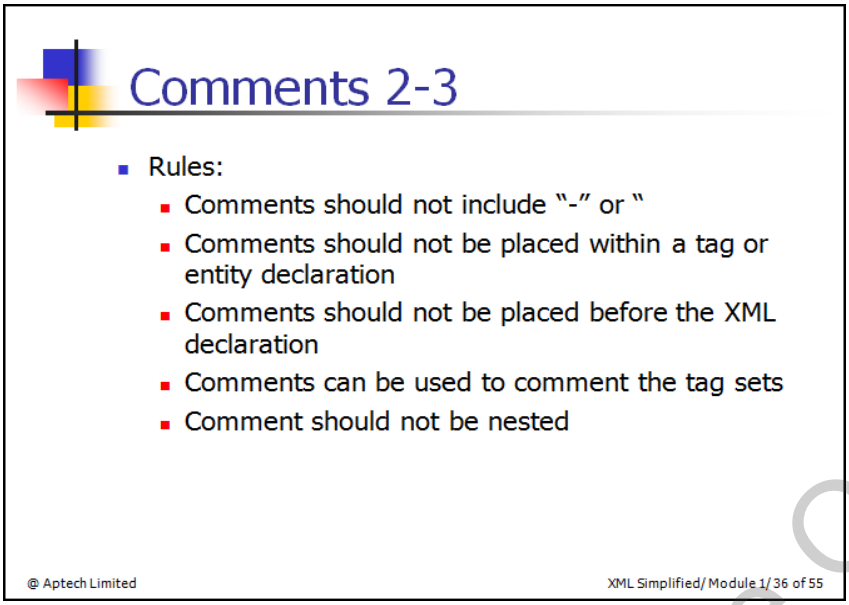


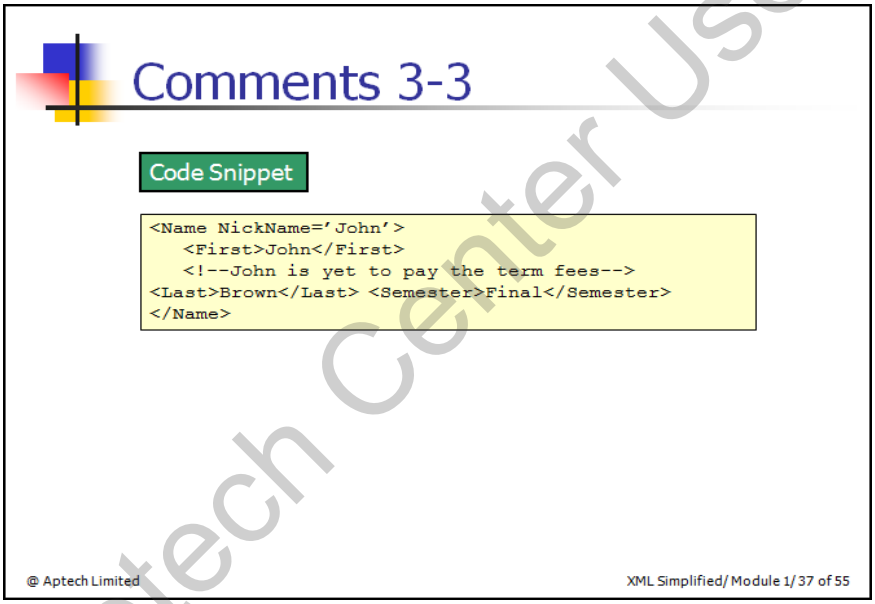
Using slide 34, explain the topics to be covered in lesson 4.

Slides 35 to 37

Let us understand the comments.







Using slides 35 to 37, explain the comments.

XML comments are used for the people to give information about the code in the absence of the developer. It makes a document more readable. Comments are not restricted to DTD, but may be placed anywhere in the document.

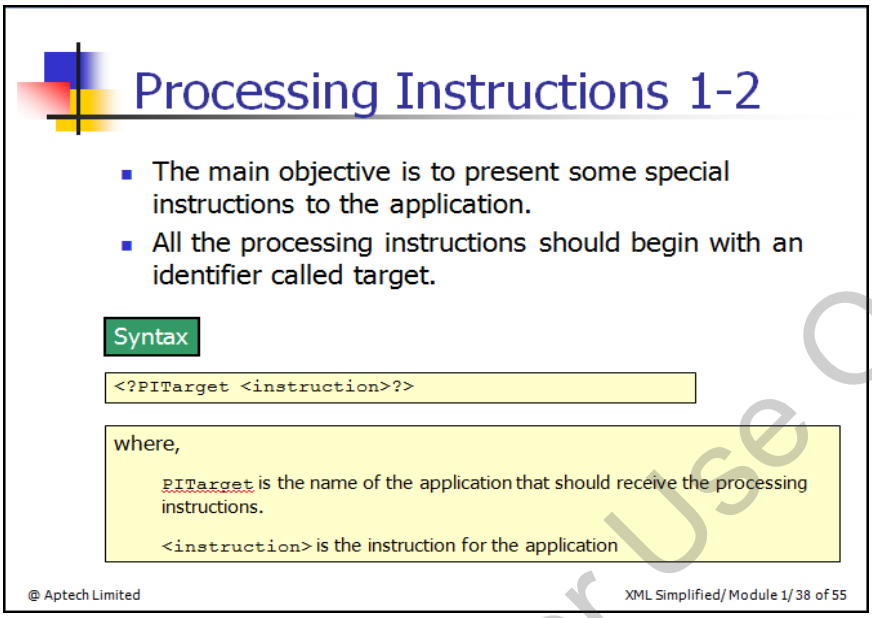
Comments in XML are similar to those in HTML. Comments should be used only when needed, as they are not processed. Comments are used only for human consumption rather than machine consumption. Since the comments are not parsed, their presence or absence does not make any difference to the processors.

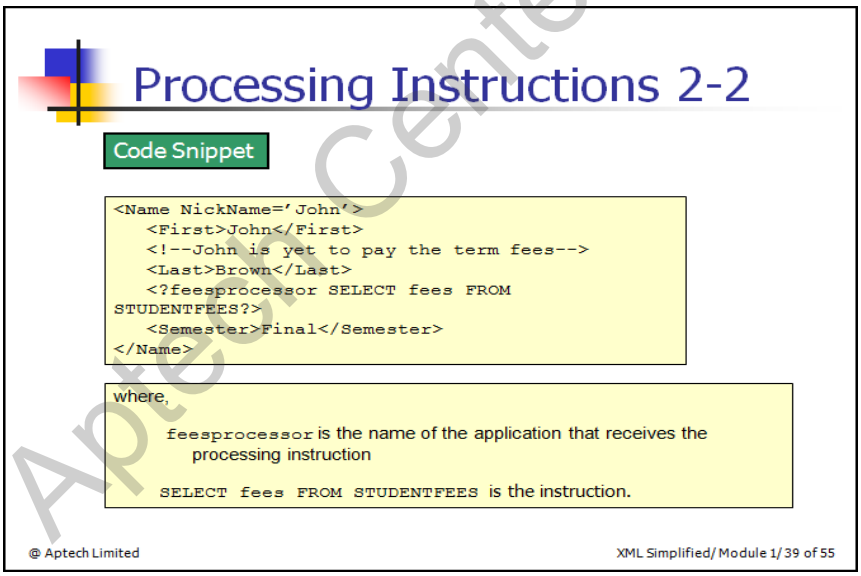
Comments start with the string < ! −− and end with the string −−>. The parser believes that the comment has come to an end when it finds a > as shown in figure on slide 35.

Explain the rules for writing comment and demonstrate the code using slides 36 and 37.

Slides 38 and 39

Let us understand the processing instructions.





Using slides 38 and 39, explain the processing instructions.

Processing instructions are information which is application specific. These instructions do not follow XML rules or internal syntax. With the help of a parser, these instructions are passed to the application. The application can either use the processing instructions or pass them on to another application.

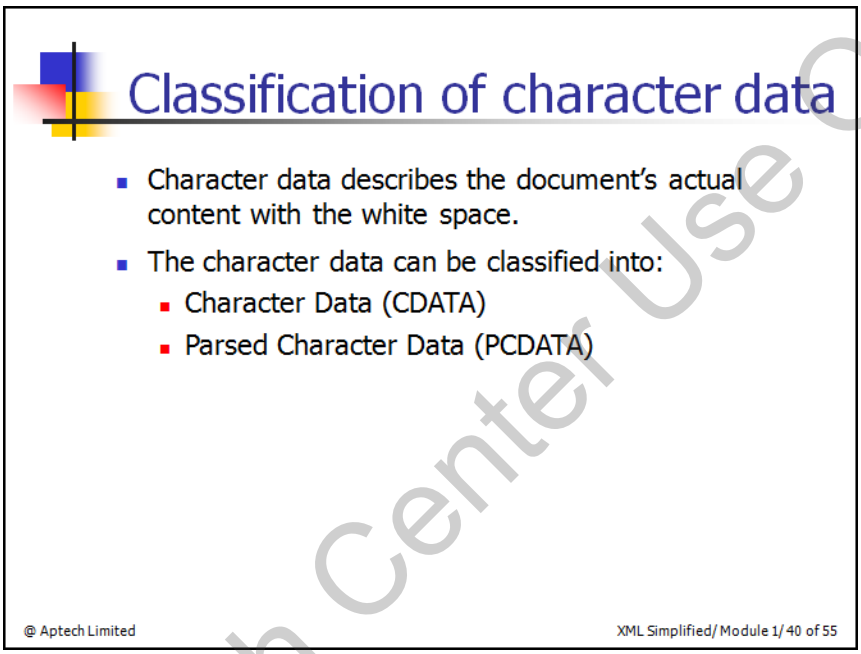
The main objective of a processing instruction is to present some special instructions to the application. All processing instructions must begin with <? and end with ?>.

Though an XML declaration also begins with <? and end with ?>, it is not considered as a processing instruction. It is because an XML declaration provides information only for the parsers and not for the application. In some cases, the application might need the information in the processing instruction only if it displays the output to the user.

Then, explain the syntax using slide 38 to write process instructions. Also, explain the code on slide 39 which demonstrates an example of processing instruction.

Slide 40

Let us understand the classification of character data.



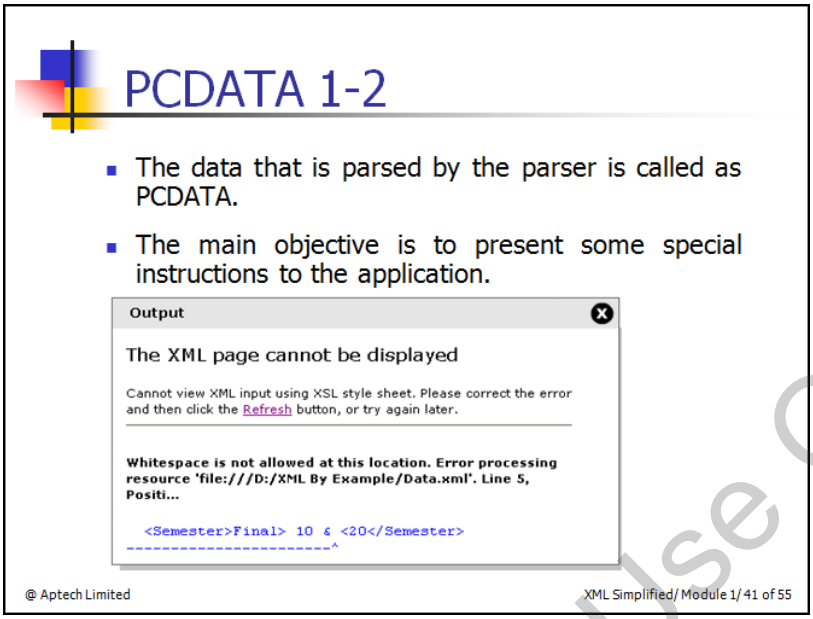
Using slide 40, explain the classification of character data.

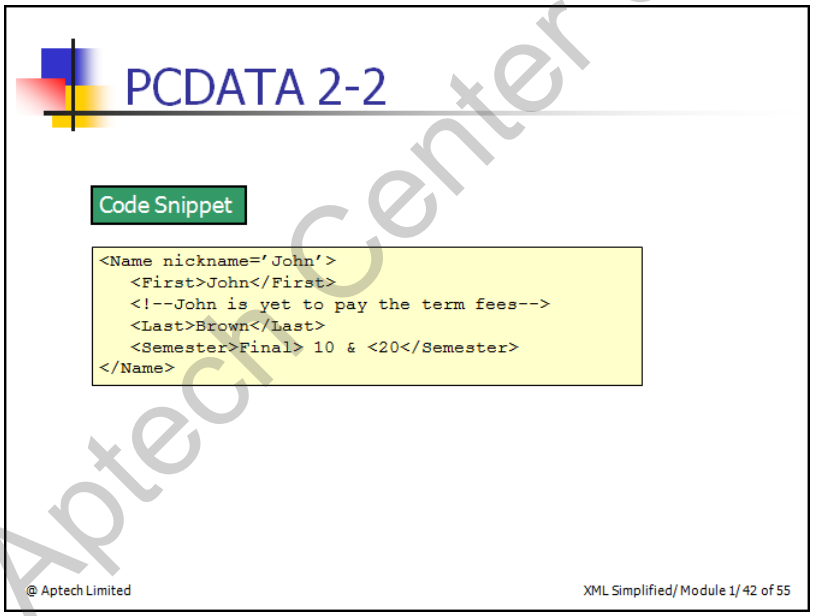
An XML document is divided into markup and character data.
Character data describes the document's actual content with the white space. The text in character data is not processed by the parser and thus, not treated as a regular text. The character data can be classified into:

- CDATA
- PCDATA

Slides 41 and 42

Let us understand the PCDATA.





Using slides 41 and 42, explain the PCDATA.

The data that is parsed by the parser is called as parsed character data (PCDATA). The PCDATA specifies that the element has parsed character data. It is used in the element declaration.

Escape character like '<' when used in the XML document, will make the parser interpret it as a new element. As a result, it will generate an error as shown in figure on slide 41.

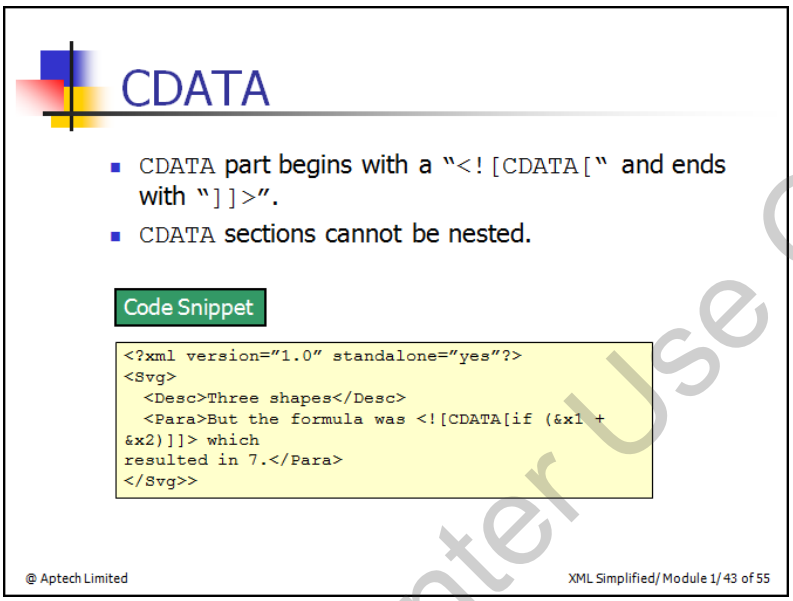
Explain the code on slide 42 which demonstrates an example of PCDATA.

Tips:

To avoid error for special characters, replace the '<'character with an entity reference.

Slide 43

Let us understand the CDATA.



Using slide 43, explain the CDATA.

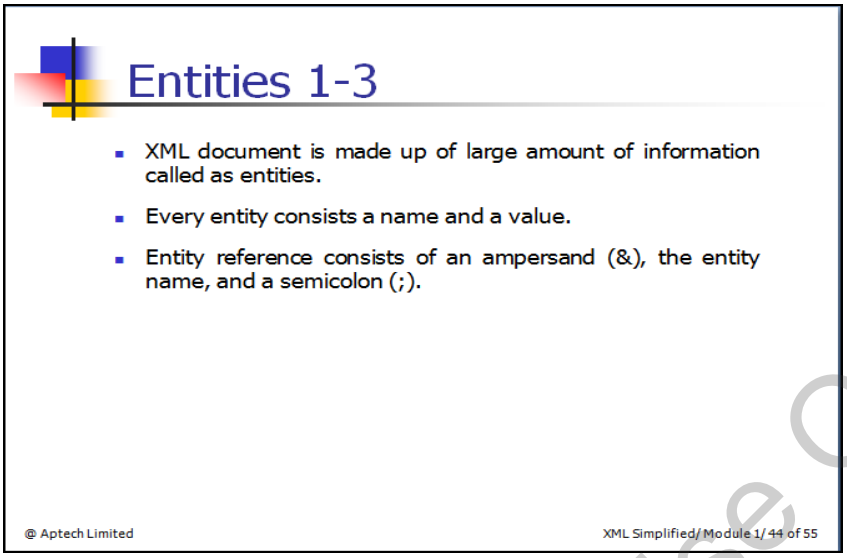
CDATA stands for character data that has reserved and white space characters in it. Though the text inside the CDATA is not parsed by the parser, it is commonly used for scripting code. XML parser ignores all tags and entity references inside the CDATA blocks. The CDATA block indicates to the parser that it is just a text but not a markup. A CDATA block always begins with the delimiter `<![CDATA[` and ends with the delimiter `]]>`.

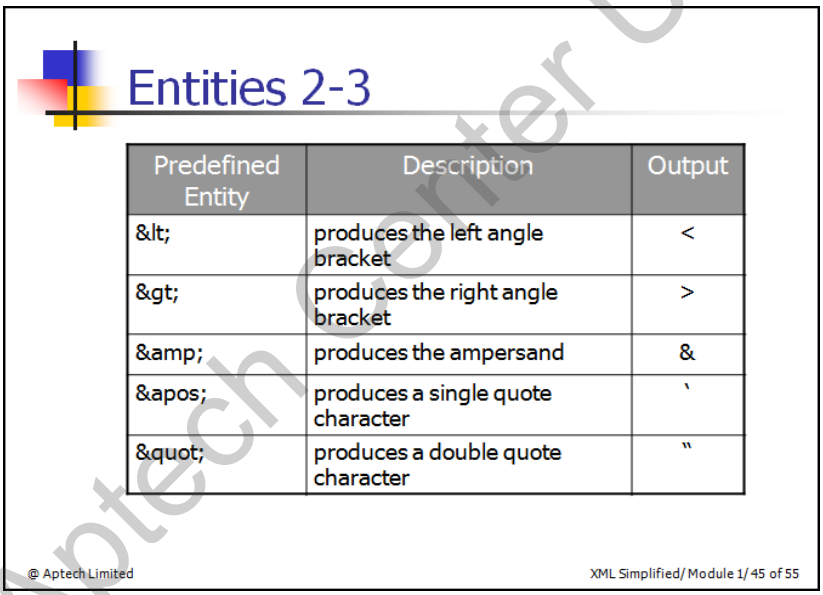
Since the ending delimiter marks the end of the CDATA block, the character string `]]/` is not allowed in the middle of the CDATA block. This will signal the end of the CDATA section.

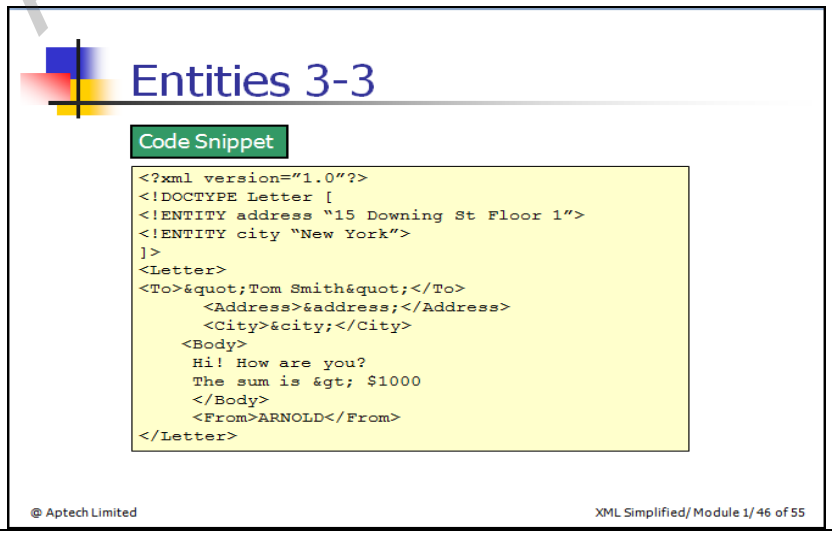
Explain the syntax of CDATA and then explain the code on slide 43 which demonstrates CDATA.

Slides 44 to 46

Let us understand the entities.







Using slides 44 to 46, explain the entities.

The XML document is made up of large amount of information called as entities. Entities are used to avoid typing long pieces of text repeatedly, within the document.

Explain the categories of entities are as follows:

- **Character entities**: They form the mechanism, which is used in place of a character's literal form. They provide the meaning of '>' when the symbol '`>`' is typed. Character entities can also be used with decimal or hexadecimal values with a condition that the numbers support Unicode coding. An XML processing program replaces character entities with their equivalent characters.

- **Content entities**: These entities are used to replace certain values. They are similar to text substitution macros in programming languages such as C. Content entity has the following syntax: `<!Entity name value>`

- **Unparsed entities**: These entities when used, turn off the parsing process. They can be used to include multimedia content in the XML document.

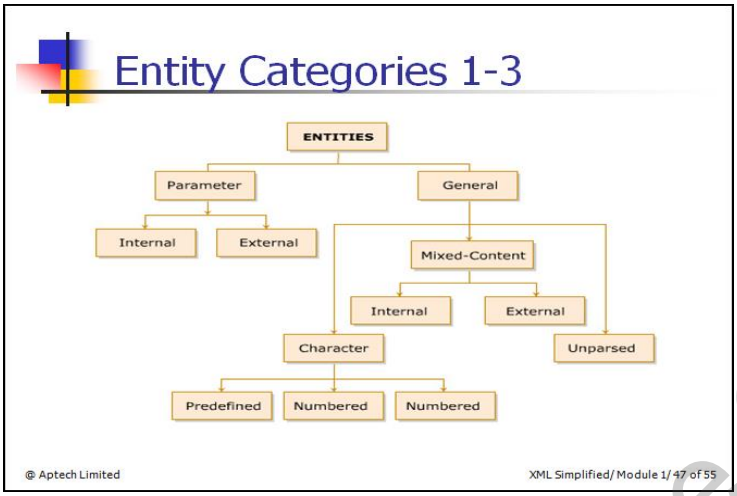
Every entity consists a name and a value. The value ranges from a single character to a XML markup file. As the XML document is parsed, it checks for entity references. For every entity reference, the parser checks the memory to replace the entity reference with a text or markup.
All the entities must be declared before they are used in the document. An entity can be declared either in a document prolog or in a DTD.

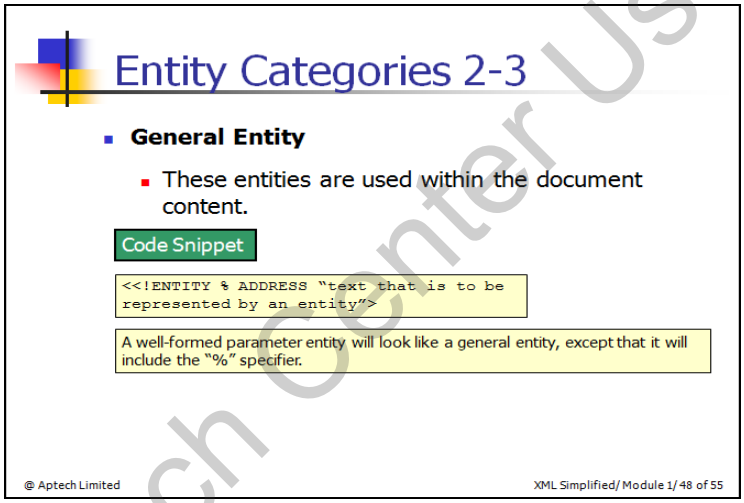
Some of the entities are defined in the system and are known as pre-defined entities. These entities are described in table shown on slide 45.

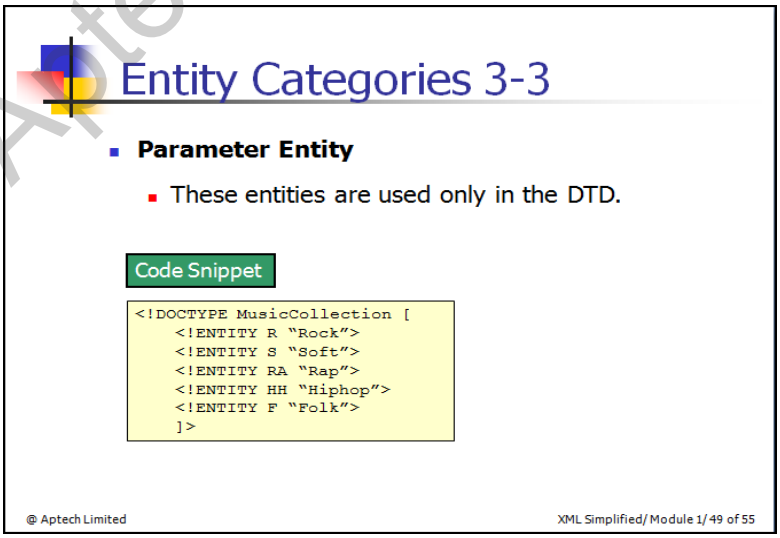
Then, explain the code on slide 46 which demonstrates an example of entities.

Slides 47 to 49

Let us understand the entity categories.







Using slides 47 to 49, explain the entity categories.

Entities are used as shortcuts to refer to the data pages. There are two types of entities that are as follows:

- General Entity
- Parameter Entity

General entities are used within the document content. General entities can either be declared internally or externally. The references for general entities start out with an ampersand (&) and end with a semicolon (;). The entity's name is present within these two characters.

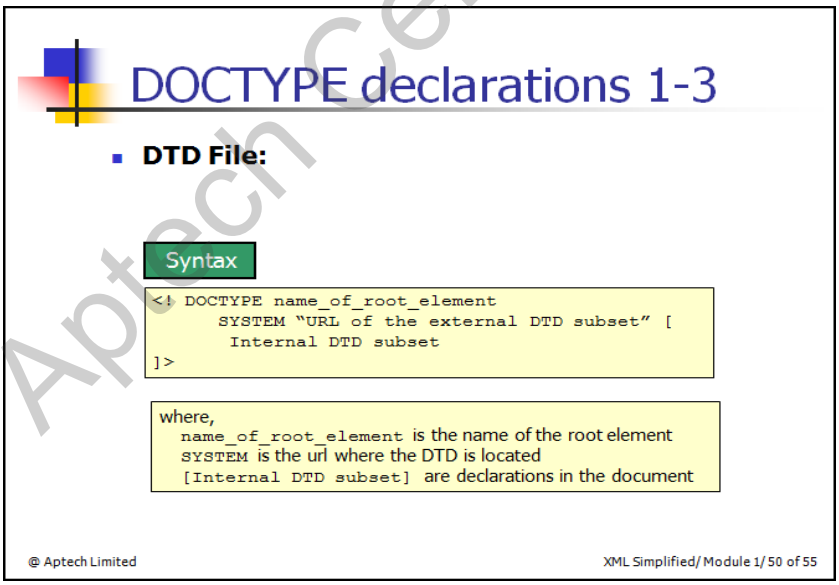
Explain the code on slide 48 which demonstrates an example of general entity.

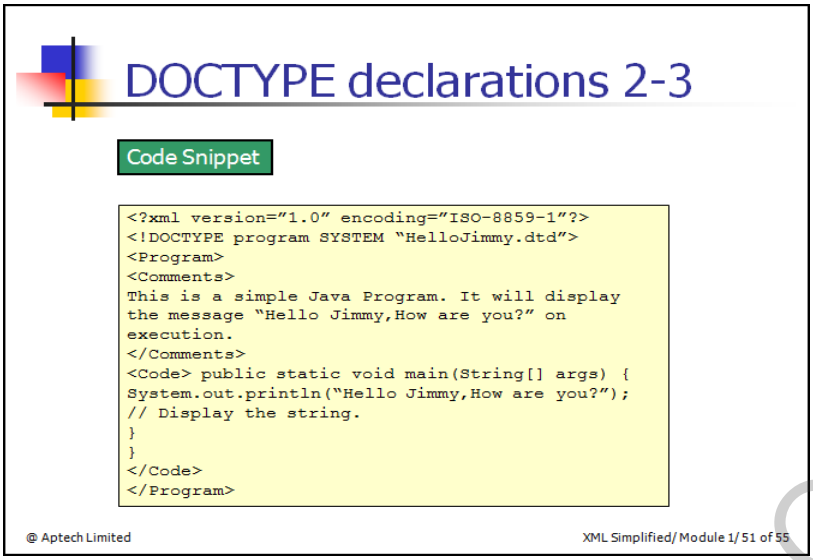
Mention parameter entities are used only in the DTD. These types of entities are declared in DTD. Both internal and external parameter entities should not be used in the content of the XML document as the processor does not recognise them. A well-formed parameter entity is similar to general entity, except that, it will include the % specifier. The reference is also similar to the general entity reference. References to these entities are made by using percent-sign (%) and semicolon (;) as delimiters.

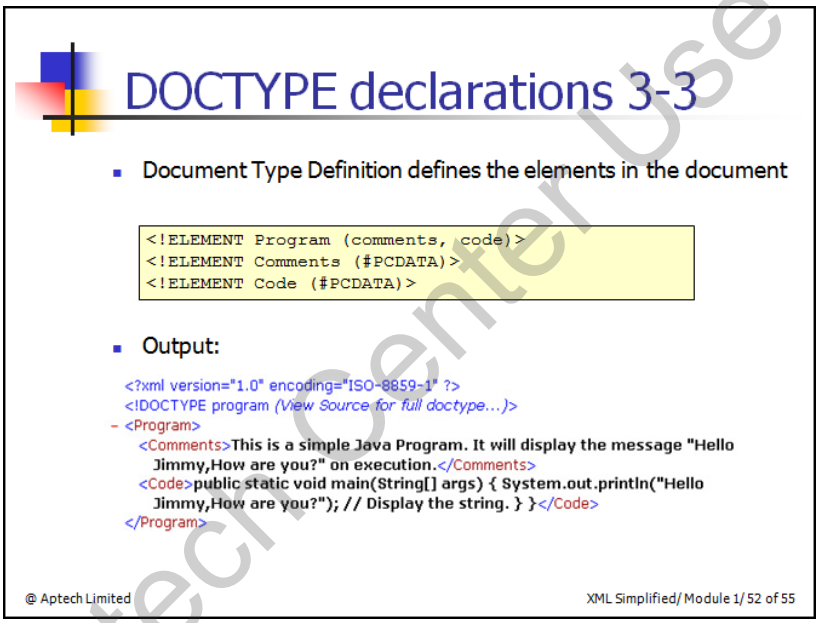
Explain the code on slide 49 which demonstrates an example of parameter entity.

Slides 50 to 52

Let us understand the DOCTYPE declarations.







Using slides 50 to 52, explain the DOCTYPE declarations.

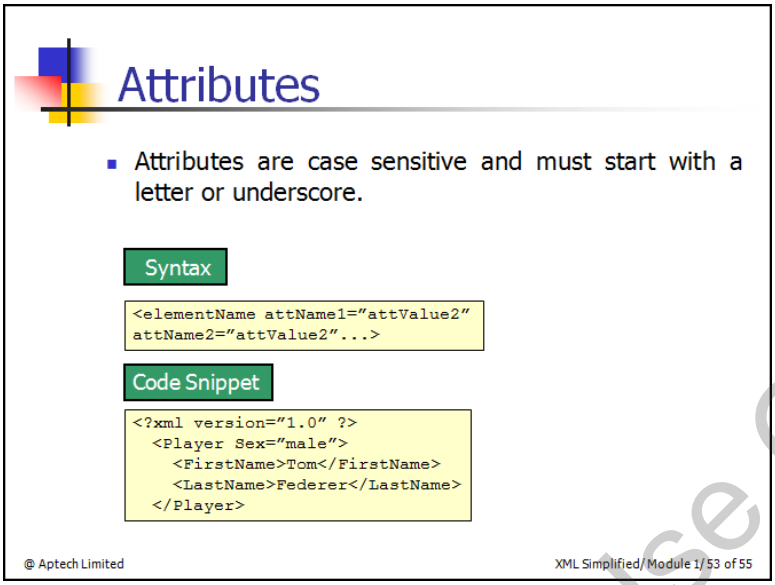
The DTD defines the elements to be used in the document. A DTD is declared to indicate what DTD the document adheres to. It can be declared either in the XML document or can be referenced to the external document.

Explain the syntax of document type declaration using slide 50.

Explain the code on slide 51 demonstrates an example of document type declaration.

Slide 53

Let us understand the attributes.



Using slide 53, explain the attributes in XML document.

Attributes are part of the elements. They provide information about the element and are embedded in the element start-tag. An attribute consists of an attribute name and an attribute value. The name always precedes its value, which are separated by an equal sign. The attribute value is enclosed in the quotes to delimit multiple attributes in the same element.

An attribute can be of `CDATA`, `ENTITY`, `ENUMERATION`, `ID`, `IDREF`, `NMTOKEN`, or `NOTATION`.

Explain the syntax and code snippet which demonstrates an example of attributes using slide 53.

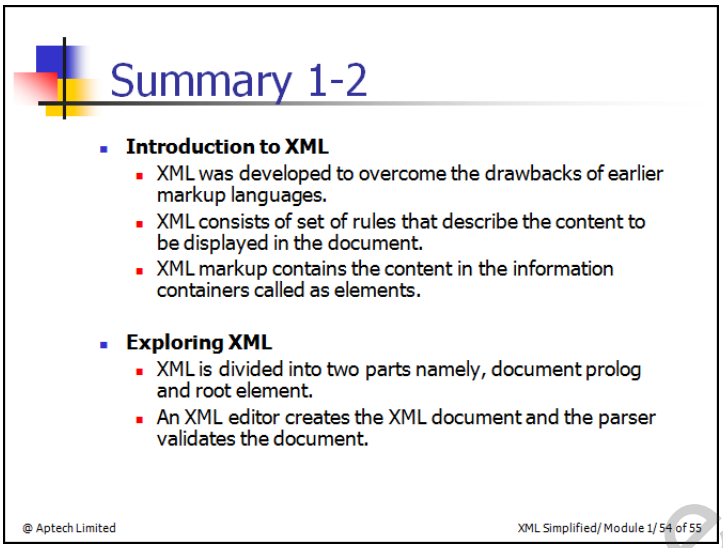
Tips:

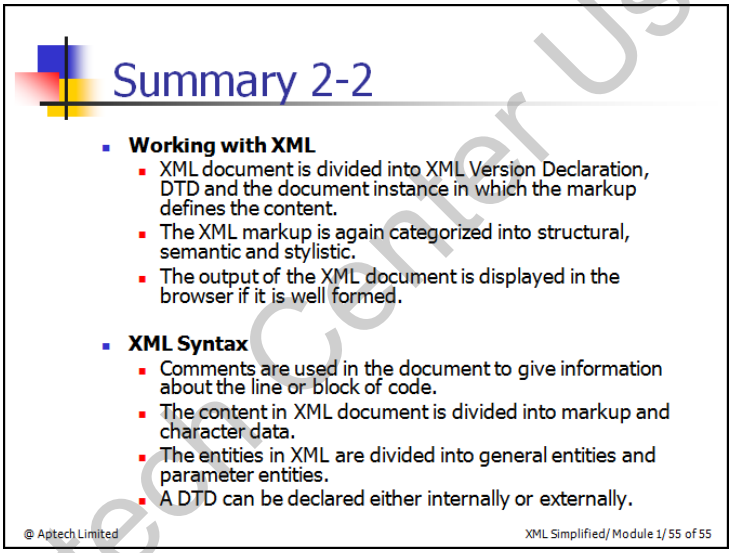
Some of the problems with using attributes are:

- Attributes cannot contain multiple values.
- Attributes cannot contain tree structures.
- Attributes are not easily expandable.

Slides 54 and 55

Let us summarize the session.





Using slides 54 and 55, summarize the session. End the session with a brief summary of what has been taught in the session.

1.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Namespaces that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 2 - Namespaces

2.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

2.1.1 Objectives

By the end of this session, the learners will be able to:

- XML Namespaces
- Working with Namespaces Syntax

2.1.2 Teaching Skills

To teach this session, you should be well-versed with XML namespaces and the reasons for using namespaces in XML documents. You should also aware yourself with the syntax of creating namespaces.

To teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

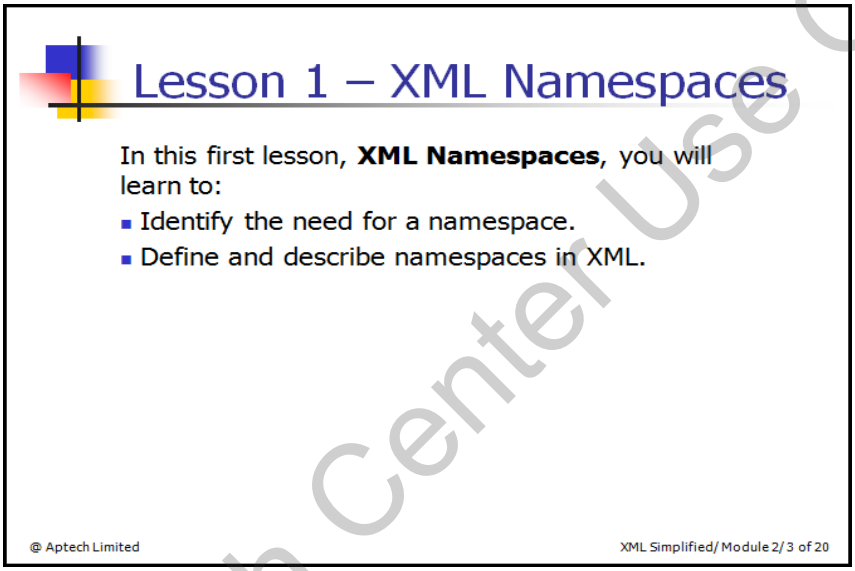
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces them to the need for a namespace. They will learn the syntax for namespace attribute in XML. They will also learn how to use default namespaces.

2.2 In-Class Explanations

Slide 3

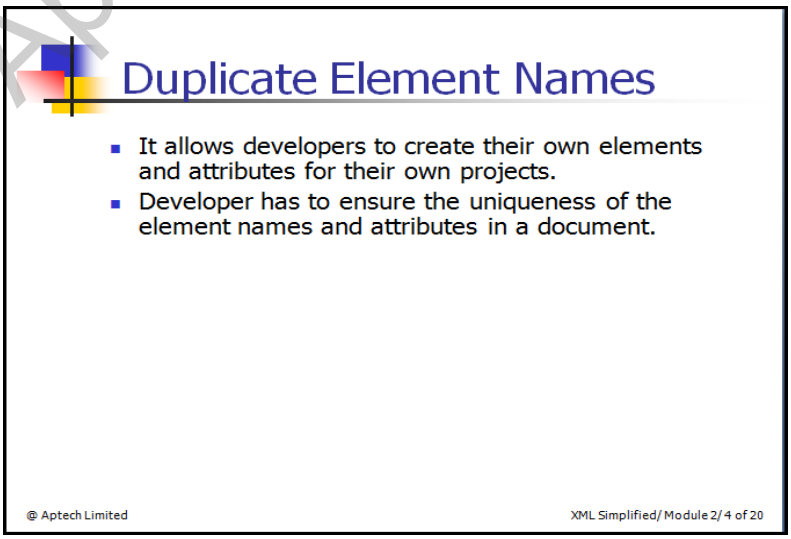
Let us understand the objectives of lesson 1.



Using slide 3, explain the topics to be covered in lesson 1.

Slide 4

Let us understand the duplicate elements name.



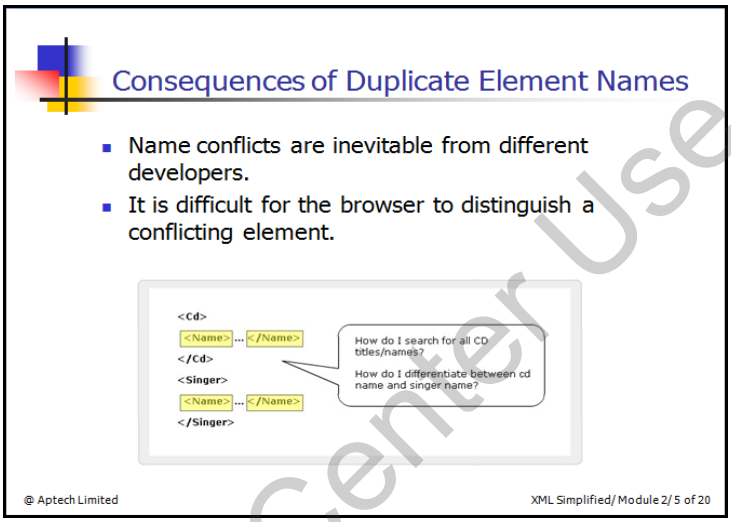
Using slide 4, explain the duplicate elements name.

Mention XML allows developers to create the XML documents with their own elements and attributes within the projects. Now, it may happen that element names defined by one develop may get repeated by some other developer while creating XML documents.

For example, an author of an XML document includes element called `<title>` in a `<CD>` element and another author creates an element `<title>` in a `<Book>` element. A problem will arise if the `<title>` element created by two different authors, is merged into a single XML document. Thus, XML developer has to ensure the uniqueness of the element names and attributes in a document.

Slide 5

Let us understand consequences of duplicate element names.



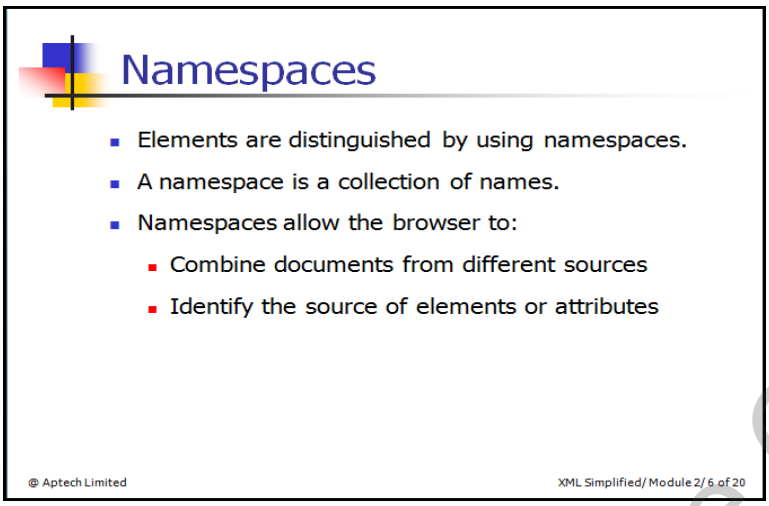
Using slide 5, explain the consequences of using duplicate element names.

When authors begin integrating the XML documents from different developers, name conflicts are inevitable. In such a scenario, it becomes difficult for the browser to distinguish a conflicting element in a XML document.

Explain the figure provided on the slide to explain the assigning of duplicate names to elements in an XML document.

Slide 6

Let us understand the namespaces.



Using slide 6, explain the namespaces.

In XML, elements are distinguished by using namespaces. XML namespaces provide a globally unique name for an element or attribute so that they do not conflict one another.

A namespace is a collection of names that can be used as element names or attribute names in XML document.

XML namespaces provide the following advantages:

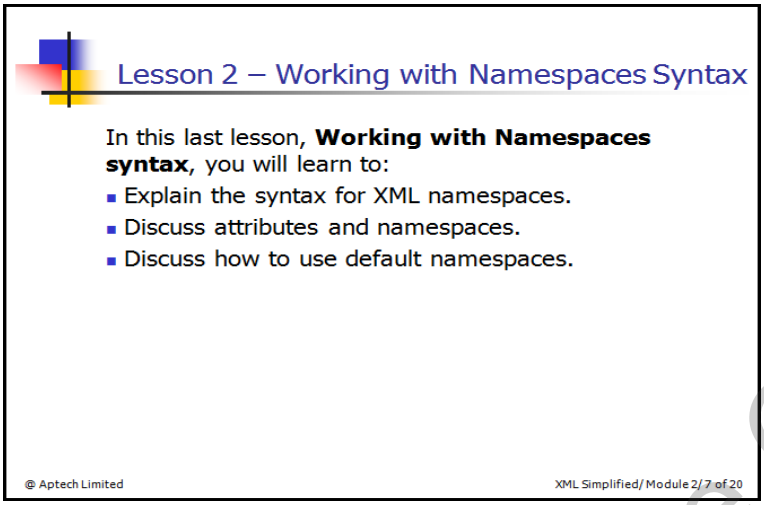
- **Reusability**
  XML Namespaces enable reuse of markup by making use of the elements and attributes that were defined earlier.

- **Modularity**
  Reusable code sessions can be written, and these can be invoked for specific elements or attributes. Elements and attributes from different sessions can be integrated into a single XML document. Universally unique element names and attributes guarantee that such sessions can be invoked for certain elements and attributes.

- **Extensibility**
  XML namespaces provide the XML documents with the ability to embed elements and attributes from other vocabularies such as MathML, eXtensible HyperText Markup Language (XHTML), and so forth.

Slide 7

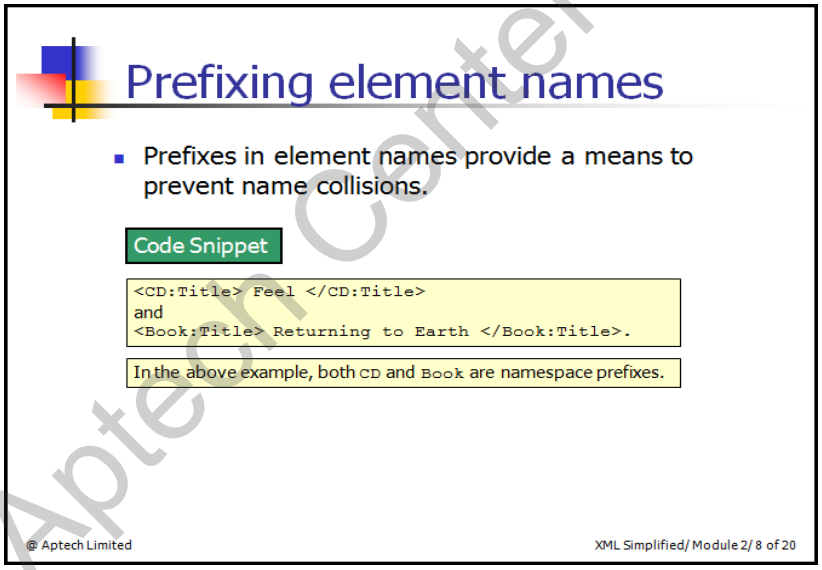
Let us understand the objectives of lesson 2.



Using slide 7, explain the topics to be covered in the lesson 2.

Slide 8

Let us understand how to prefix an element name.



Using slide 8, explain how to prefix an element name.

Namespaces are a mechanism by which element and attribute names can be assigned to groups. They also ensure that there is no conflict within element names. The best way to solve this problem is for every element in a document to have a completely distinct name. Therefore, the XML namespace prefixes are directly attached to names as well as names of its attributes and descendants.

Mention, using prefixes in the element names provide a means for the document authors to prevent name collisions, as demonstrated in the code shown in slide 8.

Explain the same scenario of `<title>` duplication in `<CD>` and `<Book>`, demonstrate prefixing of the elements with namespace prefix.

<CD:Title> Feel </CD:Title>

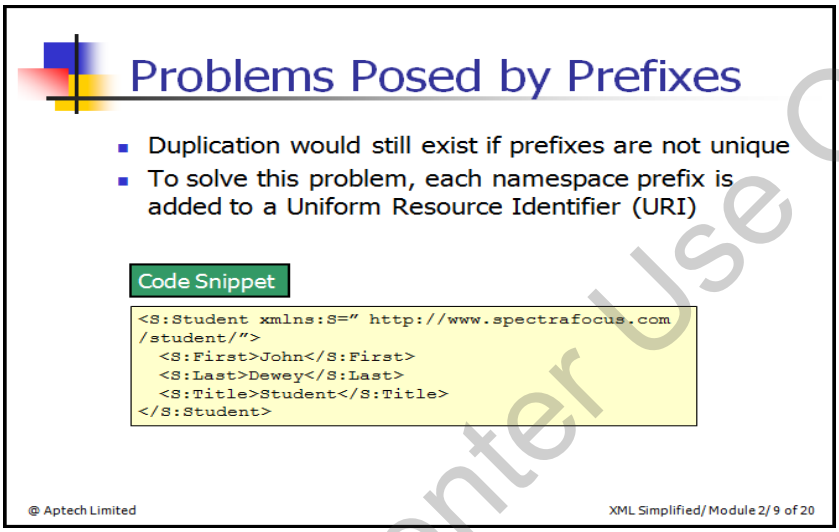
and

<Book:Title> Returning to Earth </Book:Title>

Here, both `CD` and `Book` are namespace prefixes.

Slide 9

Let us understand the problems posed by prefixes.



Using slide 9, explain the problems posed by prefixes.

There is a drawback to the prefix approach of the namespaces in an XML document. The reason for prefixing an element in a XML document is to prevent it from being duplicated. However, if the prefixes are not unique, the original problem of duplication would still exist.

To solve this problem, each namespace prefix is added to a Uniform Resource Identifier or URI that uniquely identifies the namespace.

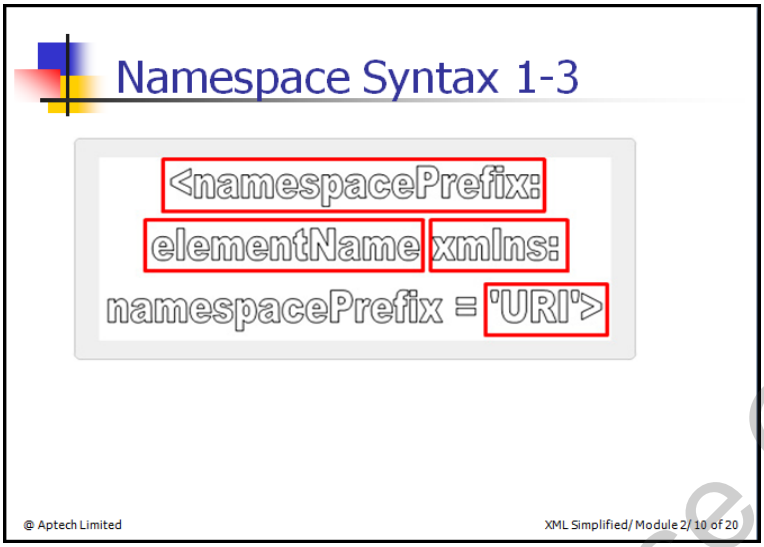
URI is a series of characters used to differentiate names.

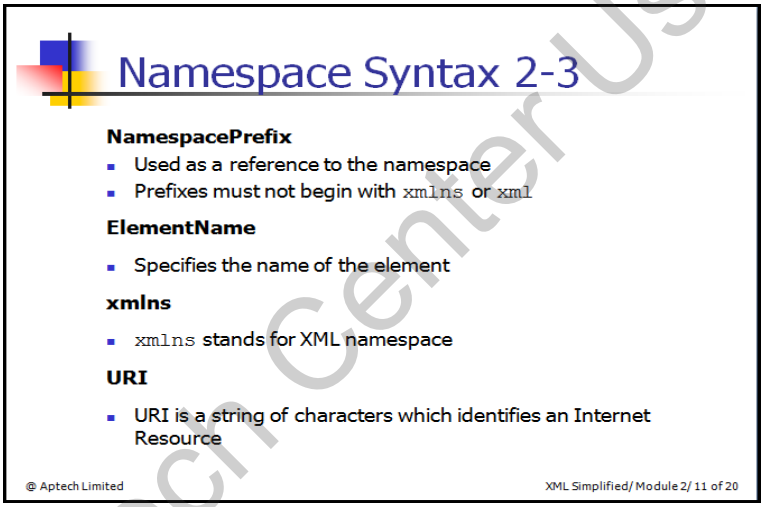
To guarantee name uniqueness in any XML documents that the company creates, the documents are associated with their namespace.

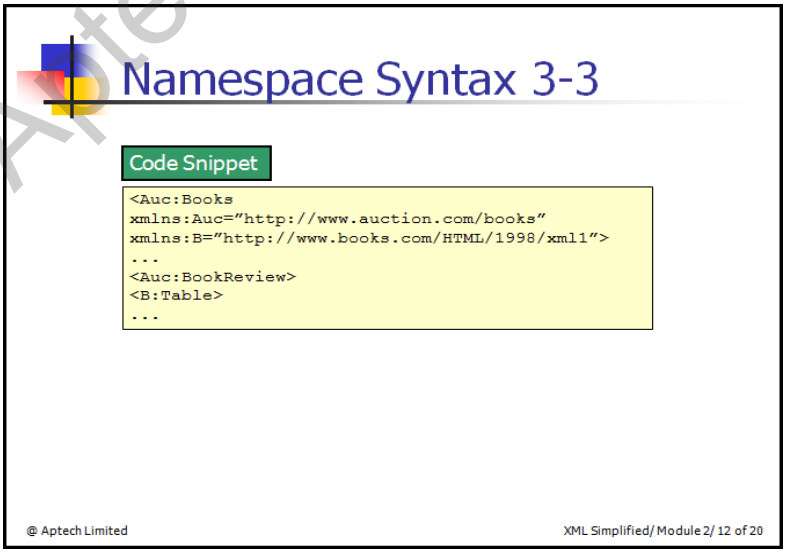
Explain the code snippet demonstrating a company's URI which is located at `http://www.spectrafocus.com`.

Slides 10 to 12

Let us understand the namespace syntax.







Using slides 10 to 12, explain the namespace syntax.

The XML namespace attribute is placed in the start tag of an element and the syntax for the namespace is given in figure.

- **namespacePrefix -** Each namespace has a prefix that is used as a reference to the namespace. Prefixes must not begin with `xmlns` or `xml`. A namespace prefix can be any legal XML name that does not contain a colon. A legal XML name must begin with a letter or an underscore.
- **elementName -** It specifies the name of the element.
- **Xmlns -** The `xmlns` attribute is what notifies an XML processor that a namespace is being declared. `xmlns` stands for XML namespace.
- **URI –** A URI is a string of characters which identifies an Internet Resource. The URI includes Uniform Resource Name (URN) and a Uniform Resource Locator (URL). URLs contain the reference for a document or an HTML page on the Web. URIs are also case-sensitive.

Tips:

The namespace URI is not parsed by the parser. Thus, its purpose is to uniquely identify the namespace.

In-Class Question:

After you finish explaining the namespace syntax, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

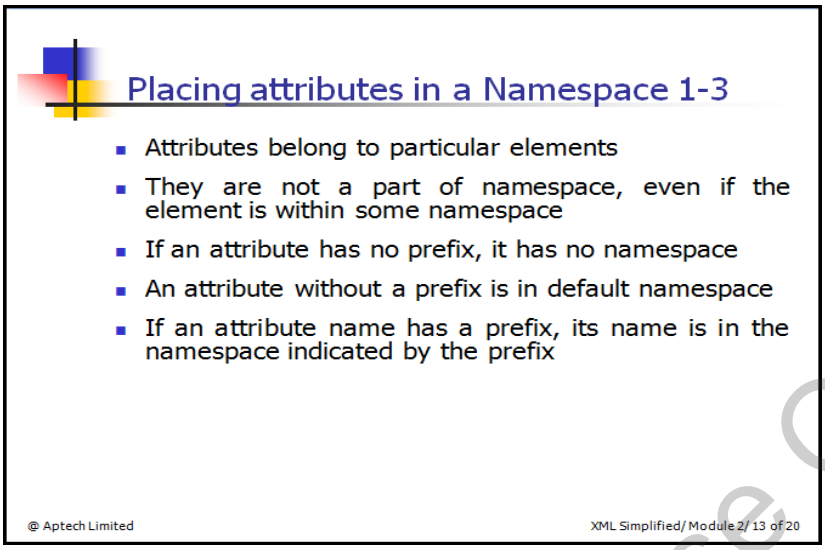
Whether it is important to have valid URIs names for the resources?

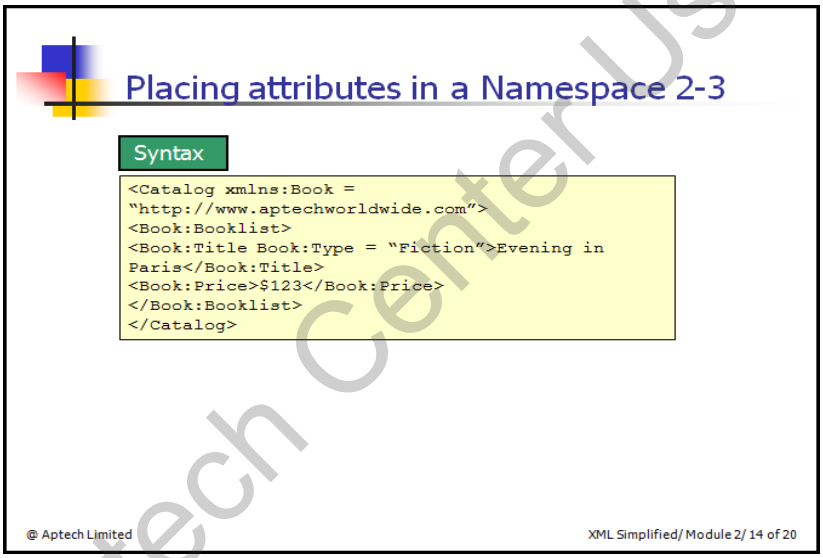
Answer:

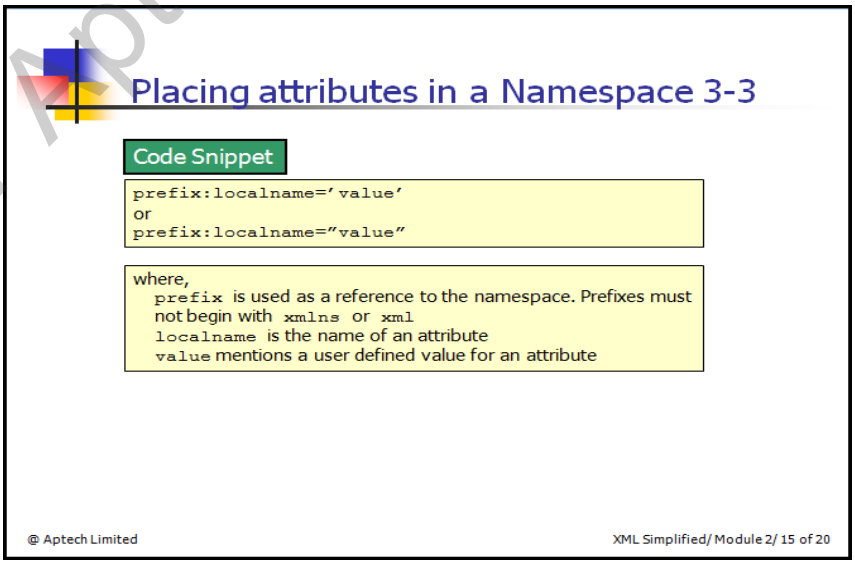
Not necessary, URIs do not need to be valid, as they are treated just like strings by XML namespace. Thus, these URIs are valid: `http://www.xmlnamespaces.com` or `http://xmlnamespaces.com`.

Slides 13 to 15

Let us understand how to place attributes in a namespaces.







Using slides 13 to 15, explain how to place attributes in a namespaces.

Attributes belong to particular elements and they are not a part of namespace, even if the element is within some namespace. However, if an attribute name has no prefix, it has no namespace. An attribute without a prefix is in default namespace.

If an attribute name has a prefix, its name is in the namespace indicated by the prefix.

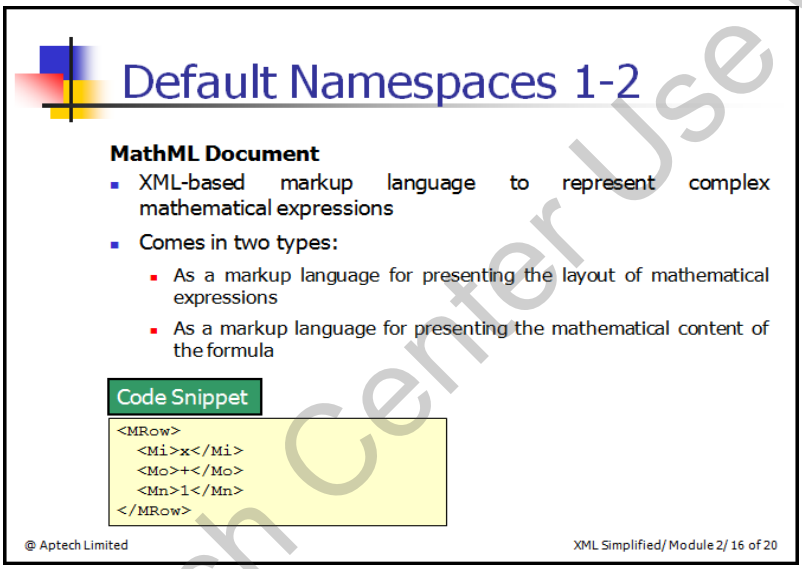
Syntax: Emp

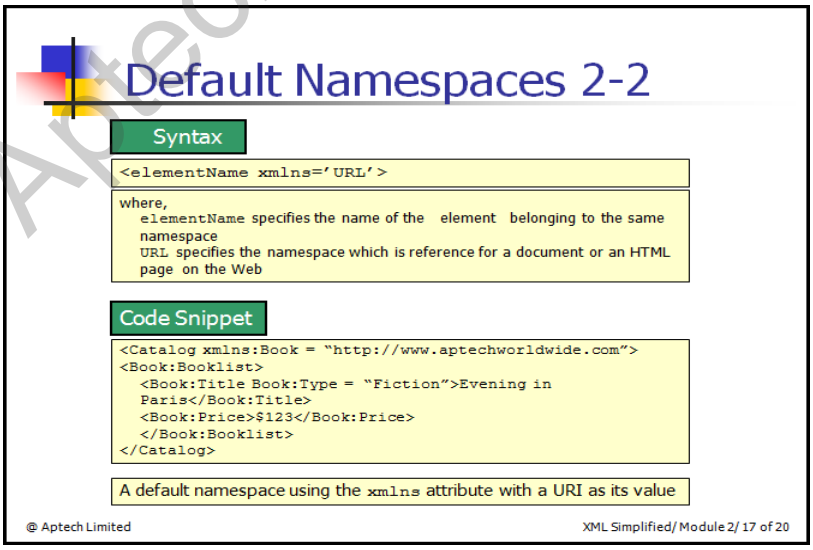
`xmlns:prefix="URI"`

Explain code using slides 14 and 15 in which the `type` attribute is associated with the `Book` namespace, since it is preceded by the prefix `Book`.

Slides 16 and 17

Let us understand the default namespaces.





Using slide 16, explain the default namespaces.

Consider an XML document which contains the details of all books in a library. Since this document includes a lot of markup all in the same namespace, it may be inconvenient to add a prefix to each element name.

This problem can be solved by adding a default namespace to an element and to its child elements using an `xmlns` attribute with no prefix.

A default namespace is used by an element and its child elements if the element does not have a namespace prefix.

Syntax:

```
xmlns="namespaceURI"
```

Explain the code which demonstrates the default namespace using slide 17.

A default namespace using the `xmlns` attribute with a URI as its value. Once this default namespace is declared, child elements that are part of this namespace do not need a namespace prefix.

In-Class Question:

After you finish explaining the default namespaces, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

What are the drawbacks of omitting the namespace?

Answer:

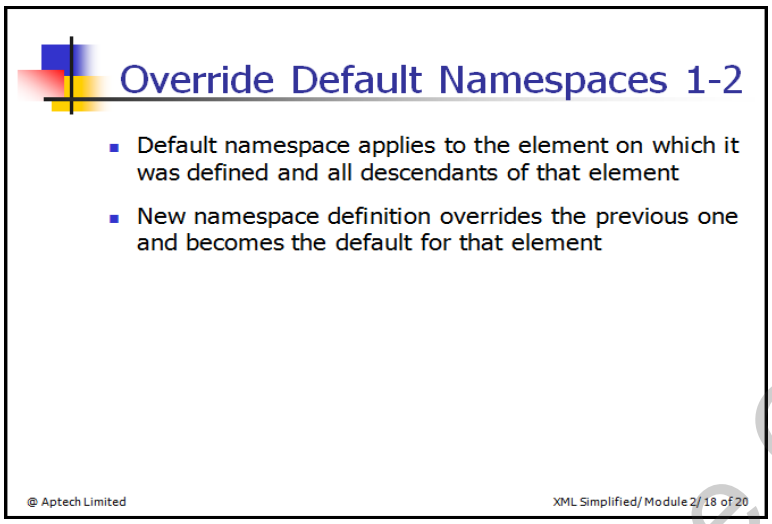
1. It is difficult to understand which element belongs to which namespace.
2. Also, which namespace is applicable is difficult to understand.

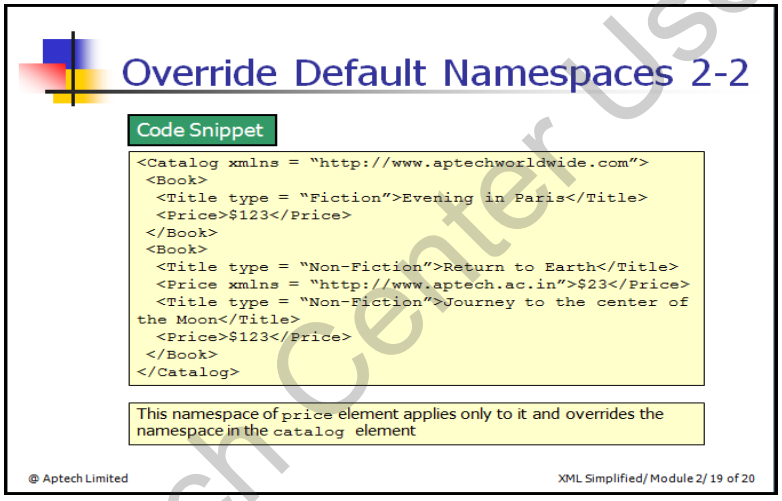
Tips:

When a default namespace is declared, the namespace is applied only to the element, and not to any attributes.

Slides 18 and 19

Let us understand the override default namespaces.





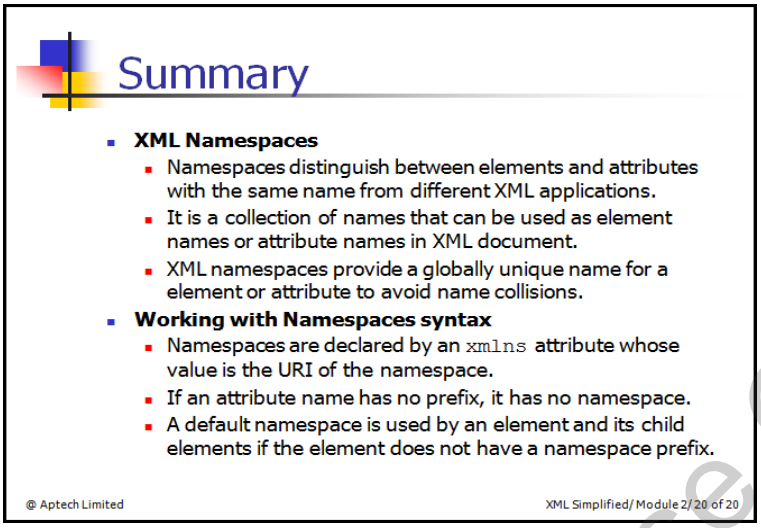
Using slides 18 and 19, explain the override default namespaces.

The default namespace applies to the element on which it was defined and all descendants of that element. If one of the descendants has another default namespace defined on it, this new namespace definition overrides the previous one and becomes the default for that element and all its descendants as demonstrated in the code.

Notice that in the `price` element in the second book element a different namespace is provided. This namespace applies only to the `price` element and overrides the namespace in the `catalog` element.

Slide 20

Let us summarize the session.



Using slide 20, summarize the session. End the session, with a brief summary of what has been taught in the session.

2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the DTDs that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 3 - DTDs

3.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

3.1.1 Objectives

By the end of this session, the learners will be able to:

- Document Type Definition
- Working with DTDs
- Valid XML Documents
- Declarations

3.1.2 Teaching Skills

To teach this session, you should be well-versed with how to create DTDs for XML files. The topics such as DOCTYPE declarations, types of DTDs, well-formedness, and validity of XML files. You should also aware yourself with the declaration of elements, attributes, and entities in a DTD.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

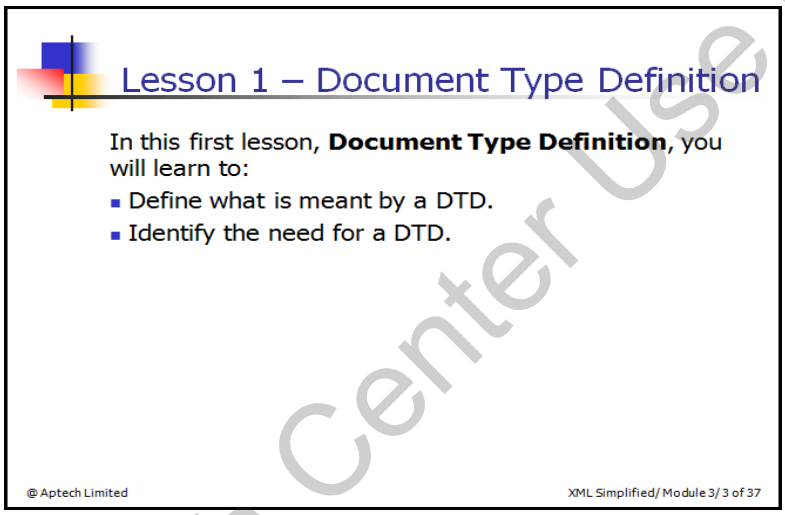
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces DTDs and how to create DTDs for XML files. They will learn the need for a DTD and its structure, DOCTYPE declarations, types of DTDs, well-formedness, and validity of XML files. They will also learn how to declare elements, attributes, and entities in a DTD.

3.2 In-Class Explanations

Slide 3

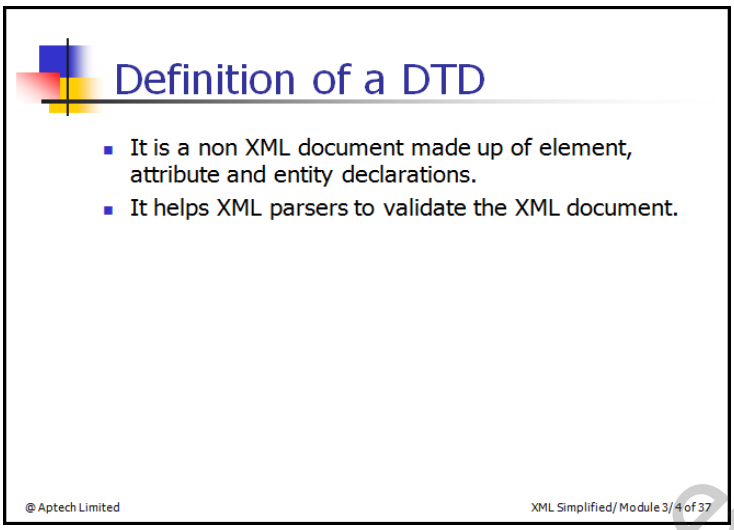
Let us understand the objectives of lesson 1.



Using slide 3, explain the topics covered in the lesson 1.

Slide 4

Let us understand the definition of a DTD.



Using slide 4, explain the definition of a DTD.

An XML parser is a software program that parses XML documents for its well-formedness. XML parsers are capable of validating XML files based on the document structure. Thus, to define the document structure, you need Document Type Definition (DTD).

DTD is a non-XML document made up of element, attribute, and entity declarations. The elements, attributes, and entities defined belong to the XML document(s), the DTD is defined for.

The DTD helps XML parsers to validate the XML document. It also helps authors to set default values for element attributes. The document's tree structure can be determined based on the DTD.

In other words, DTD defines the building blocks of an XML document. It defines the elements and attributes that forms the document structure for an XML document.

In-Class Question:

After you finish explaining the definition of DTD, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

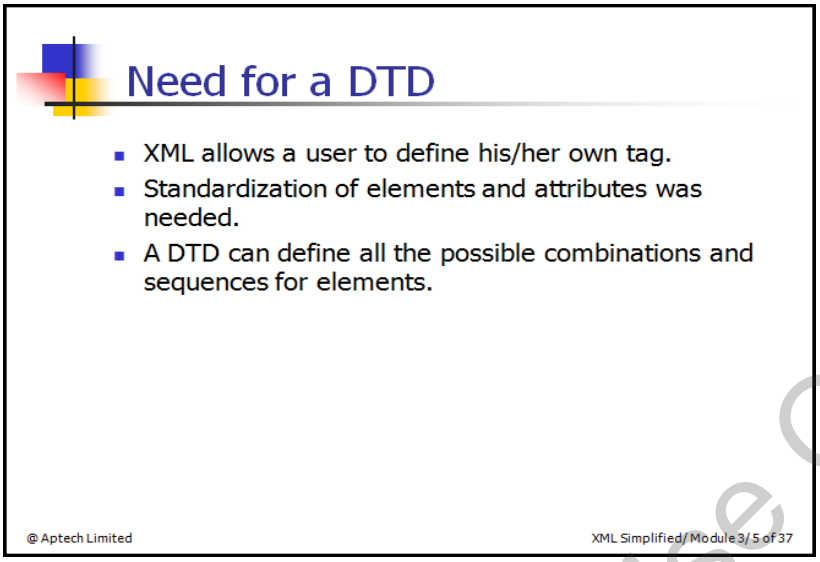
What are the main building blocks of XML documents?

Answer:

- Elements
- Attributes
- Entities
- CDATA and PCDATA

Slide 5

Let us understand the need for DTD.



Using slide 5, explain the need for DTD.

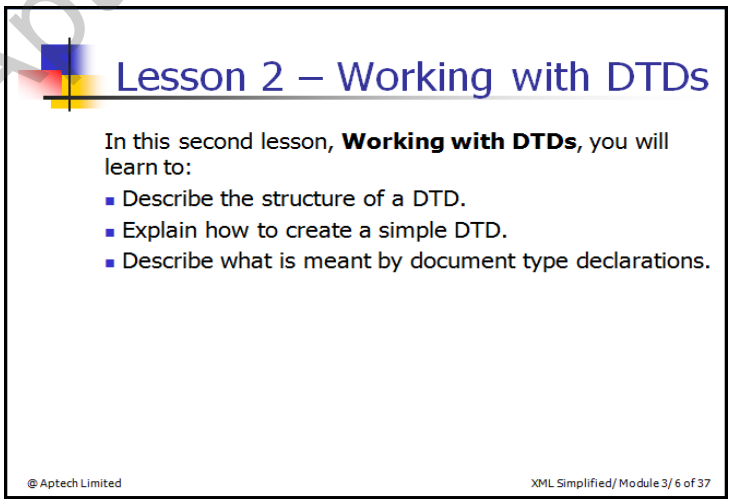
The fact that XML allowed its user to define his/her own tag was one of its biggest advantages. However, it also led to a genuine problem. People working on parts of the same XML document sometimes used similar tags for different purposes and at other times, used different tags for the same purpose. At times, same elements possessed or exhibited different attributes. Some sort of standardization of elements and attributes was needed.

That is how DTDs came into picture.

A DTD can define all the possible combinations and sequences for elements to be used in an XML document along with their attributes and their acceptable values.

Slide 6

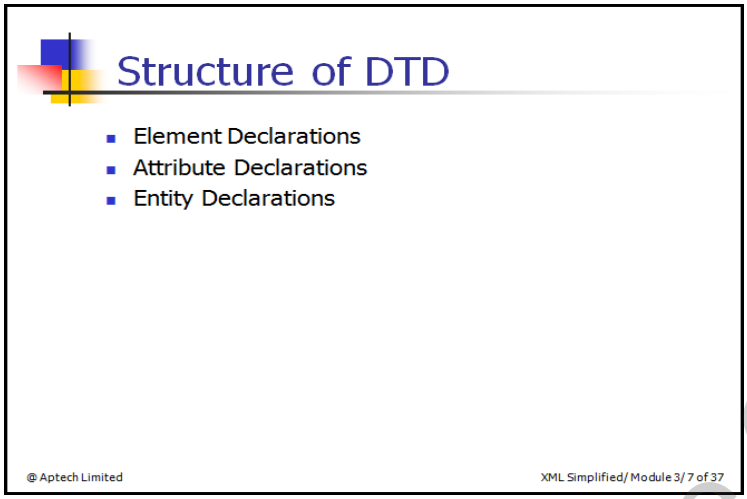
Let us understand the objectives of lesson 2.



Using slide 6, explain the topics covered in the lesson 2.

Slide 7

Let us understand the structure of DTD.



Using slide 7, explain the structure of DTD.

Explain that the DTD document contains element, attribute, and entity.

Each element declaration specifies the name of the element, and the content which that element can contain. Each attribute declaration specifies the element that owns the attribute, the attribute name, its type, and its default value (if any). Each entity declaration specifies the name of the entity and either its value or location of its value.

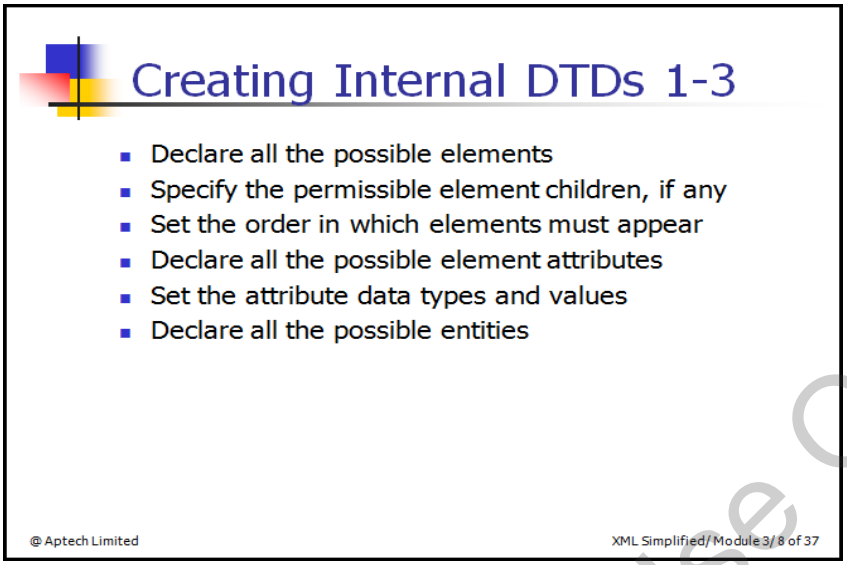
These groups of element, attribute, and entity declarations that form the three blocks of a DTD respectively can be declared in any sequence.

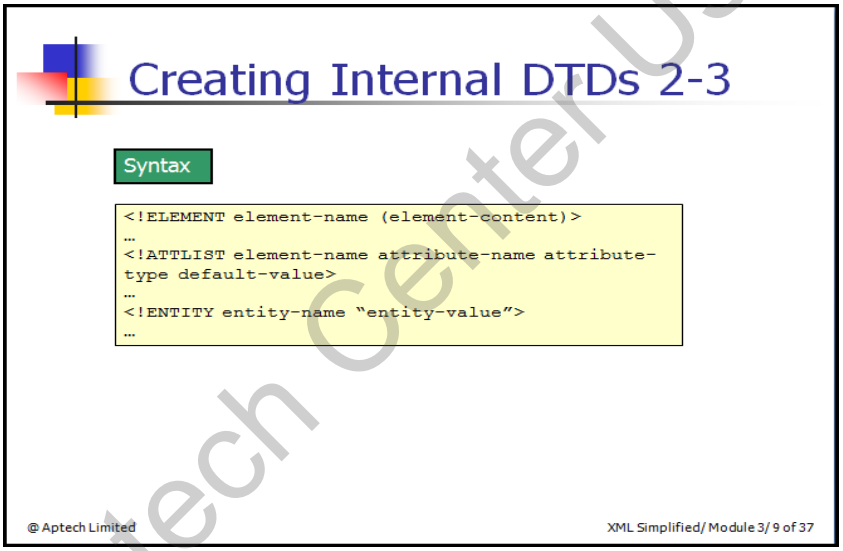
Tips:

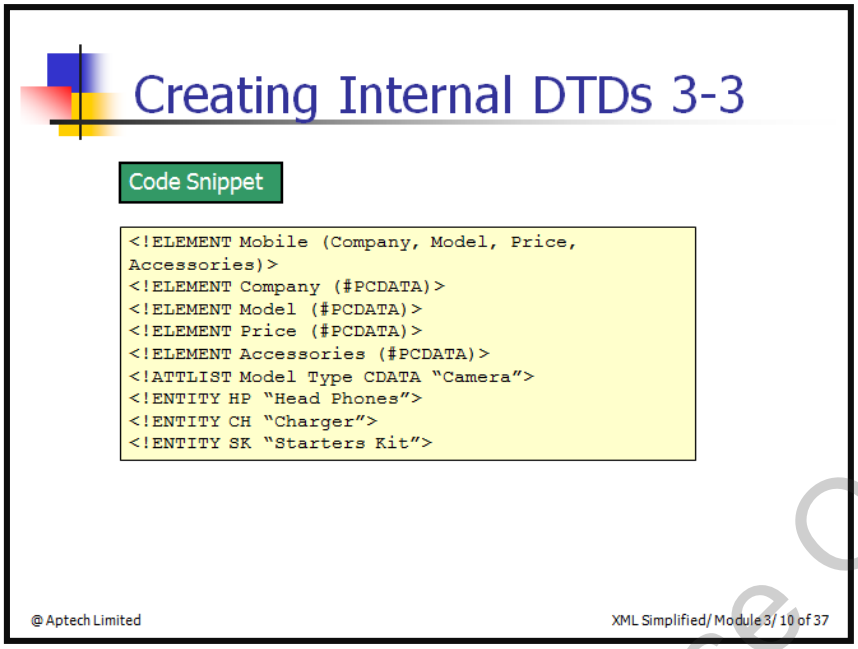
A DTD can be declared either inline, that is, inside an XML document, or as an external reference.

Slides 8 to 10

Let us understand how to create internal DTDs.







Using slides 8 to 10, explain how to create internal DTDs.

Mention that creating an internal DTD involves six step process.

Step 1 - Declare all the possible elements

In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

```
<!ELEMENT element-name (element-content)>
```

Example:

```
<!ELEMENT Mobile>
```

Step 2 - Specify the permissible element children, if any

Elements with one or more children are defined with the name of the children elements inside the parentheses:

```
<!ELEMENT element-name (child-element-name)>
```

Or

```
<!ELEMENT element-name (child-element-name,child-element-name,.....)>
```

Example:

```
<!ELEMENT Mobile (Company, Model, Price, Accessories)>
```

Step 3 - Set the order in which elements must appear

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document.

Example:

```
<!ELEMENT Mobile (Company, Model, Price, Accessories)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ELEMENT Accessories (#PCDATA)>
```

Tips:

In DTD declaration, the children must be declared. The children can also have children which must also be specified in the DTD declaration.

Step 4 - Declare all the possible element attributes

In the DTD, XML element attributes are declared with an `ATTLIST` declaration. An attribute declaration has the following syntax:

```
<!ATTLIST element-name attribute-name>
```

Example:

```
<!ATTLIST Model Type
```

Step 5 - Set the attribute data types and values

The `ATTLIST` declaration defines the element which can have the attribute, the name of the attribute, the type of the attribute, and the default attribute value.

Example:

```
<!ATTLIST Model Type CDATA "Camera">
```

Step 6 - Declare all the possible entities

Entities as variables used to define shortcuts to common text appearing in the XML document. It has the following syntax:

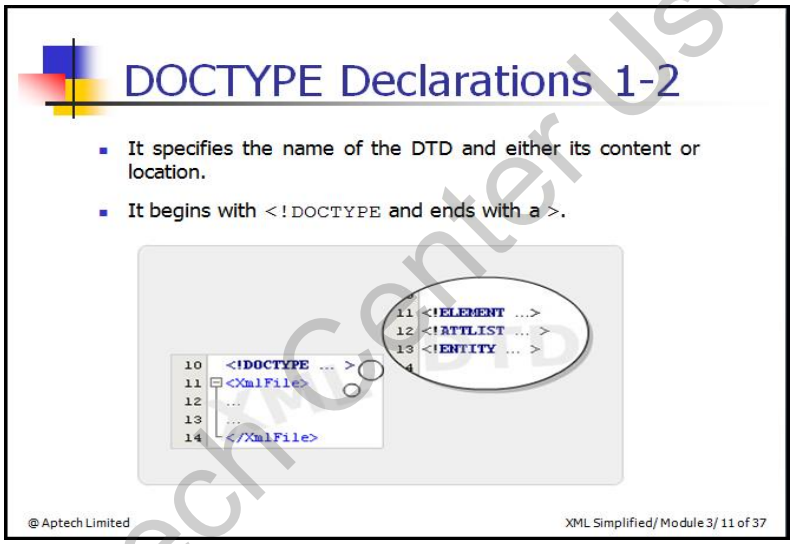
```
<!ENTITY entity-name "entity-value">
```

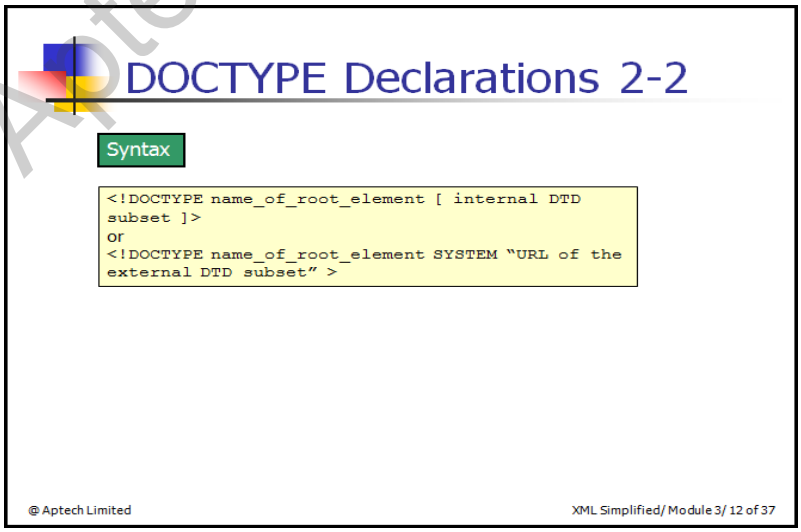
Example:

```
<!ENTITY HP "Head Phones">
<!ENTITY CH "Charger">
<!ENTITY SK "Starters Kit">
```

Slides 11 and 12

Let us understand the DOCTYPE declaration.





Using slides 11 and 12, explain the DOCTYPE declarations.

Placed in the XML document's prolog, a `DOCTYPE` declaration begins with `<!DOCTYPE` and ends with a `>`. In between is the name of the root element, followed either by a pair of square brackets containing the DTD itself or by the `SYSTEM` keyword and a URL specifying where the DTD can be found.

Without the document type declaration, a DTD is nothing but plain text.

Syntax:

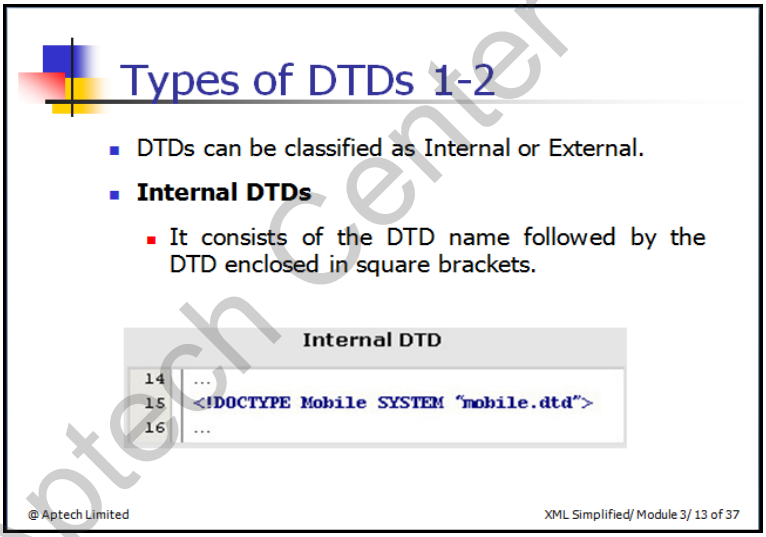
```
<!DOCTYPE name_of_root_element [ internal DTD subset ]>
```

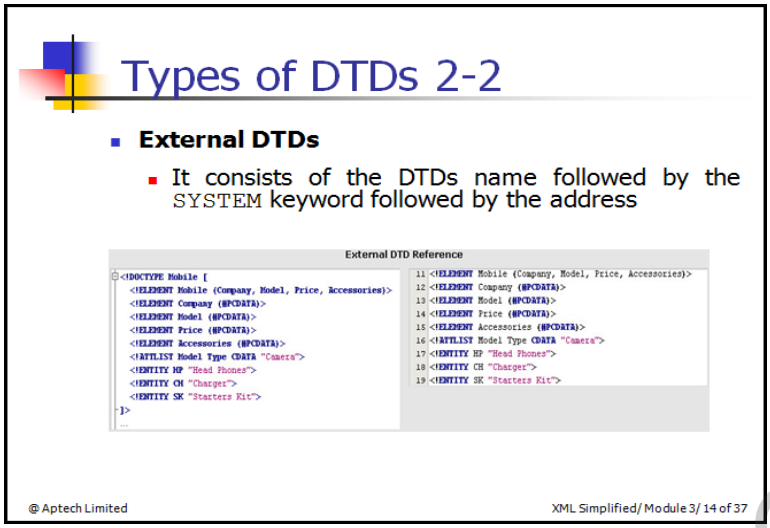
Or

```
<!DOCTYPE name_of_root_element SYSTEM "URL of the external DTD subset" >
```

Slides 13 and 14

Let us understand the types of DTDs.





Using slides 13 and 14, explain the types of DTDs.

DOCTYPE declarations have two basic forms of syntax, depending on the DTD used.

DTDs can be classified as Internal or External DTDs.

If the DTD is declared inside the XML file, it should be wrapped in a DOCTYPE definition. The DOCTYPE definition syntax is as follows:

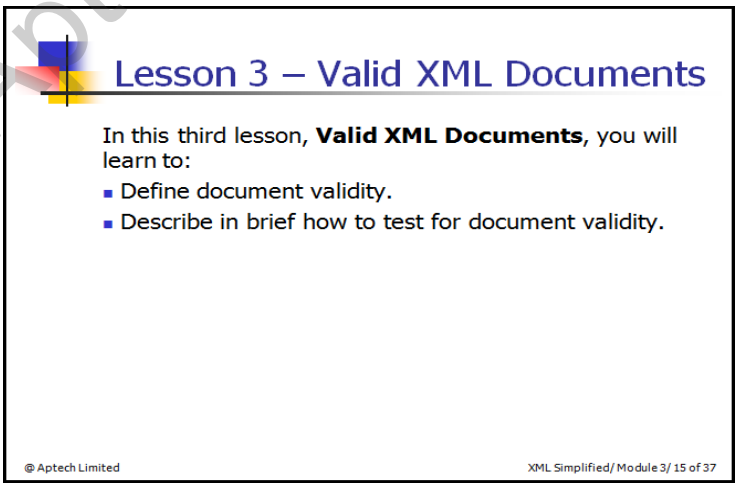
```
<!DOCTYPE root-element [element-declarations]>
```

If the DTD is declared in an external file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element SYSTEM "filename">
```

Slide 15

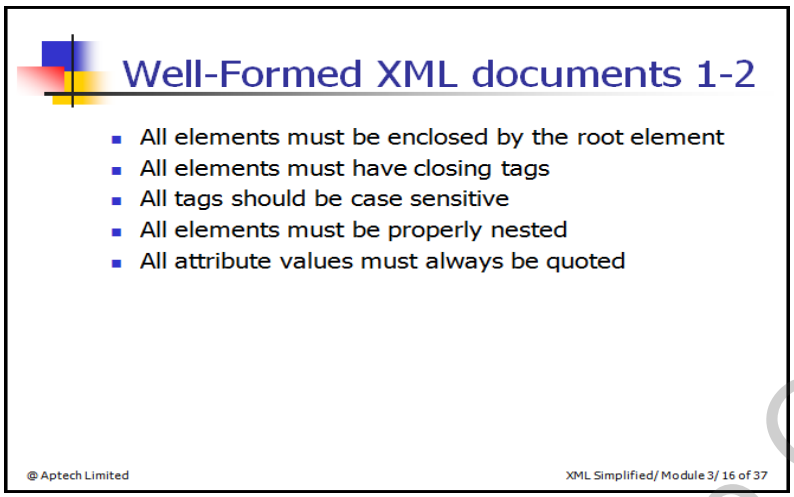
Let us understand the objectives of lesson 3.

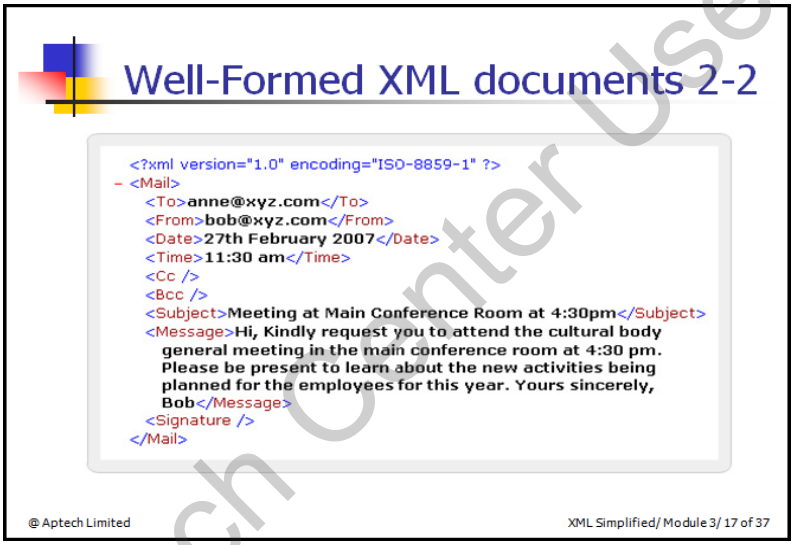


Using slide 15, explain the topics covered in the lesson 3.

Slides 16 and 17

Let us understand the well-formed XML document.





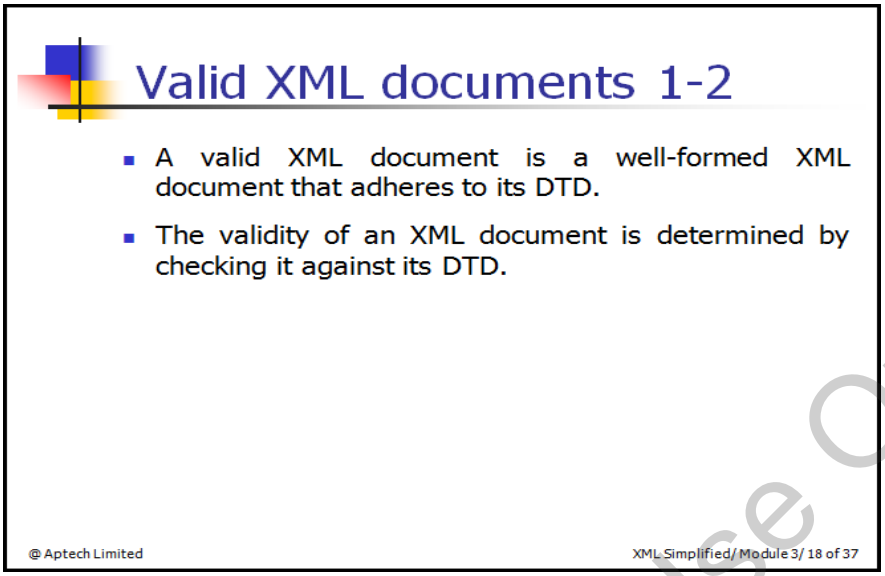
Using slides 16 and 17, explain the well-formed XML document.

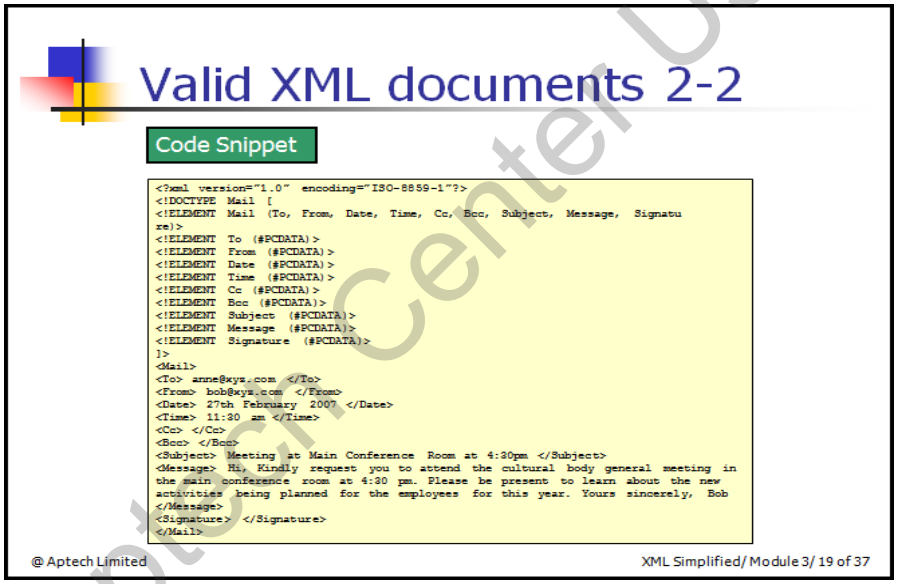
For an XML document to execute properly, it should be well-formed. A well-formed XML document adheres to the basic XML syntax rules. The World Wide Web Consortium in its specifications, states that XML documents with errors should not be processed by any program.

Explain the basic XML syntax rules and example of well-formed XML document provided on the slides.

Slides 18 and 19

Let us understand the valid XML document.





Using slides 18 and 19, explain the valid XML document.

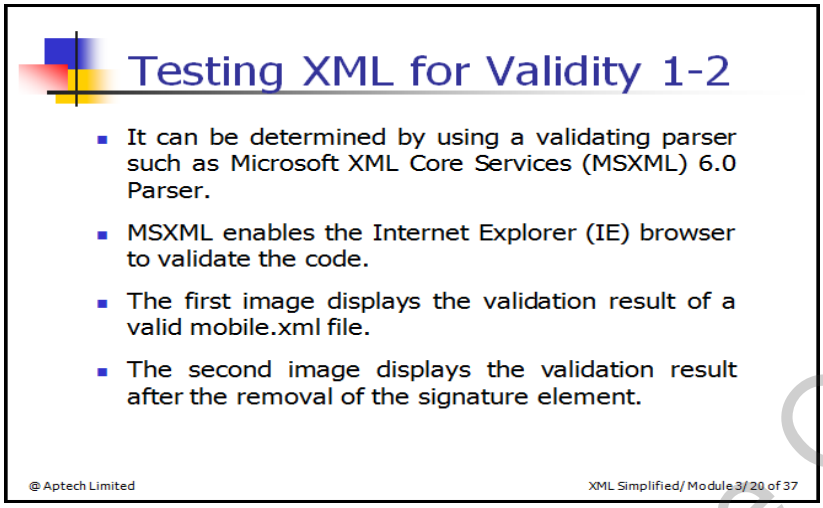
A valid XML document is a well-formed XML document that adheres to its DTD. The validity of an XML document is determined by checking it against its DTD. Once it has been confirmed that the components used in the XML document adhere to the declarations in the DTD, a well-formed XML document can be termed as a valid XML document.

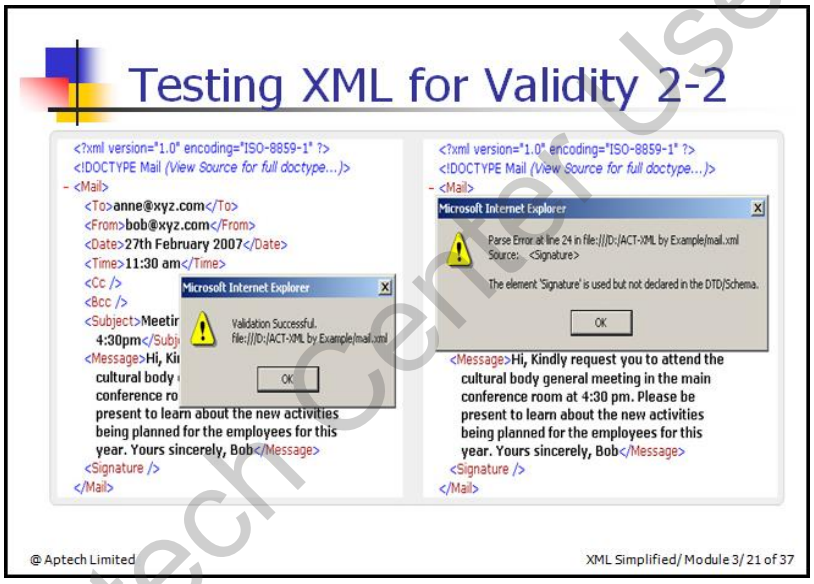
Validity of XML documents plays an important role in all XML applications. Be it creating modular parts of the code, or data interchange, or processing of transferred data, or database access, and so forth, validity of the XML document is a recommended feature.

Explain the code on slide 19 which demonstrates a valid XML document.

Slides 20 and 21

Let us understand how to test XML for validity.





Using slides 20 and 21, explain how to test XML for validity.

Validity being a desirable trait, it is necessary to be able to validate XML documents after their creation. Validity of XML documents can be determined by using a validating parser such as MSXML 6.0 Parser.

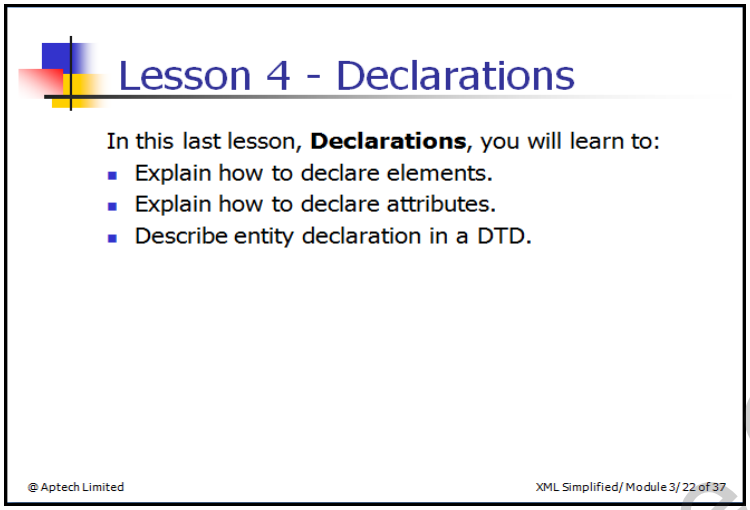
MSXML enables the Internet Explorer (IE) browser to validate the code. Once the code is displayed in IE, right-click the code to display the context menu. The menu provides the option of validating the code. On selection, IE internally validates the code against the code's DTD.

The first figure on slide 21 displays the validation result of a valid `mobile.xml` file.

The second figure on slide 21 displays the validation result after the removal of the `signature` element from the document's DTD.

Slide 22

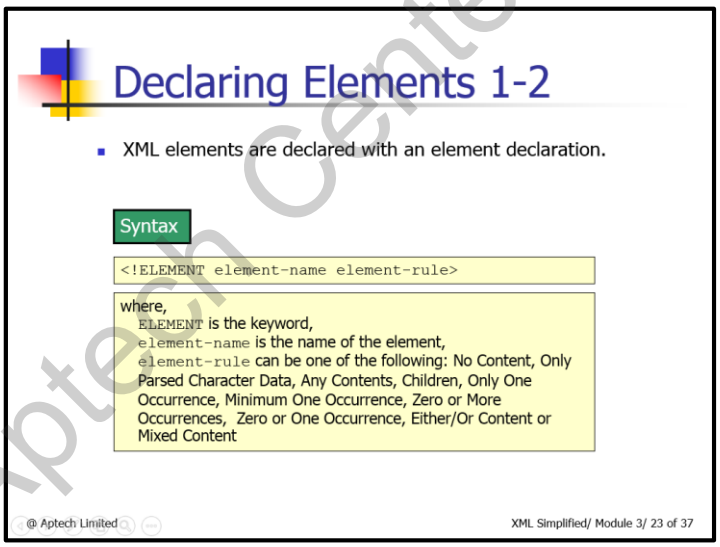
Let us understand the objectives of lesson 4.

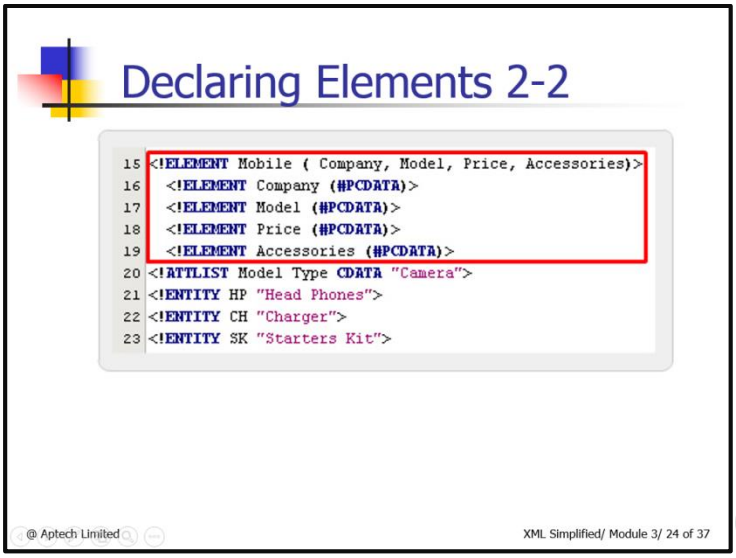


Using slide 22, explain the topics covered in the lesson 4.

Slides 23 and 24

Let us understand how to declare elements.





Using slides 23 and 24, explain how to declare elements.

In the DTD, XML elements are declared with an element declaration. Explain the syntax for declaring elements.

Some of the element-rule options for the element declaration are as follows:

- **Empty Elements**

Empty elements are declared with the category keyword **EMPTY**.

```
<!ELEMENT element-name EMPTY>
```

Example:

```
<!ELEMENT br EMPTY>
XML example: <br />
```

- **Elements with Parsed Character Data**

Elements with only parsed character data are declared with #PCDATA inside parentheses.

```
<!ELEMENT element-name (#PCDATA)>
```

Example:

```
<!ELEMENT from (#PCDATA)>
```

- **Elements with any Contents**

  Elements declared with the category keyword **ANY**, can contain any combination of parsable data.

```
<!ELEMENT element-name ANY>

Example:

<!ELEMENT note ANY>
```

- **Elements with Children (sequences)**

  Elements with one or more children are declared with the name of the children elements inside parentheses.

```
<!ELEMENT element-name (child1)>
Or
<!ELEMENT element-name (child1,child2,...)>

Example:

<!ELEMENT message (to, from, subject, body)>
```

- **Declaring Only One Occurrence of an Element**

```
<!ELEMENT element-name (child-name)>

Example:

<!ELEMENT message (subject)>
```

The example declares that the child element "subject" must occur once, and only once inside the "message" element.

- **Declaring Minimum One Occurrence of an Element**

```
<!ELEMENT element-name (child-name+)>

Example:

<!ELEMENT message (subject+)>
```

The + sign in the example declares that the child element "subject" must occur one or more times inside the "message" element.

- **Declaring Zero or More Occurrences of an Element**

```
<!ELEMENT element-name (child-name*)>
```

Example:

```
<!ELEMENT message (subject*)>
```

The * sign in the example declares that the child element "subject" can occur zero or more times inside the "message" element.

- **Declaring Zero or One Occurrence of an Element**

```
<!ELEMENT element-name (child-name?)>
```

Example:

```
<!ELEMENT note (message?)>
```

The ? sign in the example declares that the child element "message" can occur zero or one time inside the "note" element.

- **Declaring either/or Content**

Example:

```
<!ELEMENT message (to,from,subject,(header|body))>
```

The example declares that the "message" element must contain a "to" element, a "from" element, a "subject" element, and either a "header" or a "body" element.

- **Declaring Mixed Content**

Example:

```
<!ELEMENT message (#PCDATA|to|from|header|body)*>
```

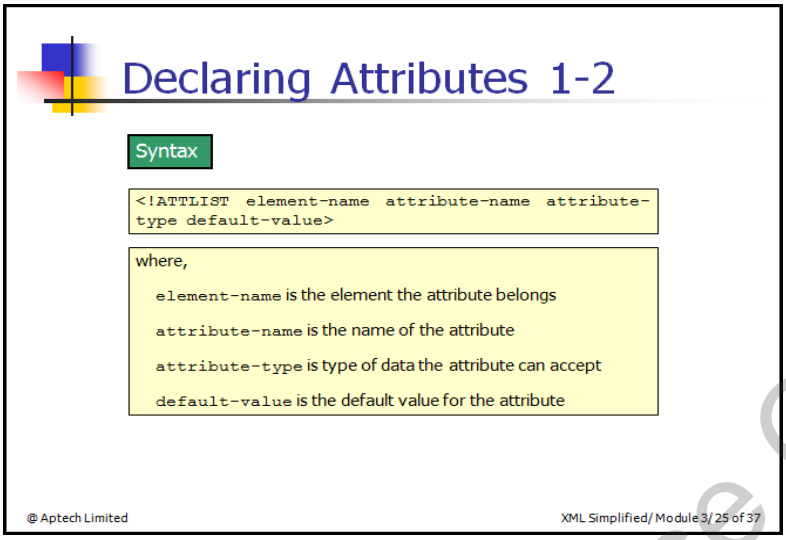
The example declares that the "message" element can contain zero or more occurrences of parsed character data, "to", "from", "header", or "body" elements.

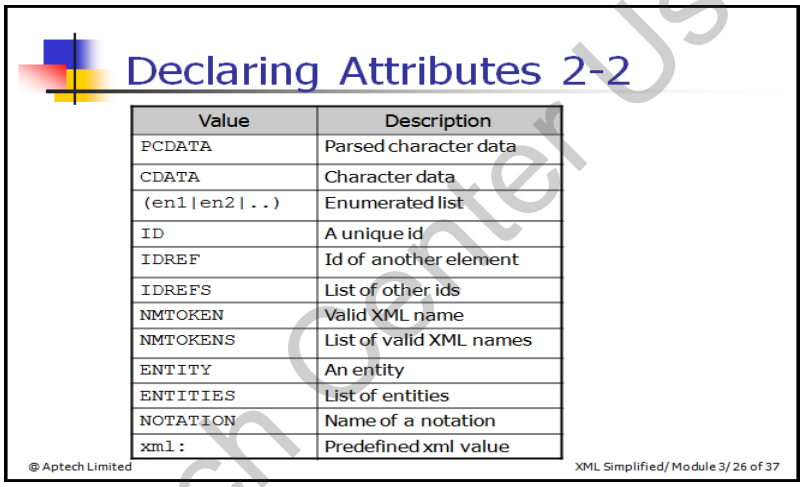
Tips:

When the element content is a text string or numeric value, the element is designated as #PCDATA. Under DTD, there is no difference between numeric type data and text type data.

Slides 25 and 26

Let us understand how to declare attributes.



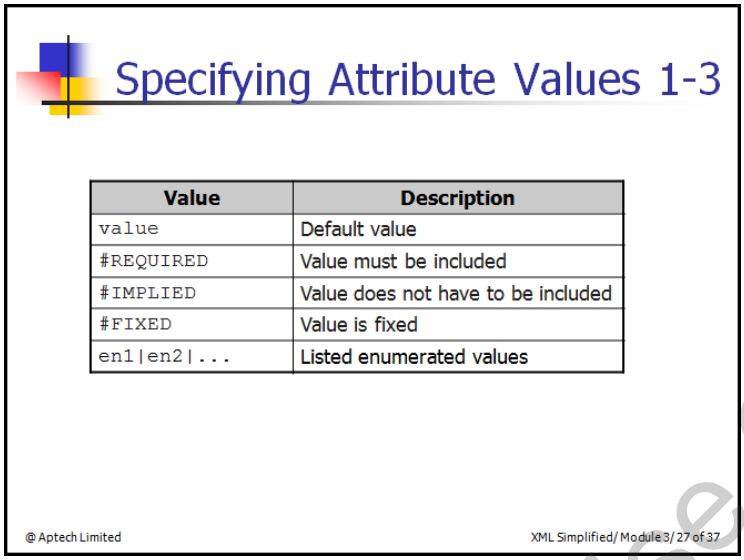


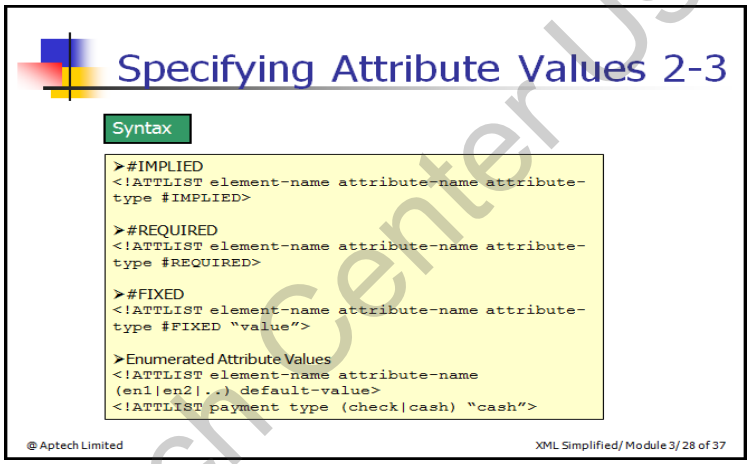
Using slides 25 and 26, explain how to declare attributes.

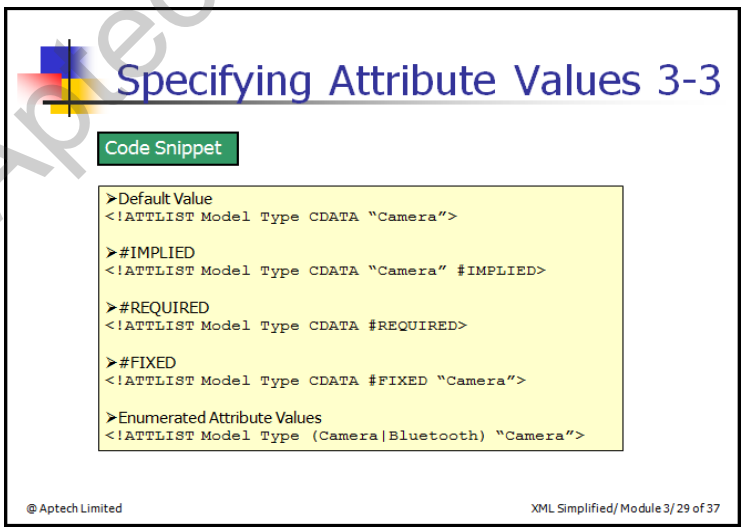
Explain the different attribute types that can be declared for the attribute declaration.

Slides 27 to 29

Let us understand how to specify the attribute values.







Using slides 27 to 29, explain how to specify the default values for the attribute.

The `attribute-value` in a DTD declaration can have default values.

The explanation for the `attribute-value` is as follows:

Default – Specifying a default value for an attribute value ensures that the attribute will be assigned a default value, even if the author of XML document does not assigns the value.

Syntax:
```
<!ATTLIST element-name attribute-name CDATA "default-value">
```

DTD example:
```
<!ATTLIST payment type CDATA "check">
```

XML example:
```
<payment type="check">
```

Implied – An implied attribute is used when you don't want to force the author to include an attribute.

Syntax:
```
<!ATTLIST element-name attribute-name attribute-type #IMPLIED>
```

DTD example:
```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML example:
```
<contact fax="555-667788">
```

Required - Use a `required` attribute to force the attribute to be present.

Syntax:
```
<!ATTLIST element-name attribute_name attribute-type #REQUIRED>
```

DTD example:
```
<!ATTLIST person number CDATA #REQUIRED>
```

XML example:
```
<person number="5677">
```

Fixed – Does not allow the author to change it. If an author includes another value, the XML parser will return an error.

Syntax:
```
<!ATTLIST element-name attribute-name attribute-type #FIXED "value">
```

DTD example:
```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

XML example:
```
<sender company="Microsoft">
```

Enumerated - When the attribute values needs to be one of a fixed set of values.

Syntax:
```
<!ATTLIST element-name attribute-name (eval|eval|..) default-value>
```

DTD example:
```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type="check">
or
<payment type="cash">
```

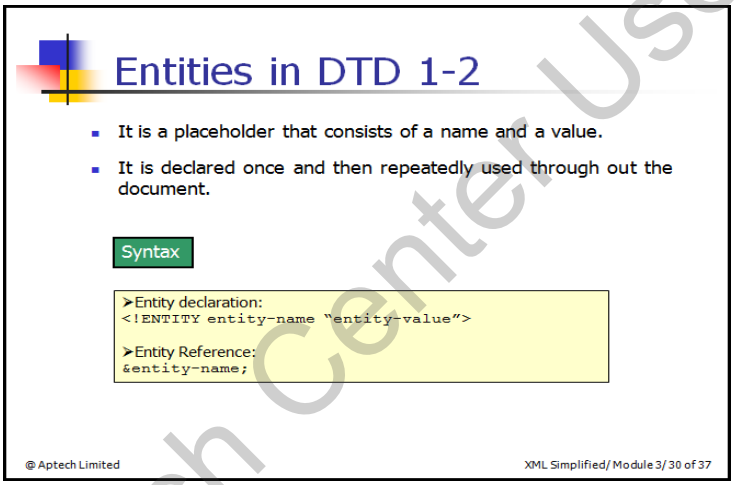
Tips:

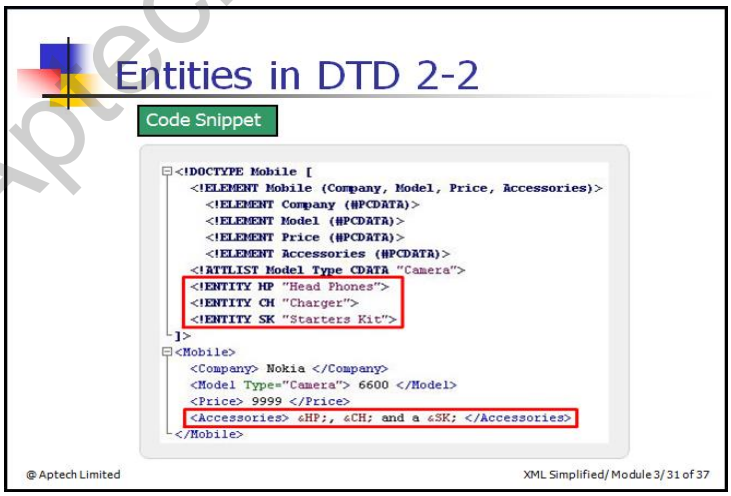
Some of the problems with attributes are:

- Attributes cannot contain multiple values
- Attributes are not easily expandable
- Attributes cannot describe structures
- Attributes are more difficult to manipulate by program code
- Attribute values are not easy to test against a DTD

Slides 30 and 31

Let us understand the entities in DTD.





Using slides 30 and 31, explain the entities in DTD.

An entity is a placeholder that consists of a name and a value. It is declared once and then repeatedly used throughout the document in place of its value.

Entities can be categorized as parameter entities and general entities.

When creating a parameter entity, you need to insert a percentage sign (%) between ENTITY and the name of the entity.

Syntax:

```
<!ENTITY % name definition>
```

General entities are further classified as:

- Character
- Mixed Content
- Unparsed

Character entities are entities that have a single character as their replacement value. Character entities are further classified as:

- Pre-defined
- Numbered
- Name

XML has a set of pre-defined entities to facilitate the use of characters such as <, >, &, ", ', since these are used by XML in its tags.

The **Pre-defined** entities and their references are shown in table 3.6.

Entity Name	Value
amp	&
apos	'
gt	>
lt	<
quot	"

Numbered entities are entities which use a character reference number, for a character from the Unicode character set, in place of the character itself. These entity references have a special syntax. Instead of the "&", "&#" is used. The reference number is used as the name, followed by a semicolon. Thus, a numbered entity reference would look like this: õ.

Name entities are the same as **Numbered** entities, except that the numbers are replaced by a descriptive name for the character.

Mixed content entities are entities that contain either text or markup language text as their replacement value. For example, the entity test could also have "trial" or

"<trial>Hi!</trial>" as its replacement value. The text "<trial>Hi!</trial>" will be replaced with the entity reference and will act as an element trial.

Unparsed entities are entities that have data that is not to be parsed. The replacement value might be an image or a file or something else. The syntax for an unparsed entity is as follows:

```
<!DOCTYPE name [
<!ENTITY entity-name SYSTEM "entity-location" NDATA Notation Identifier>
]>
```

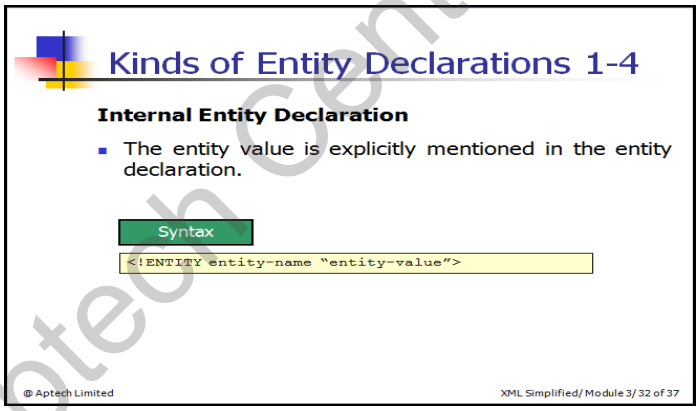
Syntax to declare and reference entity is as follows:

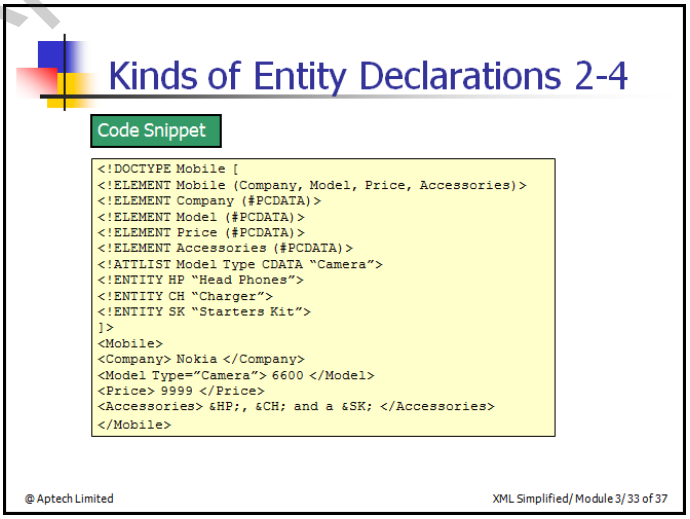
Entity Declaration:
```
<!ENTITY entity-name "entity-value">
```

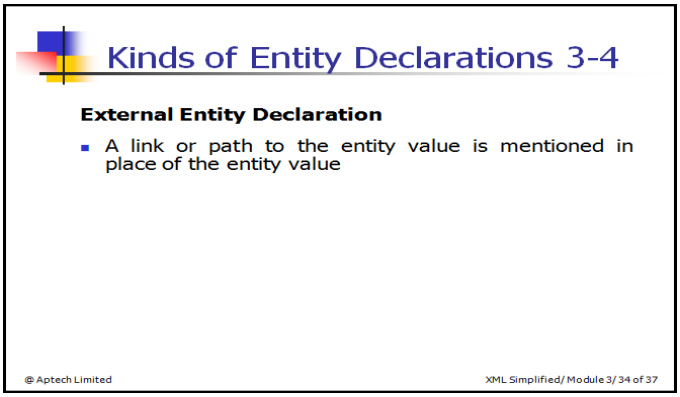
Entity Reference:
```
&entity-name;
```

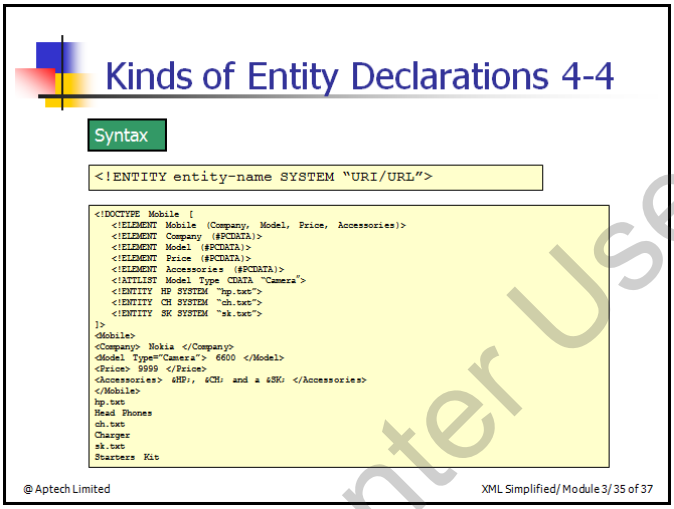
Slides 32 to 35

Let us understand the different kinds of entity declarations.









Using slides 32 to 35, explain the kind of entity declarations.

Entity references are used extensively in XML. That leads to Internal and External entity declarations, the two basic kinds of entity declarations used in XML.

In internal entity declarations, the entity value is explicitly mentioned in the entity declaration.

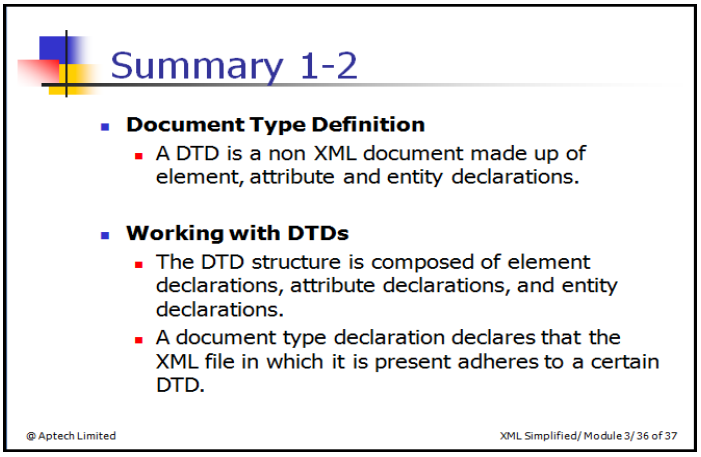
Explain the syntax of internal entities using slide 32. Also, explain the code on slide 33 which demonstrates the properties of internal entity declaration.

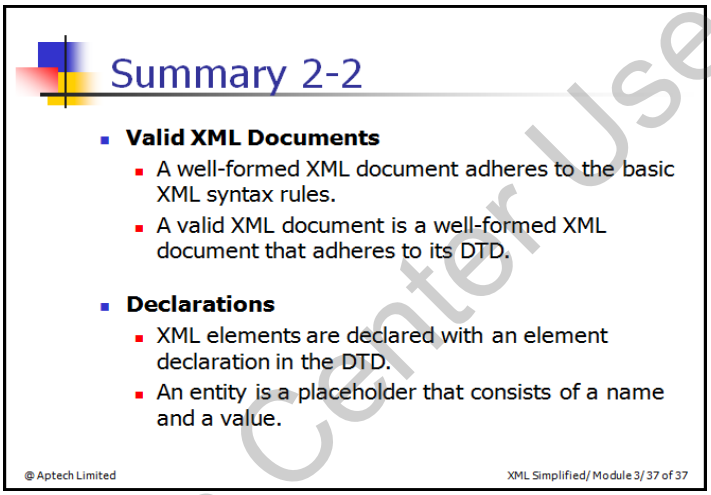
In external entity declaration, a link or path to the entity value is mentioned in place of the entity value with the help of the **SYSTEM** keyword. External entity declarations are in a way similar to declaring external DTDs.

Explain the syntax of internal entities using slide 34. Also, explain the code on slide 35 which demonstrates the properties of external entity declaration.

Slides 36 and 37

Let us summarize the session.





Using slides 36 and 37, summarize the session. End the session, with a brief summary of what has been taught in the session.

3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the XML Schema that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 4 - XML Schema

4.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

4.1.1 Objectives

By the end of this session, the learners will be able to:

- XML Schema
- Exploring XML Schemas
- Working with Complex Types
- Working with Simple Types

4.1.2 Teaching Skills

To teach this session, you should be well-versed with XML schema and its features. The topics such as structure of an XML document with the help of schema, different elements, and data types supported by an XML schema.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

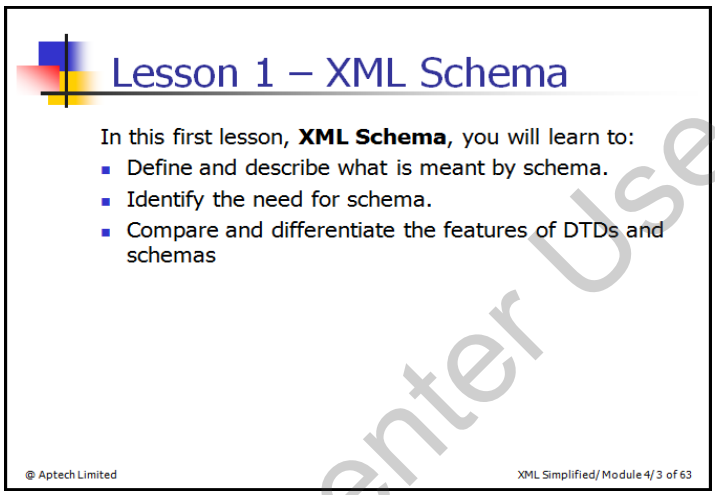
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces XML schema and its features. They will learn how to define the structure of an XML document with the help of schema. They will also understand the different elements and data types supported by an XML schema.

4.2 In-Class Explanations

Slide 3

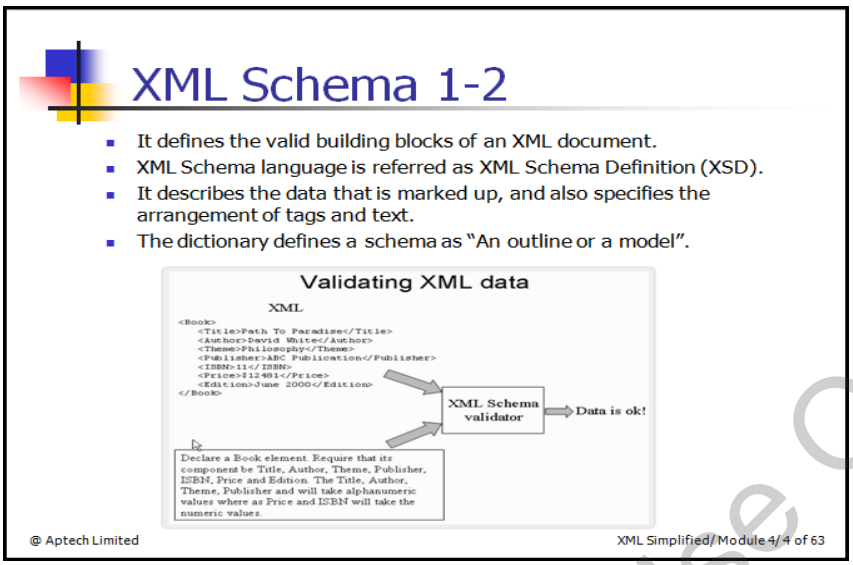
Let us understand the objectives of lesson 1.

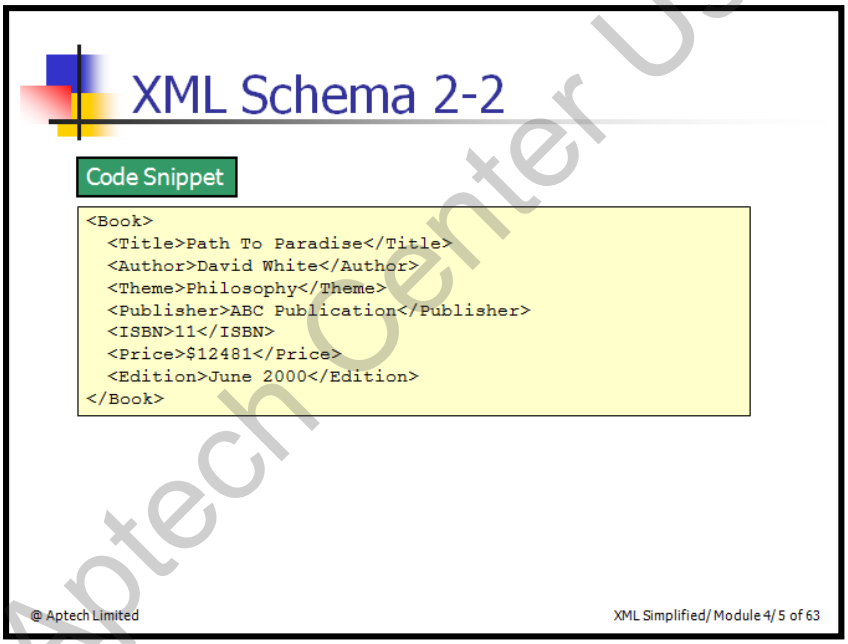


Using slide 3, explain the topics covered in the lesson 1.

Slides 4 and 5

Let us understand the XML schema.





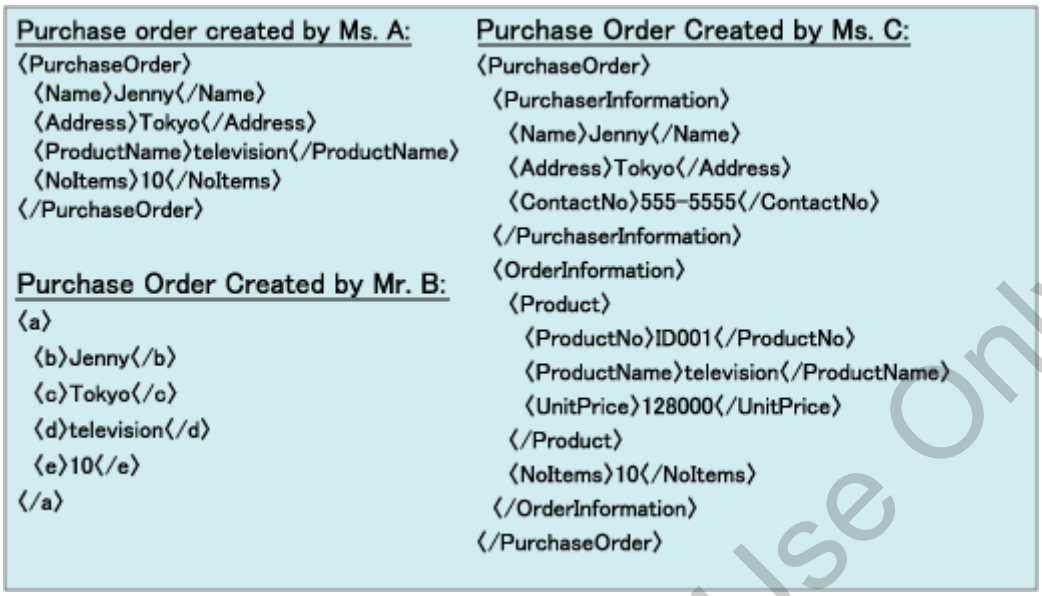
Using slides 4 and 5, explain the XML schema.

Consider a scenario where requested to create an XML document to serve as a purchase order to be sent to Mr. Y at Company X, what kind of XML document would you create?

Following are three XML document examples, created by three different individuals.

- Ms. A has created very semantic element names.
- Mr. B has opted for rather abbreviated element names.
- Ms. C has created elements having a hierarchical structure.

Following figure shows the creation of purchase order document created by the three individuals:



Any of the three examples can be considered to be proper XML documents. As long as the information required for a purchase order is included, likely any XML document you could create would be a valid XML document for the purpose.

What would change if you approached the task from Mr. Y's perspective?

Assume that Mr. Y uses the three XML documents mentioned, or your XML document, for order processing. Having received XML documents with different element names and hierarchical structures, Mr. Y would have to open each XML document in an editor, confirm whether all of the required information was present, and then process the purchase order. In this case, every purchase order would have to be processed by hand, and the entire system could never be automated.

What would happen if all XML documents sent had the same element names and hierarchical structure? With standard element names and structures, a system could be created to handle all incoming XML documents, and order processing could be automated, without Mr. Y having to verify the content of each individual document.

A "Schema" is what is required to allow the acceptance (or creation) of XML documents with a standardized element name and hierarchy structure.

Let's take another look at the task for creating an XML document to be used for a purchase order. Assume that Mr. Y sends to Ms. A, Mr. B, and Ms. C a schema document for purchase orders (XML document). Ms. A, Mr. B, and Ms. C then each create an XML document based on Mr. Y's schema. The element names and hierarchical structure of the XML documents, they send to Mr. Y, are completely identical.

Mr. Y can now use an XML parser to verify whether the documents have been created according to the schema, so there is no need to open each file and check element names and hierarchy structures. This reduces Mr. Y's workload significantly.

Thus, the use of schema for creating XML documents ensures that the hierarchical structure of the document is based on the defined schema.

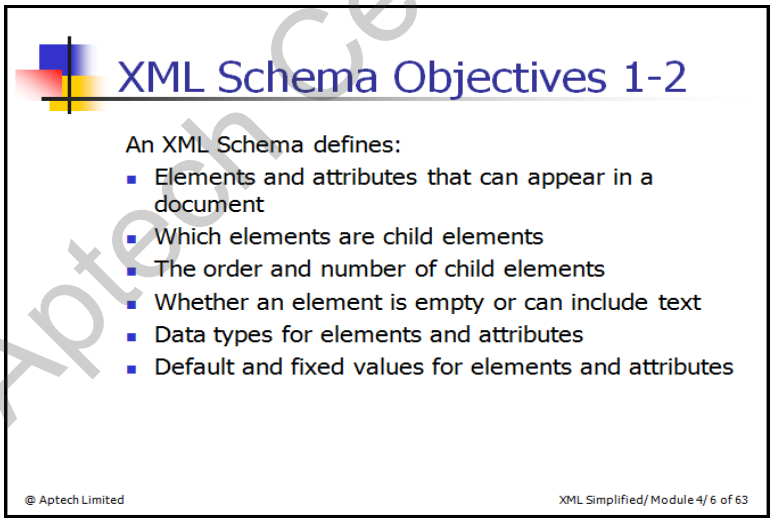
The word ‘schema’ originated from a Greek word symbolizing form or shape. The dictionary meaning of schema is: "A diagrammatic representation; an outline or a model". Initially, the word ‘schema’ was only in the reach of the philosophers, till it entered the zone of computer science. In the context of software, a schema is generally understood to be a model used to describe the structure of a database. It defines internal structures such as tables, fields, and the relationship between them.

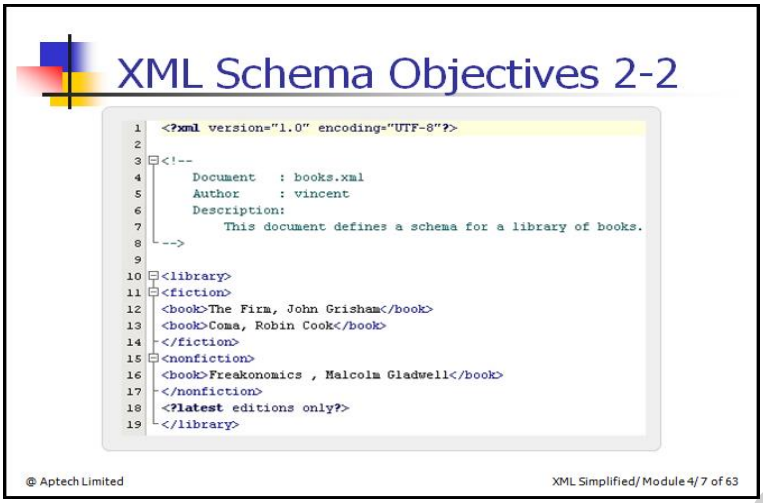
An XML Schema defines the valid building blocks of an XML document. It can be considered as a common vocabulary that different organizations can share to exchange documents. The XML Schema language is referred as XML Schema Definition (XSD).

Explain the figure on slide 4 which depicts the XML data validation. The previous knowledge of what a book is, what its attributes are, could be a kind of schema, and against which this 'BOOK' is compared would help to validate the attributes of the book.

Slides 6 and 7

Let us understand the XML schema objectives.





Using slides 6 and 7, explain the XML schema objectives.

The objective of an XML Schema is to define the valid structure of an XML document, similar to a DTD. However, XML schemas are more powerful than DTD.

Explain the features of XML schema.

Additionally, you can mention that XML schemas will be used in most Web applications as a replacement for DTD, as they are richer in data types and provide more support for XML document structure. Also, XML schemas are written in XML.

Explain the figure on slide 7 that depicts the XML document based on schema of books.

Tips:

Some of the advantages of using XML schema are as follows:

- XML Schemas are written in XML.
- XML Schemas are extensible to additions.
- XML Schemas support data types.
- XML Schemas support namespaces.

In-Class Question:

After you finish explaining the XML schema, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

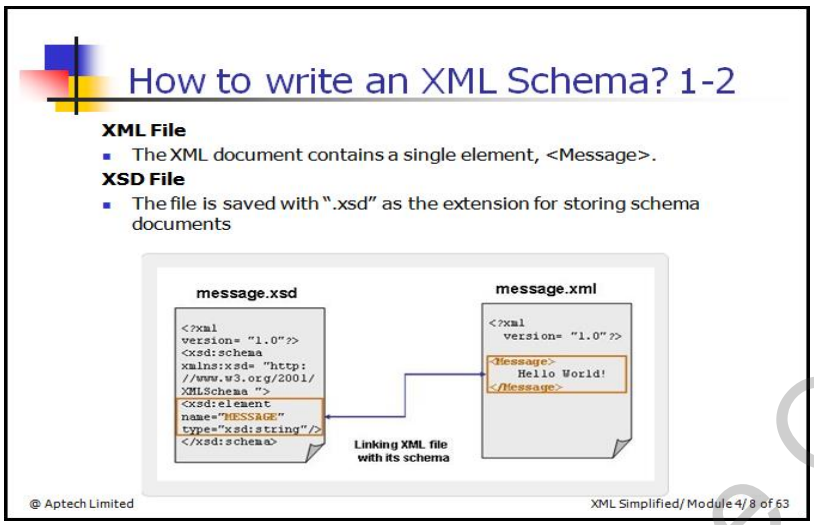
How is XML schema different from DTD?

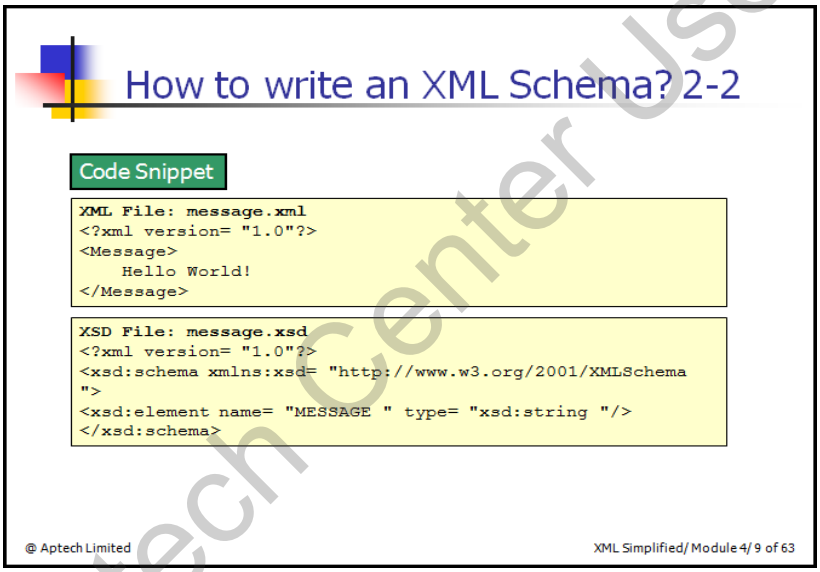
Answer:

XML schema is written in XML, whereas DTD is written in plain text file.

Slides 8 and 9

Let us understand how to write an XML schema.





Using slides 8 and 9, explain how to write XML schema.

To create the schema for XML document, you need two files, they are as follows:

- **XML File**
  Given XML document contains a single element, `<Message>`. A schema for this document has to declare the `<Message>` element.

- **XSD File**
  For storing schema documents, the file is saved with `.xsd` as the extension. Schema documents are XML documents and can have DTDs, DOCTYPE declarations.

Explain the figure on slide 8 that depicts XSD file.

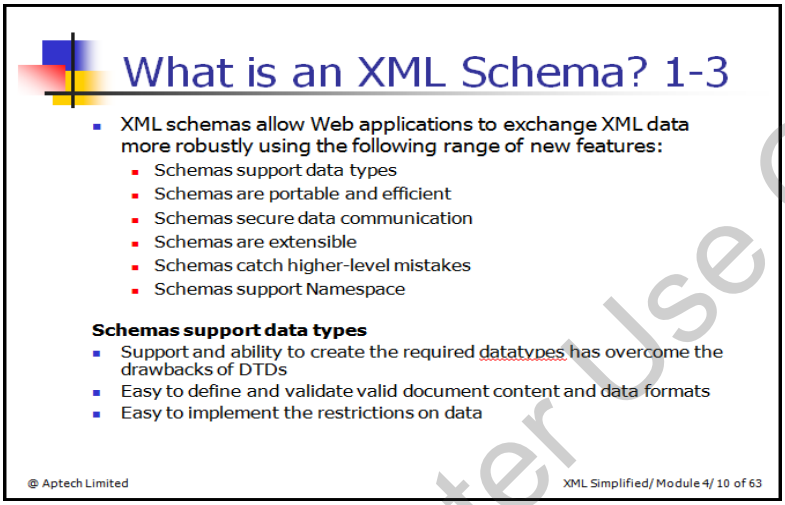
Explain the code on slide 9 which demonstrates the properties of XML schema.

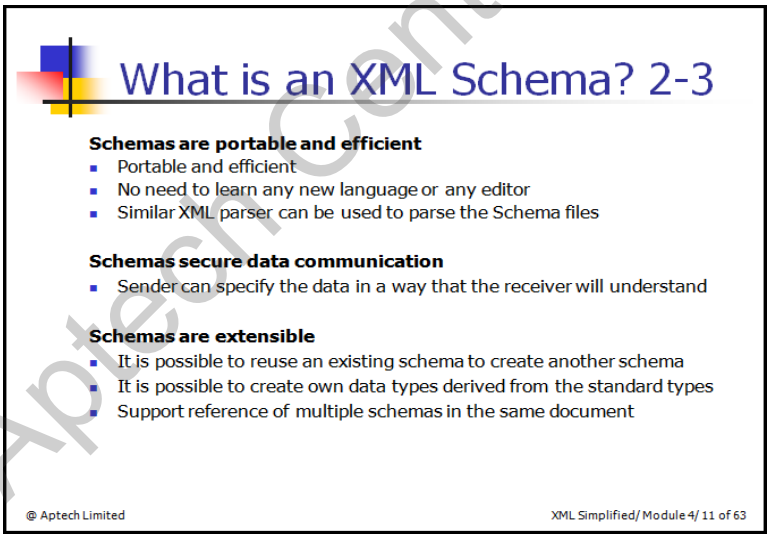
The code explanation is as follows:

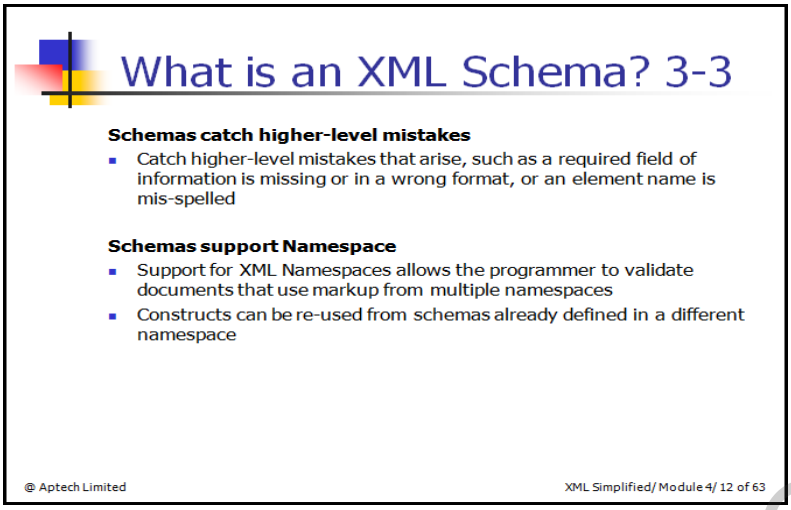
XSD file is a schema for `<Message>` elements. The file is stored as `message.xsd`. It is an XML document; therefore it has an XML declaration. This schema file may declare other elements too, including ones that are not present in this XML file, but it must at least declare the `<Message>` element.

Slides 10 to 12

Let us understand the features of XML schema.



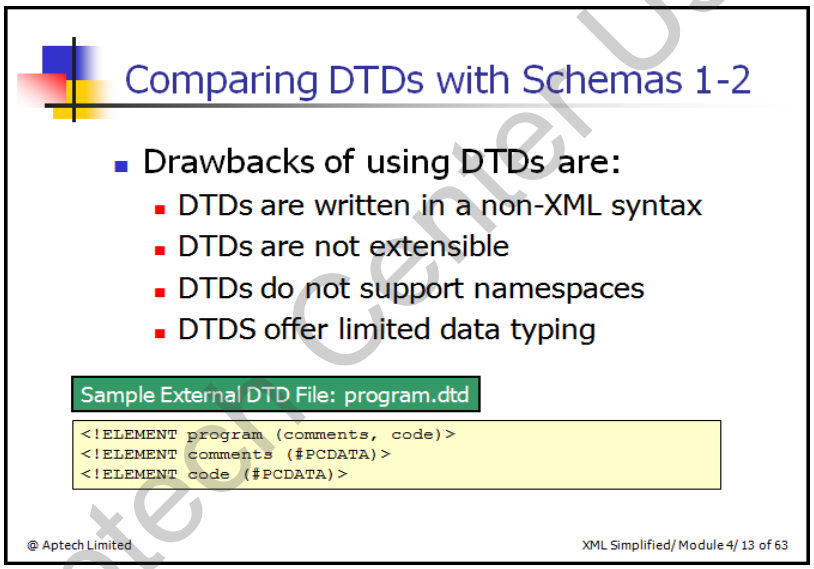


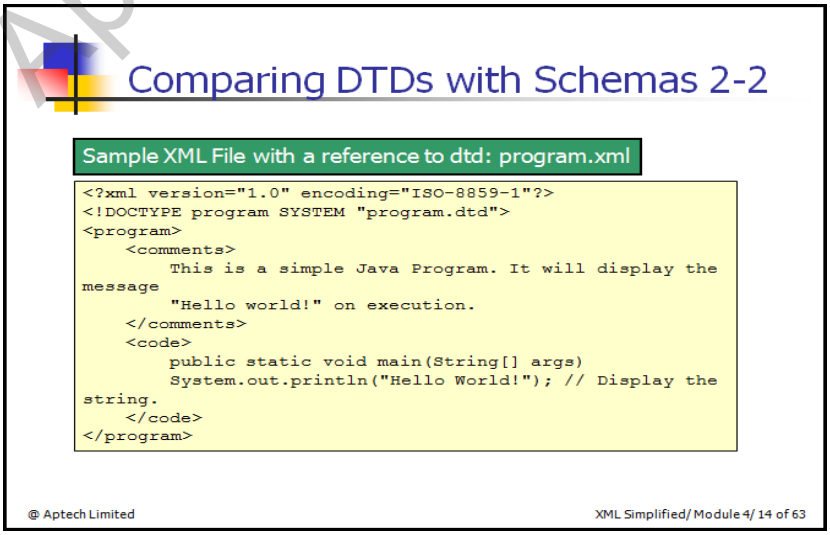


Using slides 10 to 12, explain the features of XML schema.

Slides 13 and 14

Let us understand the difference in DTDs and schemas.





Using slide 13, explain the difference in DTDs and schemas.

XML inherited the concept of DTDs from Standard Generalized Markup Language (SGML), which is an international standard for markup languages. DTDs are used to define content models, nesting of elements in a valid order, and provide limited support to data types and attributes.

Explain the drawbacks of using DTDs.

- **DTDs are written in a non-XML syntax**

  DTDs do not use XML notation and therefore they are difficult to write and use.

- **DTDs are not extensible**

  DTDs are not extensible. For example, if there is an Address DTD to catalog friends, and one wants to add a new section to the code for official contacts, then the entire DTD has to be rewritten.

- **DTDs do not support namespaces**

  Namespaces can be used to introduce an element type into an XML document. However, a Namespace cannot be used to refer to an element or an entity declaration in the DTD. If a Namespace is used, then the DTD has to be modified to include any elements taken from the Namespace.

- **DTDS offer limited data typing**

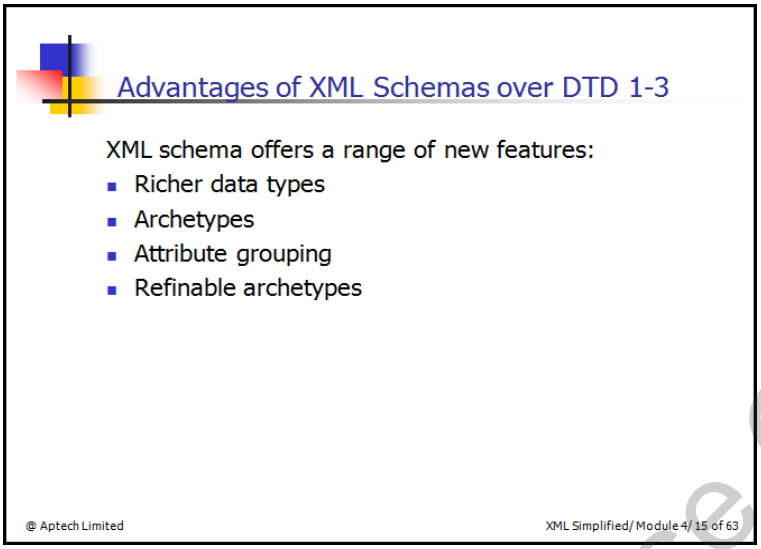
  DTDs can only express the data type of attributes in terms of explicit enumerations and a few coarse string formats. DTDs do not have a facility to describe numbers, dates, currency values, and so forth. Furthermore, DTDs do not have the ability to express the data type of character data in elements. For example, a <Zip> element can be defined to contain CDATA. However, the element cannot be constrained to just numerals when it uses a DTD.

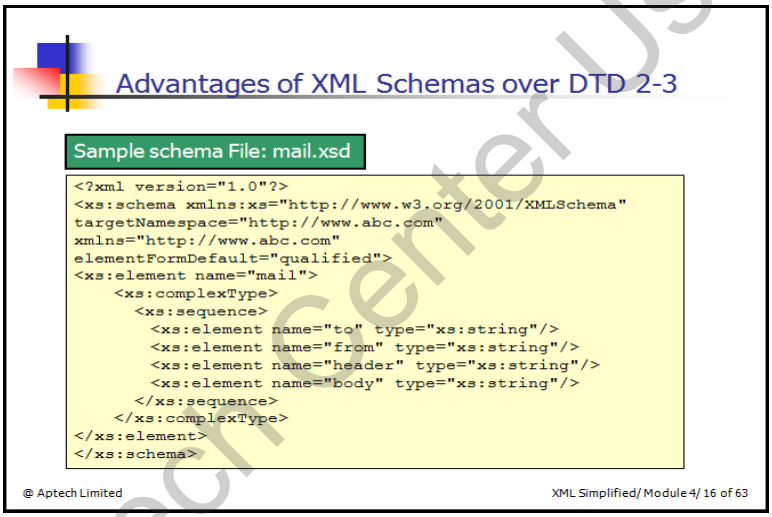
Explain the code on slide 13 which demonstrates a sample external DTD file, `program.dtd`.

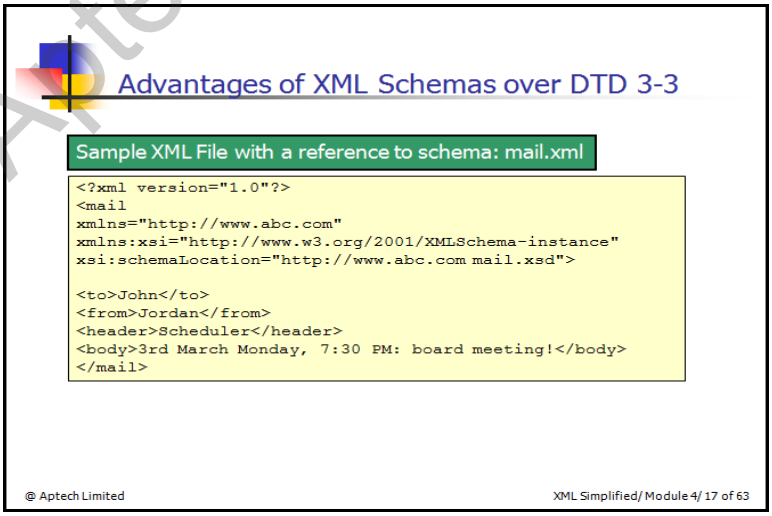
Explain the code on slide 14 which demonstrates a sample XML file with a reference to DTD program.xml.

Slides 15 to 17

Let us understand the advantages of XML schemas over DTD.







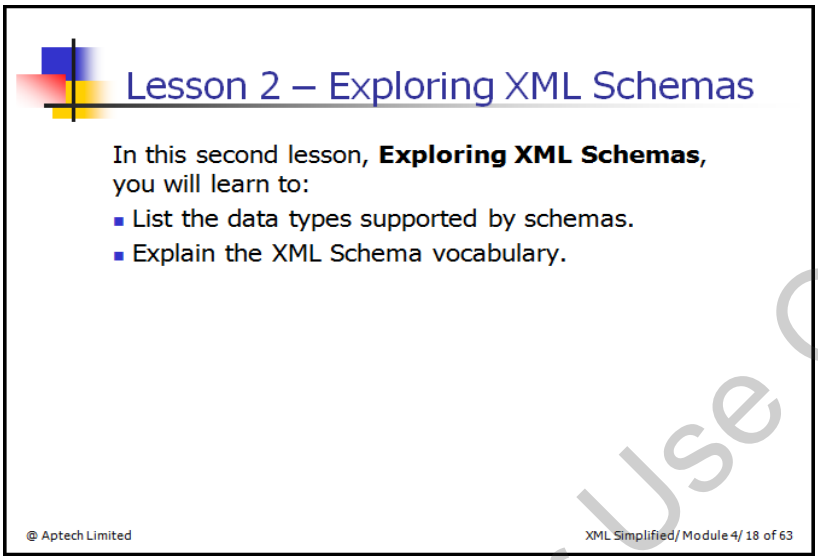
Using slides 15 to 17, explain the advantages of XML schemas over DTD.

Explain the code on slide 16 which demonstrates a sample schema file, `mail.xsd`.

Explain the code on slide 17 which demonstrates a sample XML file with a reference to schema `mail.xml`.

Slide 18

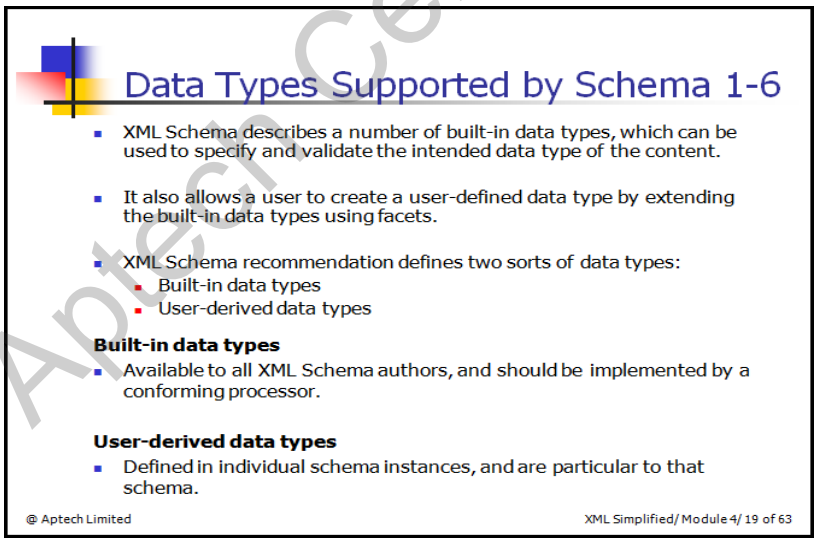
Let us understand the objectives of lesson 2.

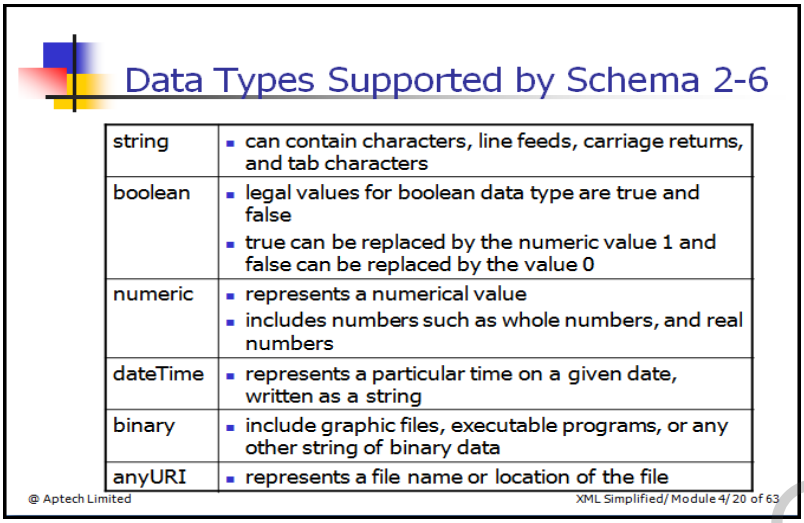


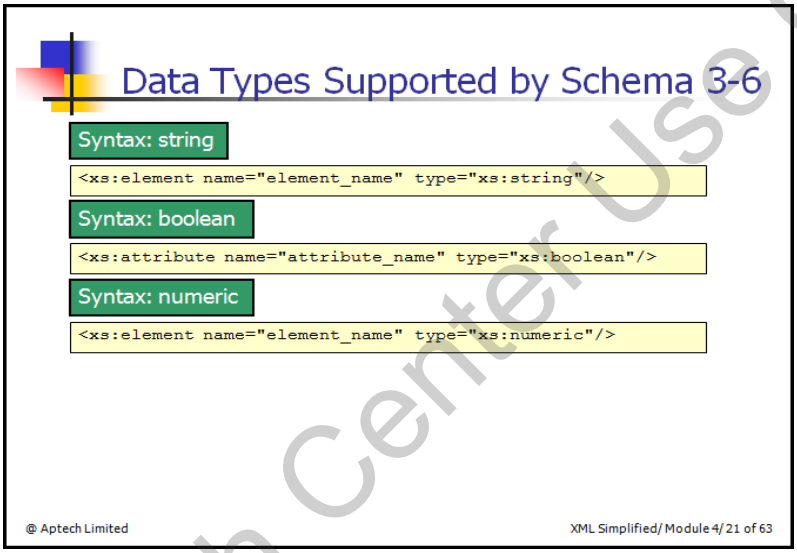
Using slide 18, explain the topics covered in the lesson 2.

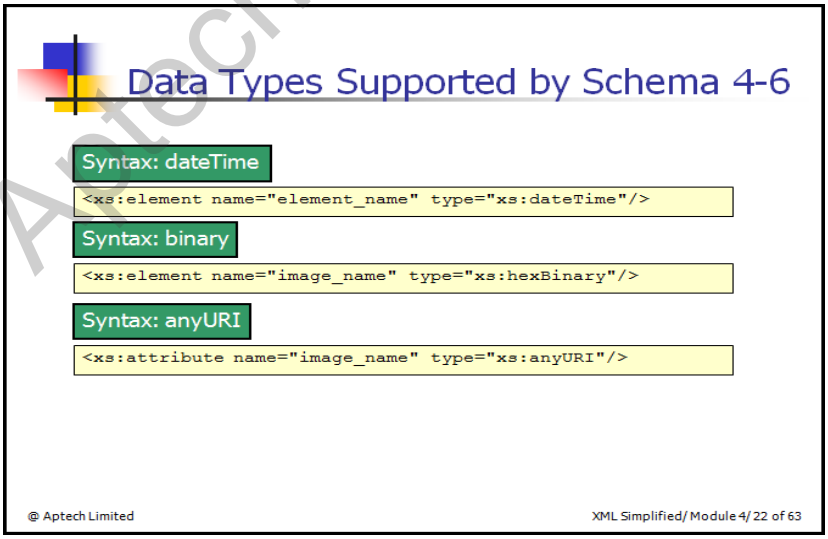
Slides 19 to 24

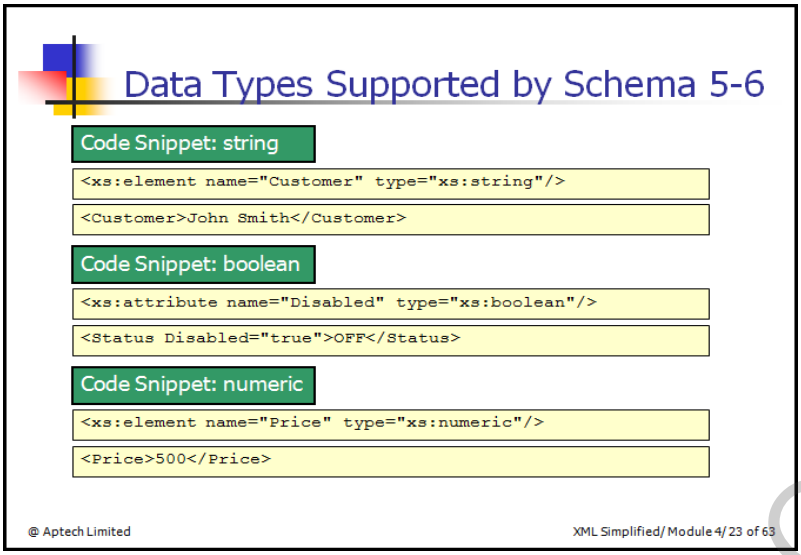
Let us understand the data types supported by schemas.

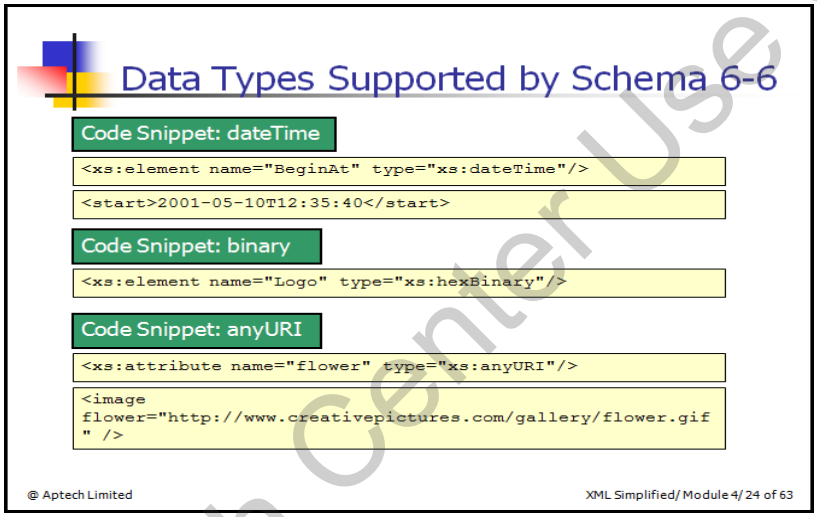












Using slide 19, explain the data types supported by schemas.

XML Schema describes a number of built-in data types, which can be used to specify and validate the intended data type of the content. It also allows a user to create a user-defined data type by extending the built-in data types using facets.

Explain two types of data types in XML schemas.

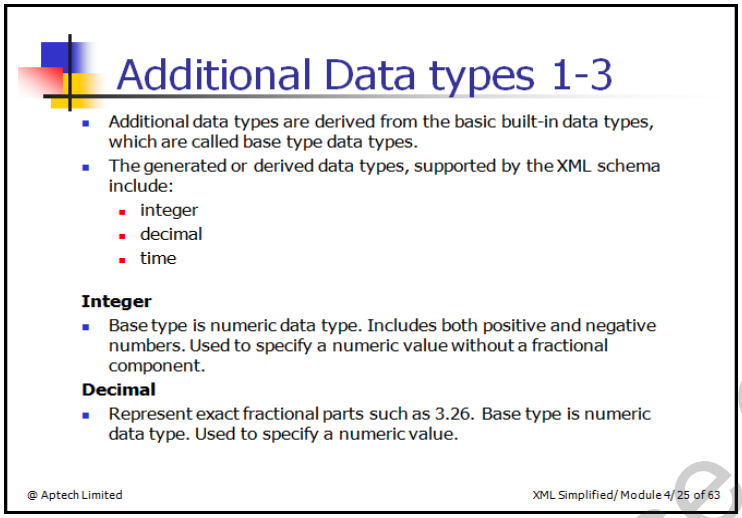
- **Built-in** data types, are available to all XML Schema authors, and should be implemented by a conforming processor.
- **User-derived** data types, are defined in individual schema instances, and are particular to that schema (although, it is possible to import these definitions into other definitions).

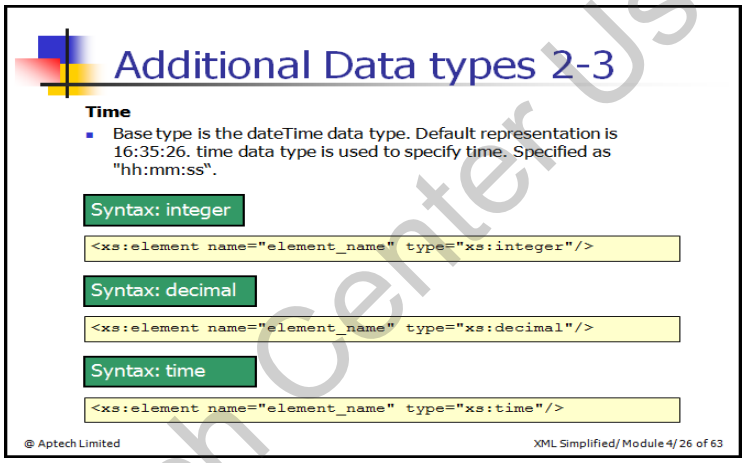
Using slide 20, explain the types of built-in data types mentioned in the table. Also, using slides 21 and 22, explain the syntax of data types.

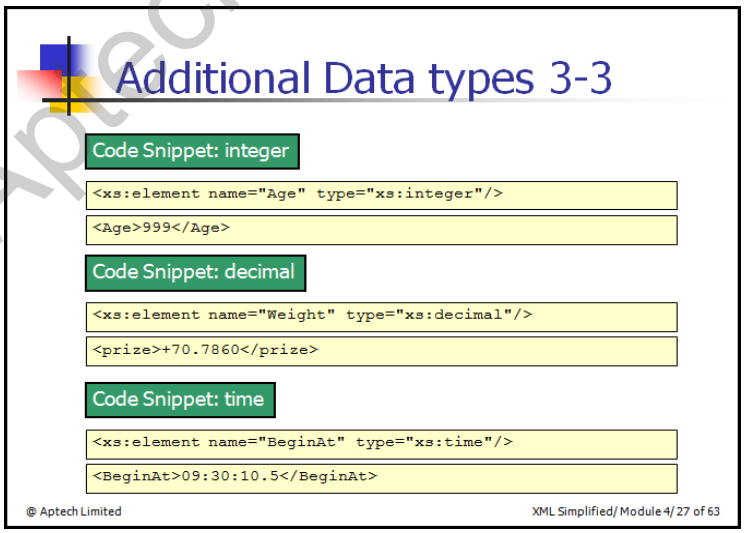
Using slides 23 and 24, explain the code snippets which demonstrates the use of data types.

Slides 25 to 27

Let us understand the additional data types.







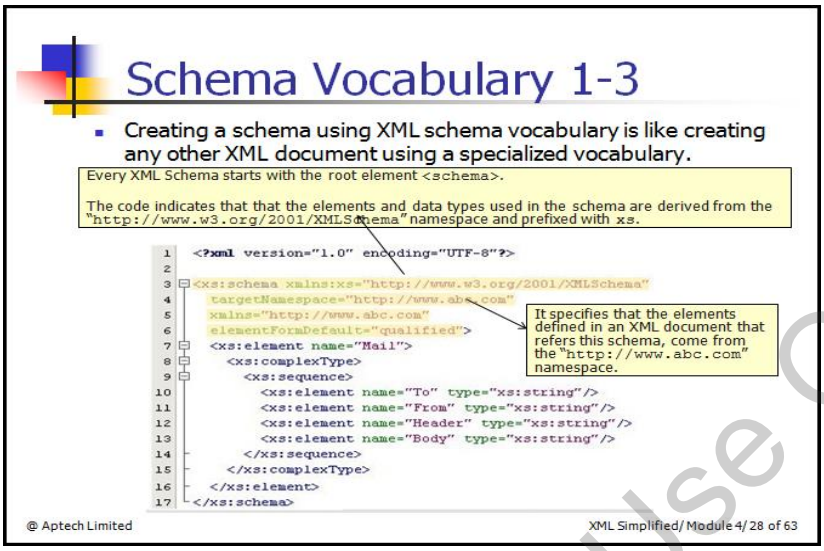
Using slides 25 to 27, explain the additional data types.

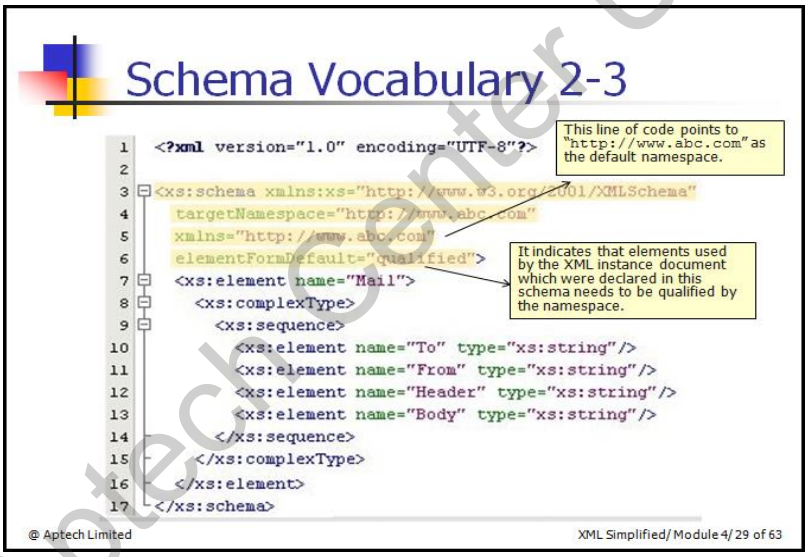
Additional data types are derived from the basic built-in data types, which are called base type data types. Explain the generated or derived data types supported by the XML schema.

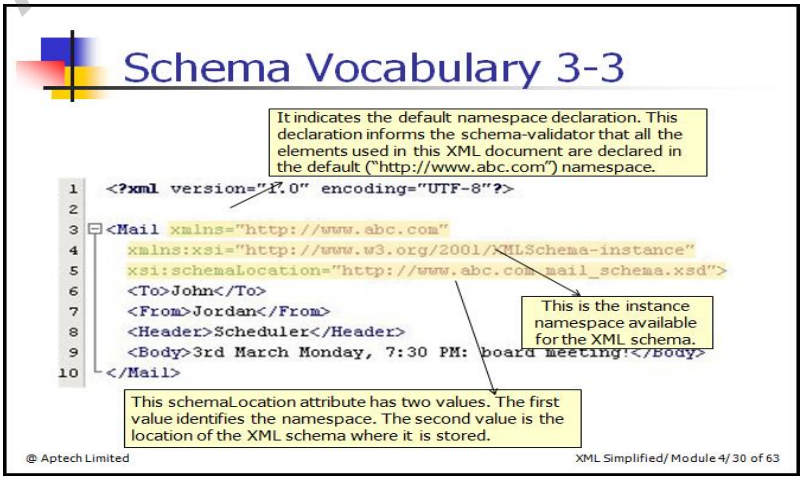
Using slides 26 and 27, explain the code and syntax for derived data types.

Slides 28 to 30

Let us understand the schema vocabulary.







Using slides 28 to 30, explain the schema vocabulary.

Creating a schema using XML schema vocabulary is like creating any other XML document using a specialized vocabulary. To understand the XML Schema vocabulary and the elements, the example discusses an XML document that will be validated against a schema.

Explanation of the different parts is as follows:

- **`xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema`**
  Every XML Schema starts with the root element `<schema>`. The code indicates that the elements and data types used in the schema are derived from the "`http://www.w3.org/2001/XMLSchema`" namespace and prefixed with `xs`.

- **`targetNamespace=http://www.abc.com`**
  It specifies that the elements defined in an XML document that refers this schema, come from the "`http://www.abc.com`" namespace.

- **`xmlns=http://www.abc.com`**
  This line of code points to "`http://www.abc.com`" as the default namespace.

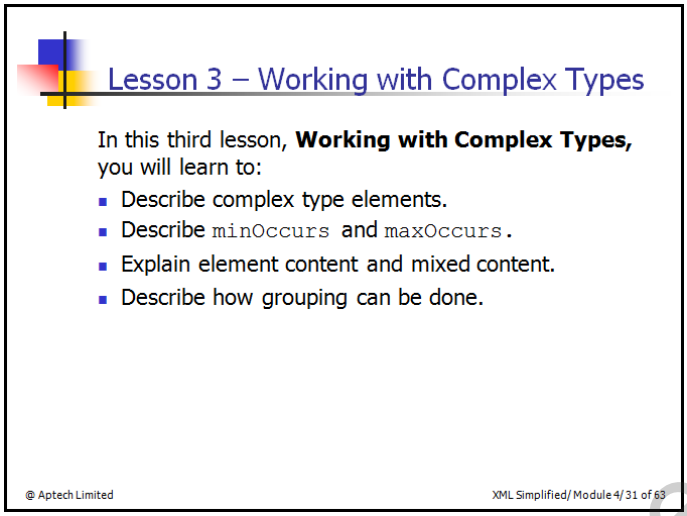
- **`elementFormDefault="qualified"`**
  It indicates that elements used by the XML instance document which were declared in this schema needs to be qualified by the namespace.

- **`xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance`**
  This is the instance namespace available for the XML schema.

- **`xsi:schemaLocation="http://www.abc.com mail.xsd">`**
  This `schemaLocation` attribute has two values. The first value identifies the namespace. The second value is the location of the XML schema where it is stored.

Slide 31

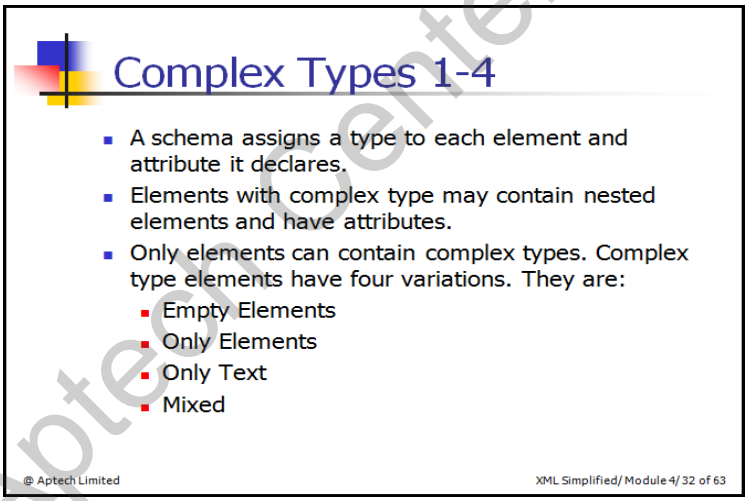
Let us understand the objectives of lesson 3.

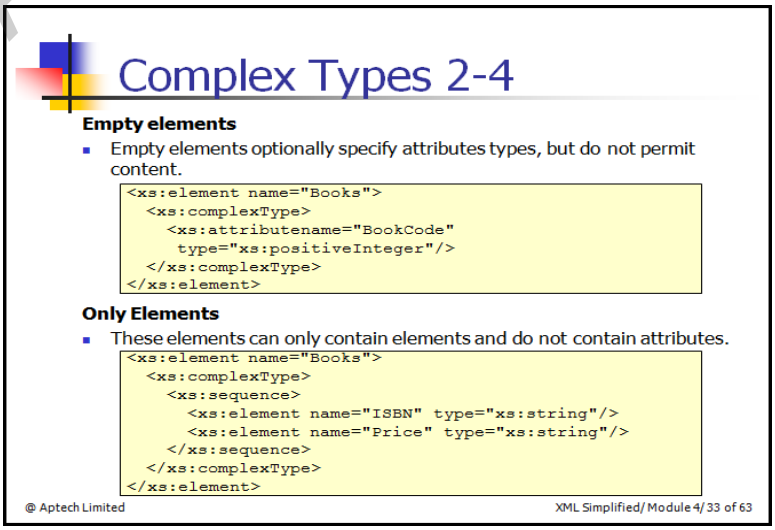


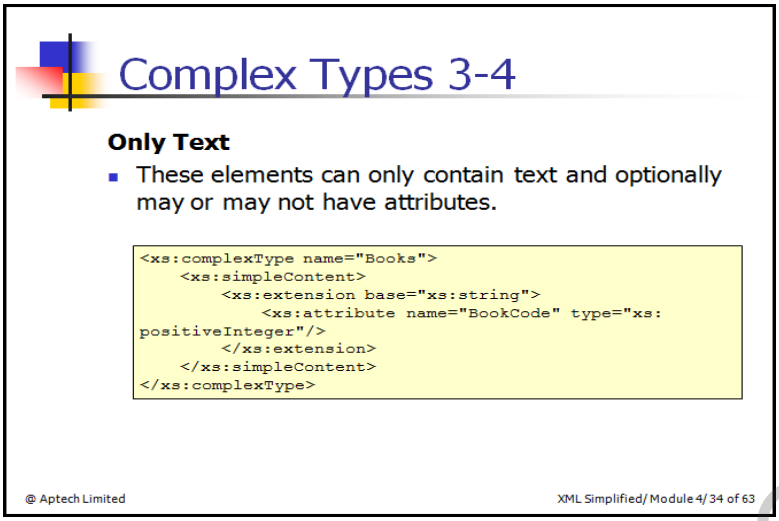
Using slide 31, explain the topics covered in the lesson 3.

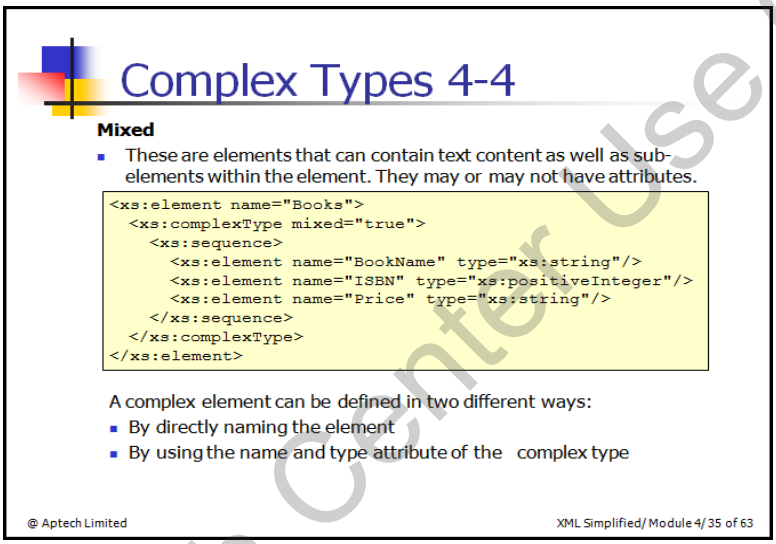
Slides 32 to 35

Let us understand the complex types.









Using slides 32 to 35, explain the complex types.

A complex element is an XML element that contains other elements and/or attributes. A schema assigns a type to each element and attribute that it declares. Only elements can contain complex types.

There are four kinds of complex elements:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

Explain the empty complex type elements in detail.

Empty elements

Empty elements optionally specify attributes types, but do not permit content as shown in the following example:

```
<xs:element name="Books">
<xs:complexType>
<xs:attributename="BookCode"
type="xs:positiveInteger"/>
</xs:complexType>
</xs:element>
```

Only Elements

These elements can only contain elements and do not contain attributes as shown in the following example:

```
<xs:element name="Books">
<xs:complexType>
<xs:sequence>
<xs:element name="ISBN" type="xs:string"/>
<xs:element name="Price" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Only Text

These elements can only contain text and optionally may or may not have attributes as shown in the following example:

```
<xs:complexType name="Books">
<xs:simpleContent>
<xs:extension base="xs:string">
<xs:attribute name="BookCode" type="xs: positiveInteger"/>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
```

Mixed

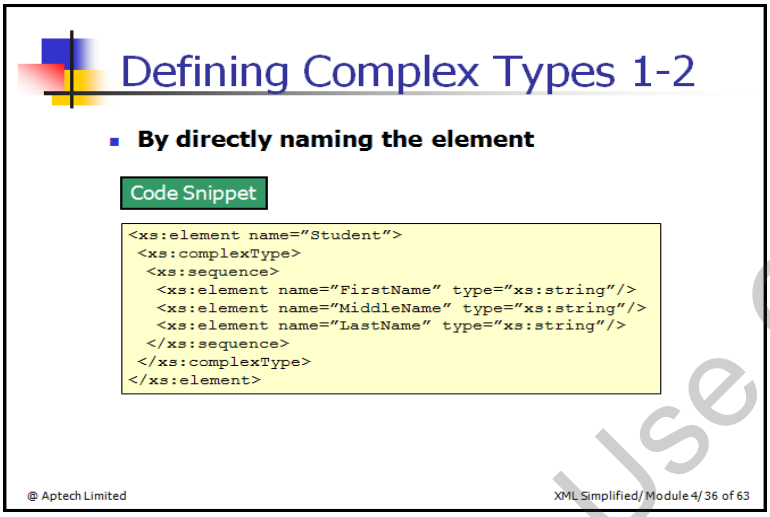
These are elements that can contain text content as well as sub-elements within the element as shown in the following example:

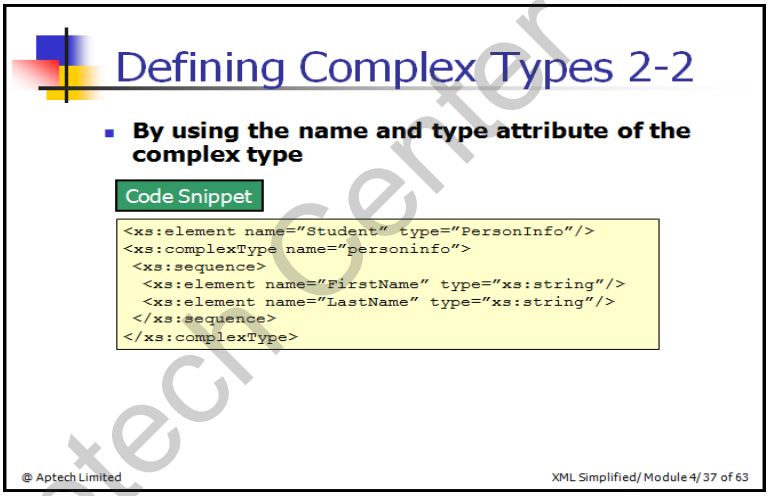
```
<xs:element name="Books">
<xs:complexType mixed="true">
<xs:sequence>
<xs:element name="BookName" type="xs:string"/>
<xs:element name="ISBN" type="xs:positiveInteger"/>
<xs:element name="Price" type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

They may or may not have attributes.

Slides 36 and 37

Let us understand how to define complex types.





Using slides 36 and 37, explain how to define complex types.

A complex element can be defined in two different ways:

By directly naming the element

Explain the code on slide 36 which defines the properties of complex element.

The element `Student` can be declared by directly mentioning it in the schema. The complex type mentions that the XML document contains nested XML elements. The sequence element indicates that the child elements `FirstName`, `MiddleName`, and `LastName` appear in the same order as they are declared.

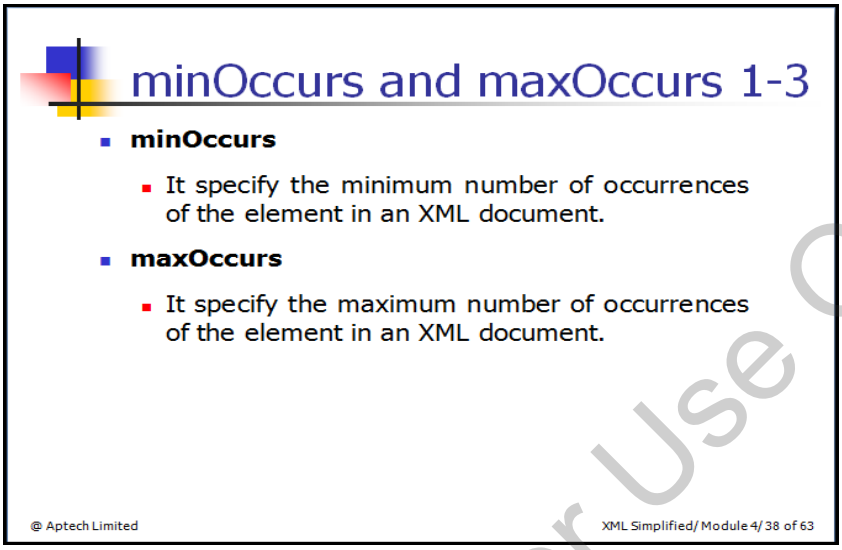
By using the name and type attribute of the complex type

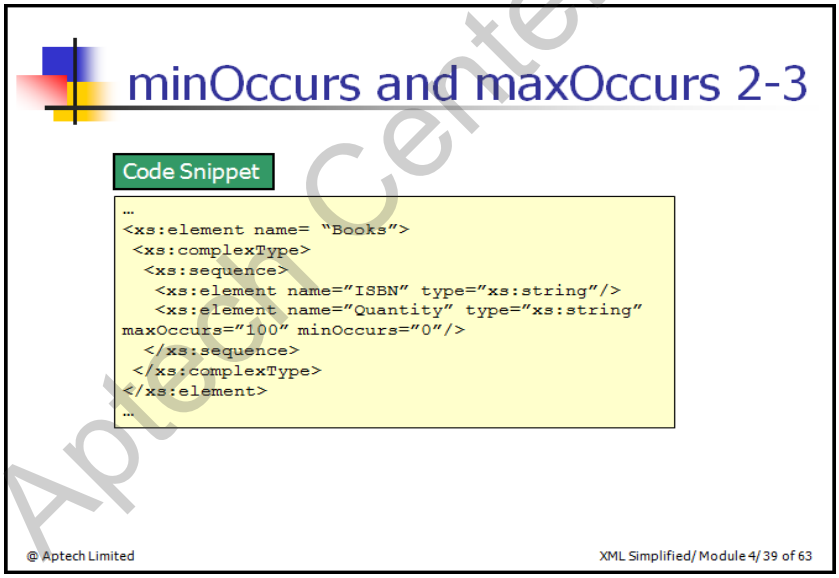
Explain the code on slide 37 which define the properties of complex element by using the name and type attribute of the complex type.

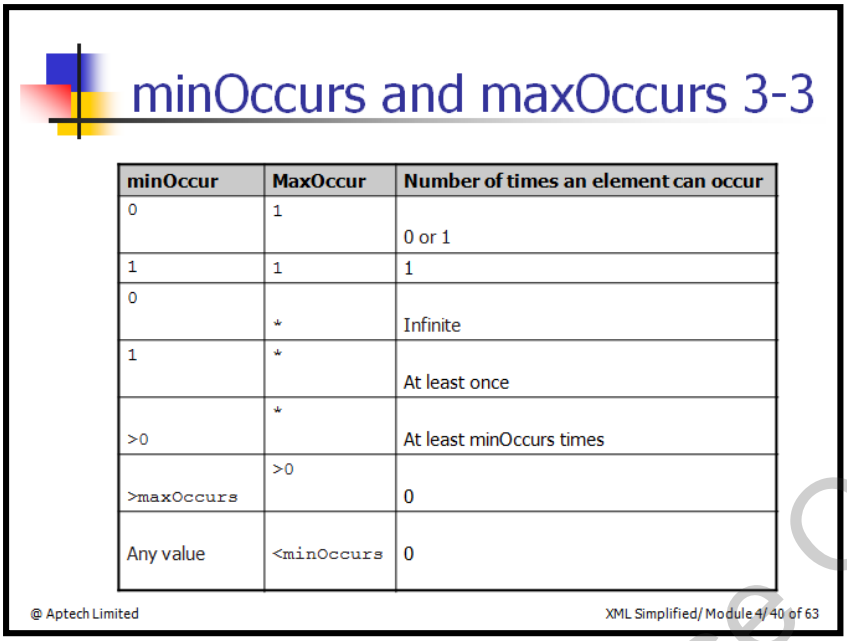
The `Student` element can have a type such as `PersonInfo` that refers to the name of the complex type. Several elements can reuse this complex type by referring to the name `PersonInfo` in their complex type declarations.

Slides 38 to 40

Let us understand the `minOccurs` and `maxOccurs`.







Using slides 38 to 40, explain the `minOccurs` and `maxOccurs`.

When working with DTDs, we used the markers `*`, `?`, and `+` to indicate the number of times a particular child element could be used as content for an element. Similarly, XML schema allows specifying a minimum and maximum number of times an element can occur.

`minOccurs` specify the minimum number of occurrences of the element in an XML document. The default value for the `minOccurs` attribute is 1. If an element has a `minOccurs` value of 0, it is optional. An attribute is optional, if the `minOccurs` is 0. If `minOccurs` is set to 1, the attribute is required.

`maxOccurs` specify the maximum number of occurrences of the element in an XML document. The default value for the `maxOccurs` attribute is 1. If its value is kept unbounded, it means that the element can appear unlimited number of times. The `maxOccurs` attribute defaults to 1 unless it is specified.

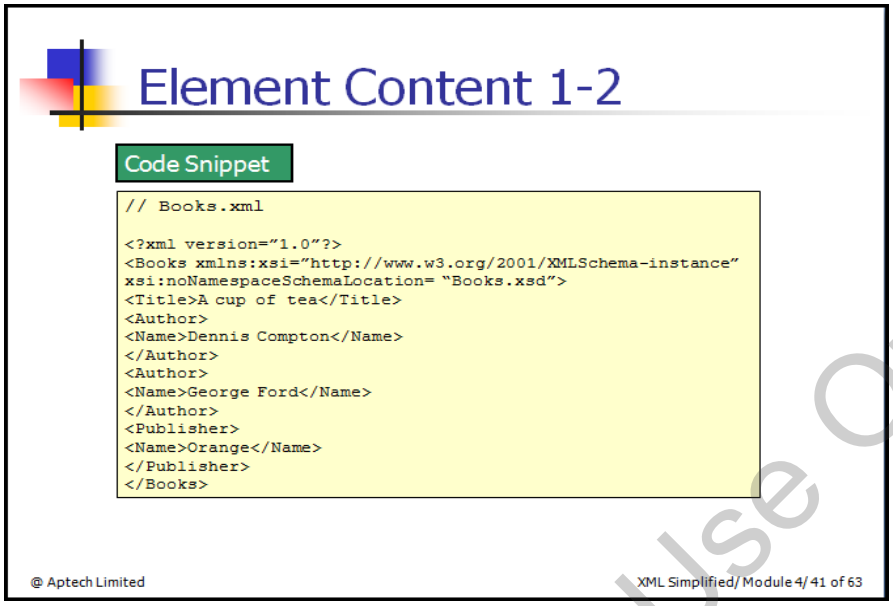
Explain the code on slide 39 which demonstrates the use of `minOccurs` and `maxOccurs` attributes.

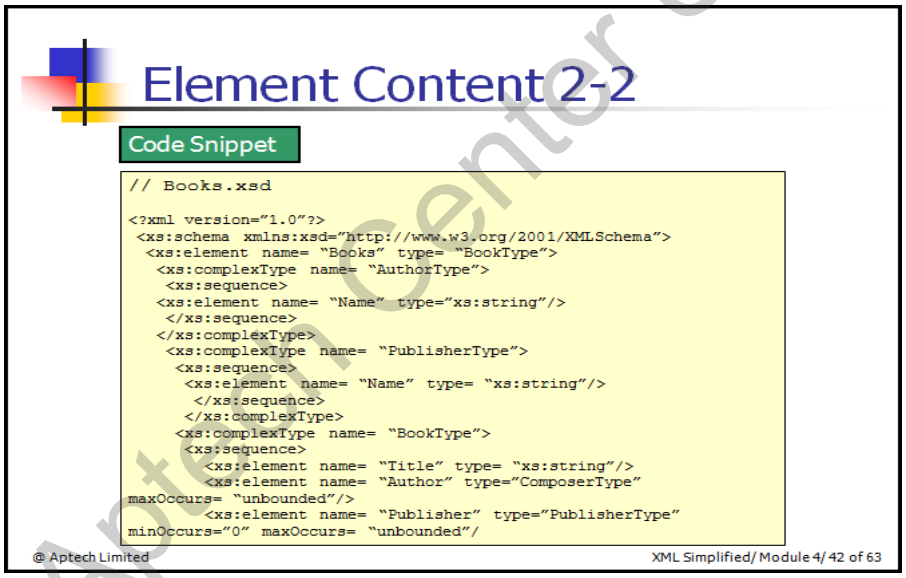
The example demonstrates that the `Quantity` element can occur a minimum of zero times and a maximum of hundred times in the `Books` element.

The relationship between the `minOccurs` and `maxOccurs` attributes is displayed in table which is shown on slide 40.

Slides 41 and 42

Let us understand the element content.





Using slide 41, explain the element content.

A complex type can mention element content. The `Element` content can contain only elements. There can be instances wherein the contained elements can also have child elements.

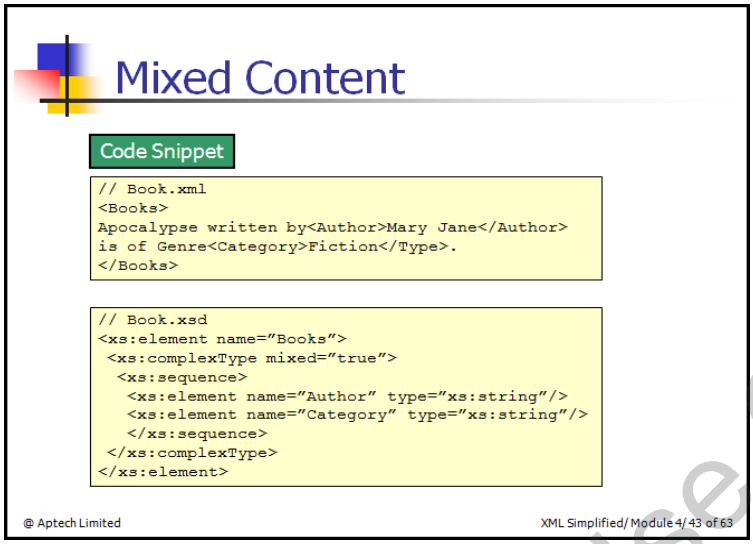
Explain the code on slide 42 which demonstrates the properties of element content.

This is an XML schema constrained document depicting book details.

In the XML document, the `Author` and the `Publisher` elements each contain `Name` elements. So built-in data types like `xs:string` cannot be used, instead `AuthorName` and `PublisherName` can be defined using top level `xs:complexType` elements.

Slide 43

Let us understand the mixed content.



Using slide 43, explain the mixed content.

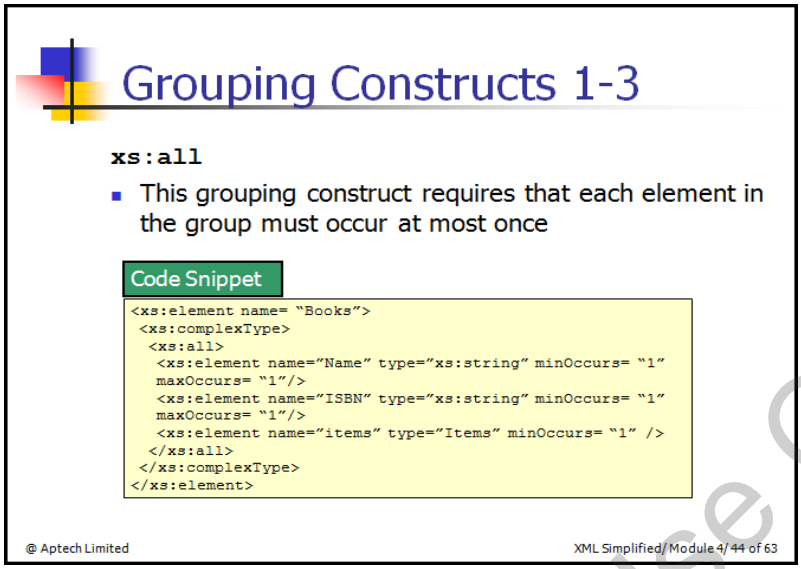
A complex type can specify its content as a mixed content. Mixed content can contain text mixed with elements. The order and the number of elements that appears in the mixed content can also be specified in the schema. This type of content may or may not have attributes.

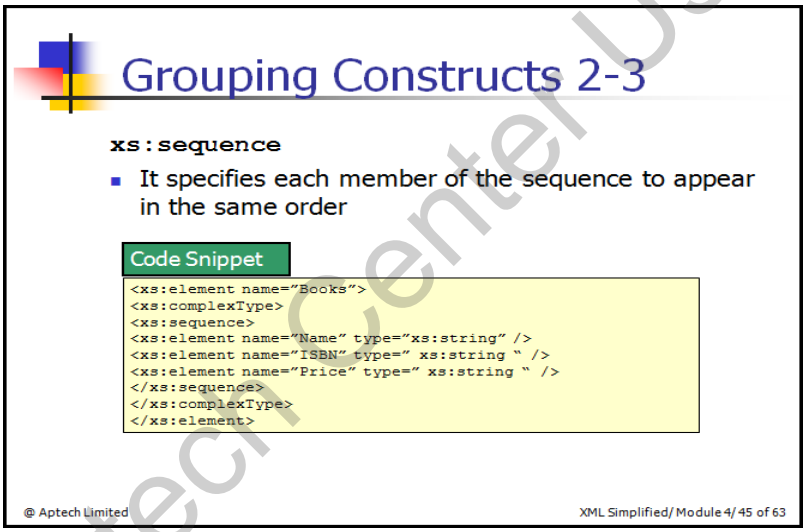
The mixed content is declared in exactly the same way, the declaration of element content is done. The only add-on to the declaration is, it takes a mixed attribute set to the true value.

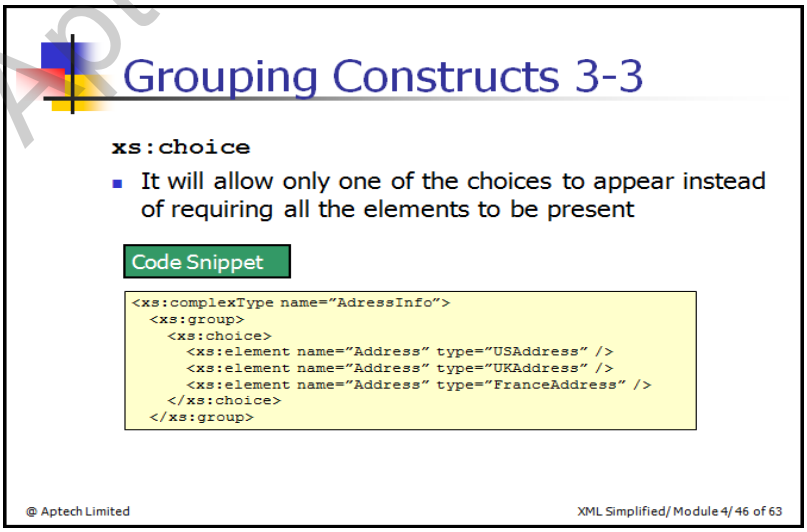
Explain the code on slide 43 which demonstrates the properties of mixed content.

Slides 44 to 46

Let us understand how to group constructs.







Using slides 44 to 46, explain how to group constructs.

The XSL provides three grouping constructs that specify whether and how ordering of individual elements is important.

Explain `xs:all` - Grouping construct requires that each element in the group must occur at most once, but that order is not important. The only caution is that in this type of grouping the `minOccurs` attribute can be 0 or 1 and the `maxOccurs` attribute has to be 1.

Explain the code on slide 44 which demonstrates this concept.

The choice element `xs:choice` is the opposite of all elements. Instead of requiring all the elements to be present, it will allow only one of the choices to appear. The choice element provides an XML representation for describing a selection from a set of element types. The choice element itself can have `minOccurs` and `maxOccurs` attributes that establish exactly how many selections may be made from the choice.

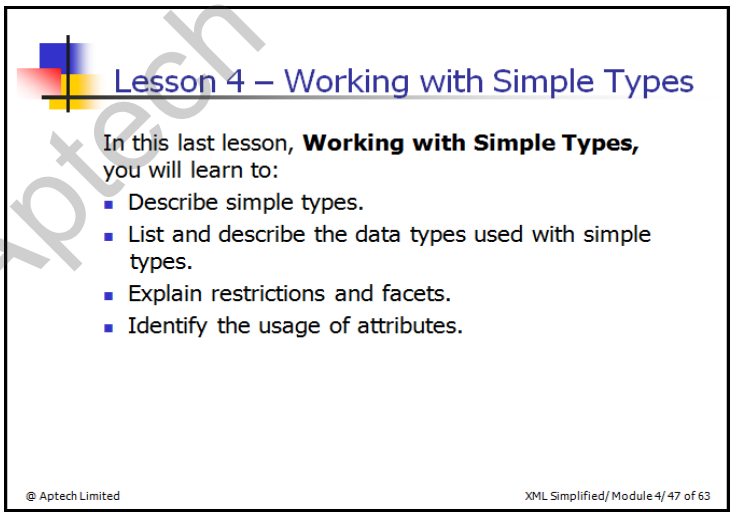
Explain the code on slide 45 which demonstrates this concept.

In the code snippet, there are three child elements that are mutually exclusive. With the `xs:choice` element declared, only one element among the choices can be a child element of the parent element `AddressInfo`.

An `xs:sequence` element specifies each member of the sequence to appear in the same order in the instance document as mentioned in the `xs:sequence` element. The number of times each element is allowed to appear can be controlled by the element's `minOccurs` and `maxOccurs` attributes. Explain the code on slide 46 which demonstrates this concept.

Slide 47

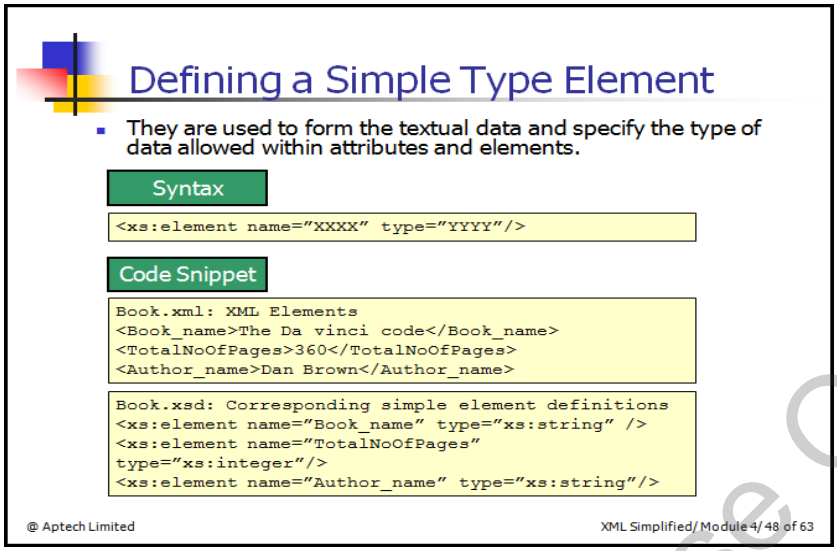
Let us understand the objectives of lesson 4.



Using slide 47, explain the topics to be covered in the lesson 4.

Slide 48

Let us understand how to define simple type element.



Using slide 48, explain how to define simple type element.

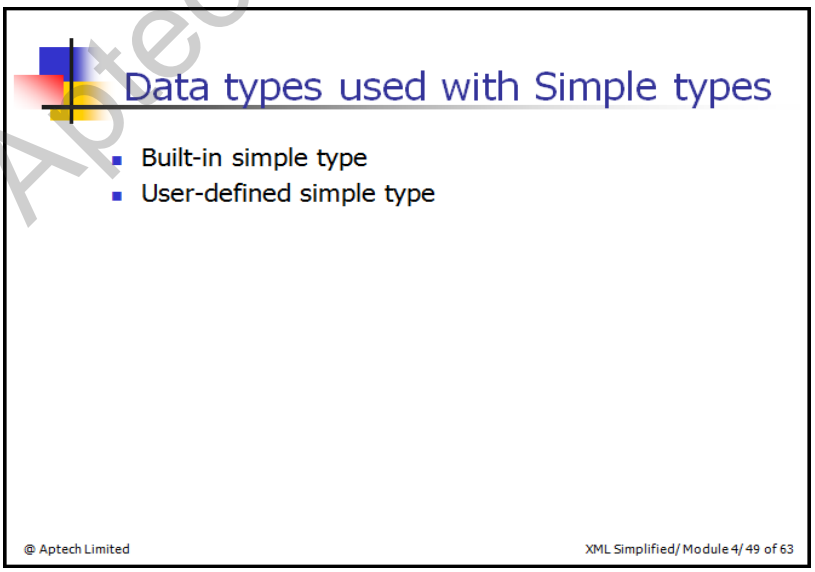
A simple type is an XML element or attribute, which contains only text and no other elements or attributes.

The simple type declarations are used to form the textual data and specify the type of data allowed within attributes and elements.

Explain the code on slide 48 demonstrates that the element `TotalNoOfPages` can be specified as an integer type, which is again a positive integer with three digits.

Slide 49

Let us understand the data types used with simple types.



Using slide 49, explain the data types used with simple types.

Elements of simple type tend to describe the content and the data type of a document, rather than its structure.

In XML Schema, one can mention the type of data an element can contain by assigning it a particular simple type definition. So, based upon the type of data it supports.

XML schema divides the elements of simple types into two broad categories:

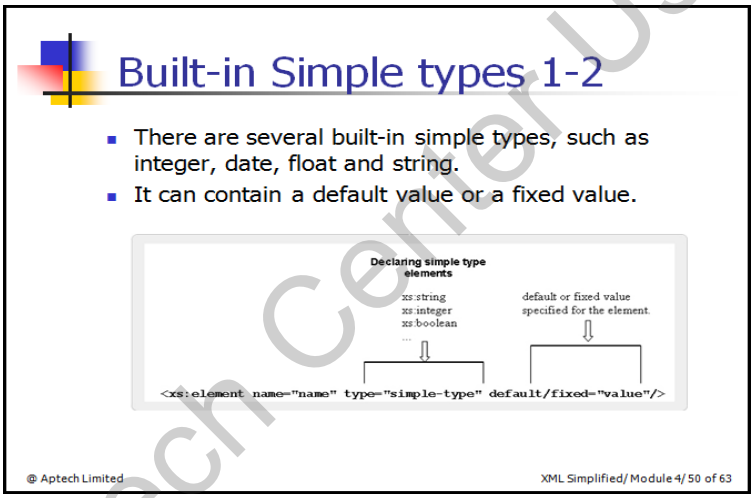
- built-in simple type
- user-defined simple type

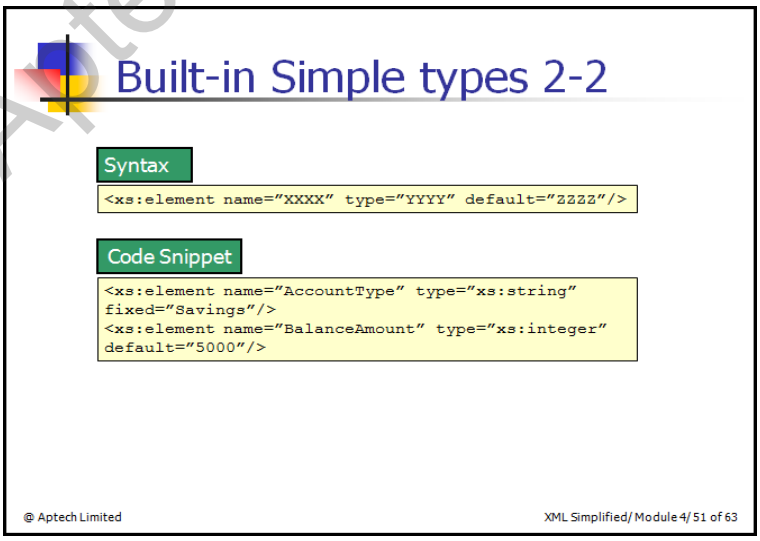
Tips:

You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Slides 50 and 51

Let us understand the built-in simple types.





Using slide 50, explain the built-in simple types.

There are several built-in simple types, such as integer, date, float, and string that one can use without further modifications.

A built-in simple element can contain a default value or a fixed value. A 'default' value is the value that is assigned automatically to the element when no other value has been specified. A 'fixed' value is assigned to an element, when there is no need to change the value for that element. Figure on slide 50 depicts built-in simple types.

Explain the syntax of built-in simple type using slide 51.

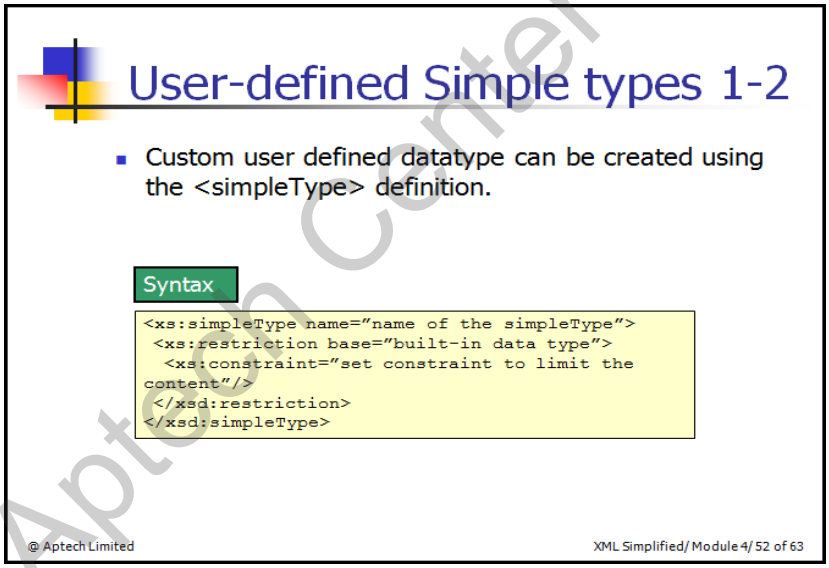
In this syntax, '`XXXX`' is the name of the element, '`YYYY`' is the data type of the element, and '`ZZZZ`' is the default value specified for the element.

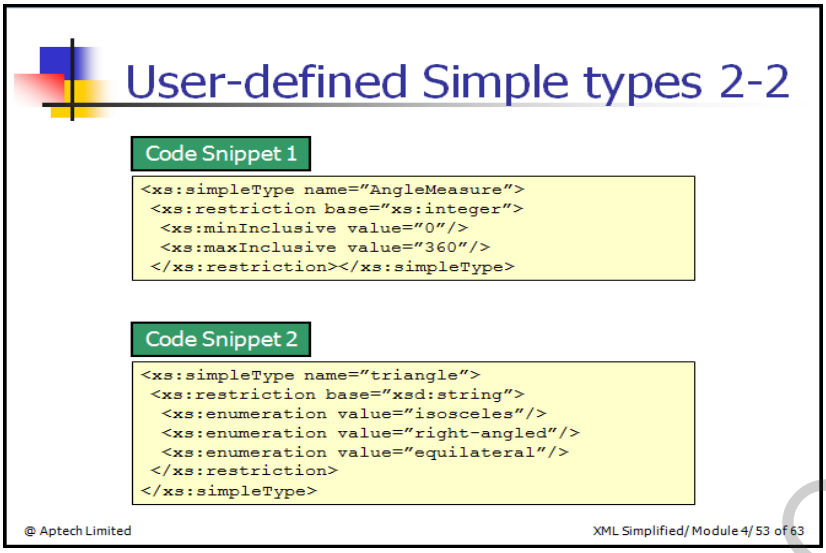
Explain the code on slide 51 which demonstrates the declaration of built-in simple type.

The code shows the declaration of built-in simple types "`AccountType`" and "`BalanceAmount`". The fixed value for the "`AccountType`" is "Savings" and the default value for "`BalanceAmount`" is "`5000`".

Slides 52 and 53

Let us understand the user-defined simple types.





Using slides 52 and 53, explain the user-defined simple types.

XML-schema supports a library of built-in data types. However, during the course of writing complex schema documents, a developer may need a type of data that is not defined in the schema recommendation. For instance, consider the declaration of an element `<AngleMeasure>`.

It implements the available restrictions to its content to accept non-negative numeric values, but still it accepts unwanted values like:

```
<AngleMeasure>490</AngleMeasure>
```

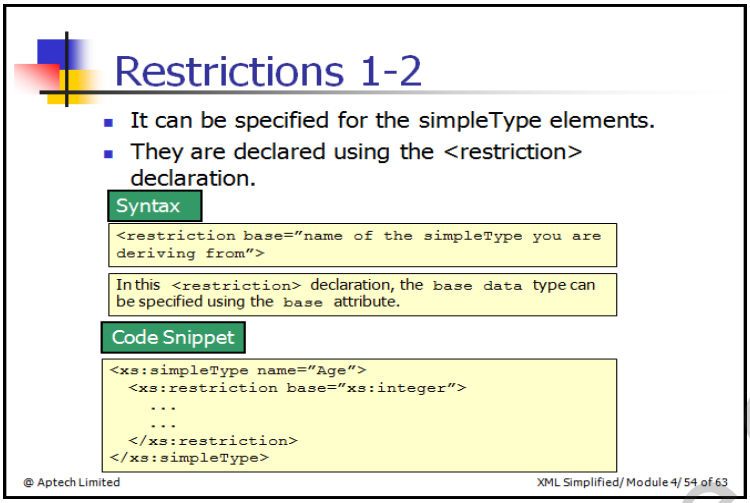
As per schema declaration, "`490`" is a valid non-negative numeric value, but still it needs numeric type that allows the values with a range from 0 to 360. Hence, a custom user defined data type can be created using the `<simpleType>` definition.

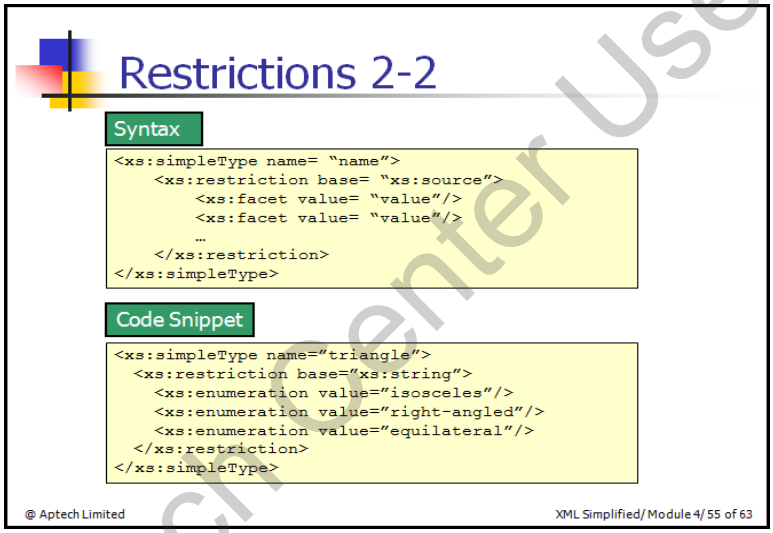
Explain the syntax of simple types.

Then, explain the codes on slide 53 which demonstrate the properties of user-defined simple types.

Slides 54 and 55

Let us understand the restrictions.





Using slide 54, explain the restrictions.

Declaration of a data type puts certain limitations on the content of an XML element or attribute. If an XML element is of type "`xs:integer`" and contains a string like "`Welcome`", the element will not be validated. These limitations are called restrictions, which defines allowable values for XML elements and attributes.

Restrictions can be specified for the `simpleType` elements and restriction types are declared using the `<restriction>` declaration. Basically, the `<restriction>` declaration is used to declare a derived `simpleType`, which is a subset of its base `simpleType`. The value of the base attribute can be any existing `simpleType`, or built-in XML Schema data type.

Explain the syntax of restrictions using slide 54.

In this `<restriction>` declaration, the base data type can be specified using the `base` attribute.

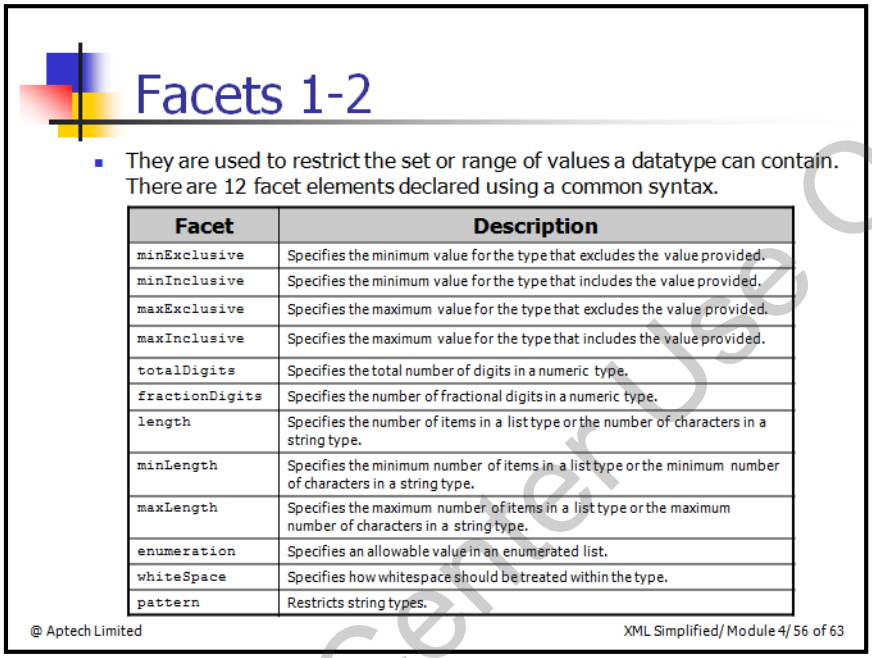
Explain the code on slide 55 which demonstrates a simpleType "`Age`" is derived by specifying the base as integer type.

Tips:

Restrictions on XML elements are called facets.

Slides 56 and 57

Let us understand the Facets.

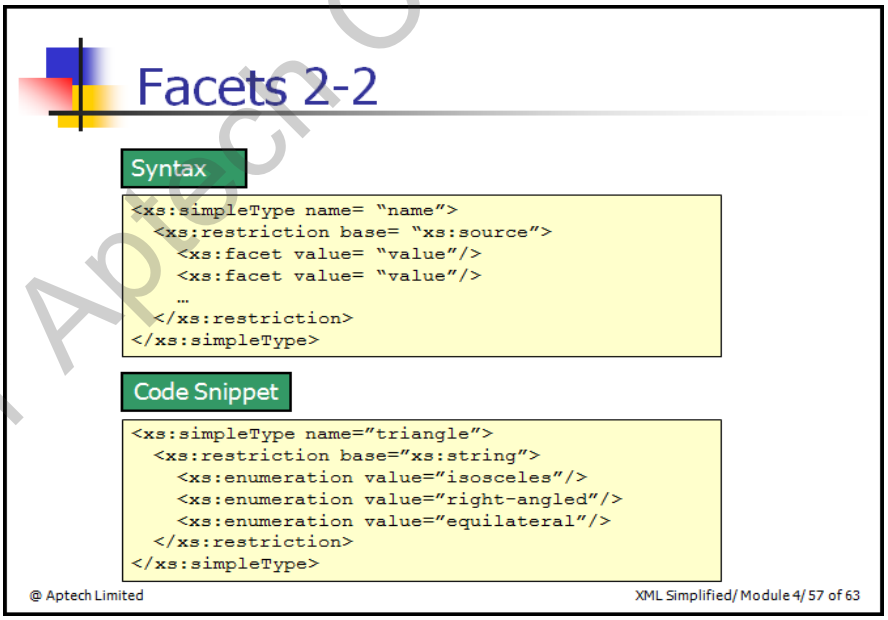


Facets 1-2

- They are used to restrict the set or range of values a datatype can contain. There are 12 facet elements declared using a common syntax.

Facet	Description
minExclusive	Specifies the minimum value for the type that excludes the value provided.
minInclusive	Specifies the minimum value for the type that includes the value provided.
maxExclusive	Specifies the maximum value for the type that excludes the value provided.
maxInclusive	Specifies the maximum value for the type that includes the value provided.
totalDigits	Specifies the total number of digits in a numeric type.
fractionDigits	Specifies the number of fractional digits in a numeric type.
length	Specifies the number of items in a list type or the number of characters in a string type.
minLength	Specifies the minimum number of items in a list type or the minimum number of characters in a string type.
maxLength	Specifies the maximum number of items in a list type or the maximum number of characters in a string type.
enumeration	Specifies an allowable value in an enumerated list.
whiteSpace	Specifies how whitespace should be treated within the type.
pattern	Restricts string types.

@ Aptech Limited XML Simplified/ Module 4/ 56 of 63



Facets 2-2

Syntax

```
<xs:simpleType name= "name">
  <xs:restriction base= "xs:source">
    <xs:facet value= "value"/>
    <xs:facet value= "value"/>
    …
  </xs:restriction>
</xs:simpleType>
```

Code Snippet

```
<xs:simpleType name="triangle">
  <xs:restriction base="xs:string">
    <xs:enumeration value="isosceles"/>
    <xs:enumeration value="right-angled"/>
    <xs:enumeration value="equilateral"/>
  </xs:restriction>
</xs:simpleType>
```

@ Aptech Limited XML Simplified/ Module 4/ 57 of 63

Using slides 56 and 57, explain the facets.

With XML Schemas, custom restrictions can be specified on XML elements and attributes. These restrictions are called facets.

Facets are used to restrict the set or range of values a data type can contain. The value range defined by the facet must be equal to or narrower than the value range of the base type.

There are 12 facet elements, declared using a common syntax. They each have a compulsory value attribute that indicates the value for the facet. One restriction can contain more than one facet. Any values appearing in the instance and value spaces must conform to all the listed facets.

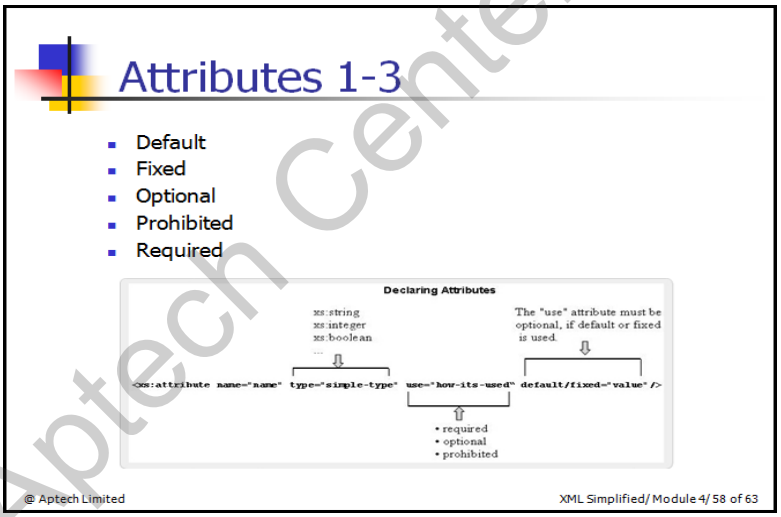
Table on slide 56 depicts the constraining facets.

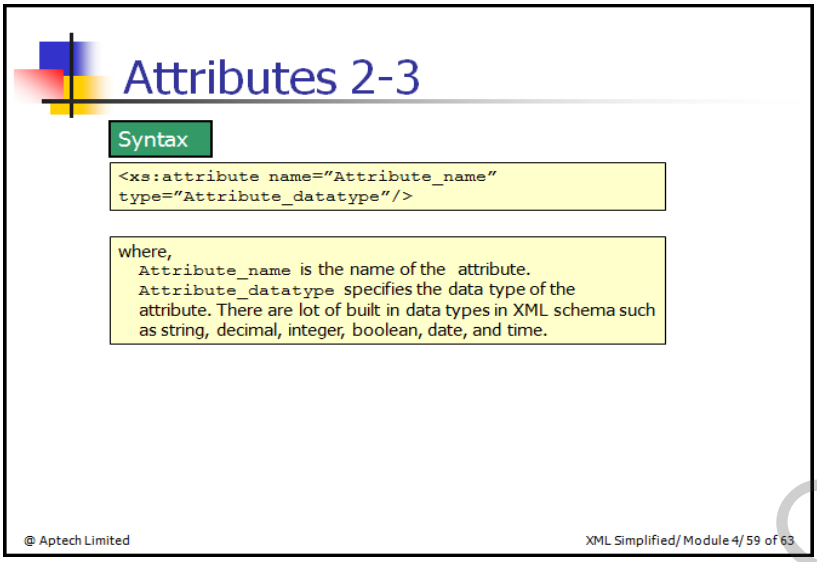
Explain the syntax and code on slide 57 which demonstrates that the value attribute gives the value of that facet.

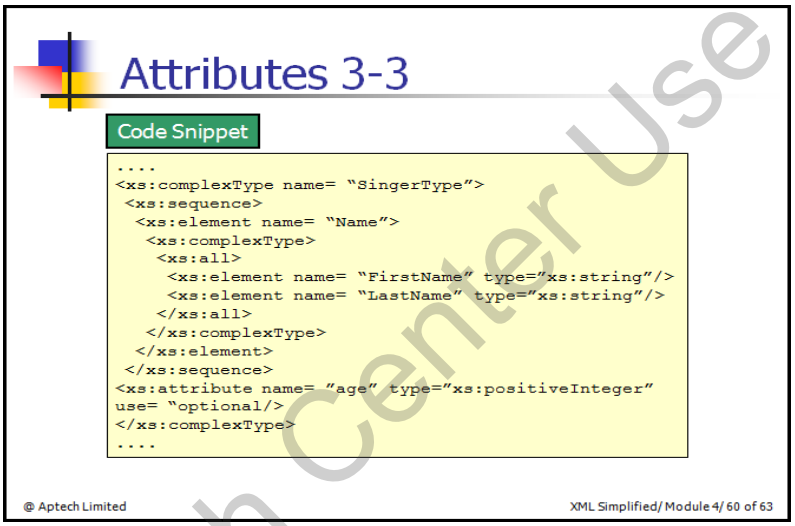
Here, the facet enumeration is added to the restriction with the value attribute as either `isosceles`, or `right-angled`, or `equilateral`. So, an element declared to be of type triangle must be a string with a value of either `isosceles`, or `right-angled`, or `equilateral`.

Slides 58 to 60

Let us understand the attributes.







Using slides 58 to 60, explain the attributes.

XML elements can contain attributes that describe elements. Within XML Schemas, attribute declarations are similar to element declarations. To declare an attribute, `<xs:attribute>` element is used.

An attribute can be indicated whether it is required or optional or whether it has a default value. A default value is automatically assigned to the attribute when no other value is specified.

The `use` attribute specifies whether the attribute is required or optional.

Explain different values of attributes. The figure on slide 58 depicts the attribute declaration.

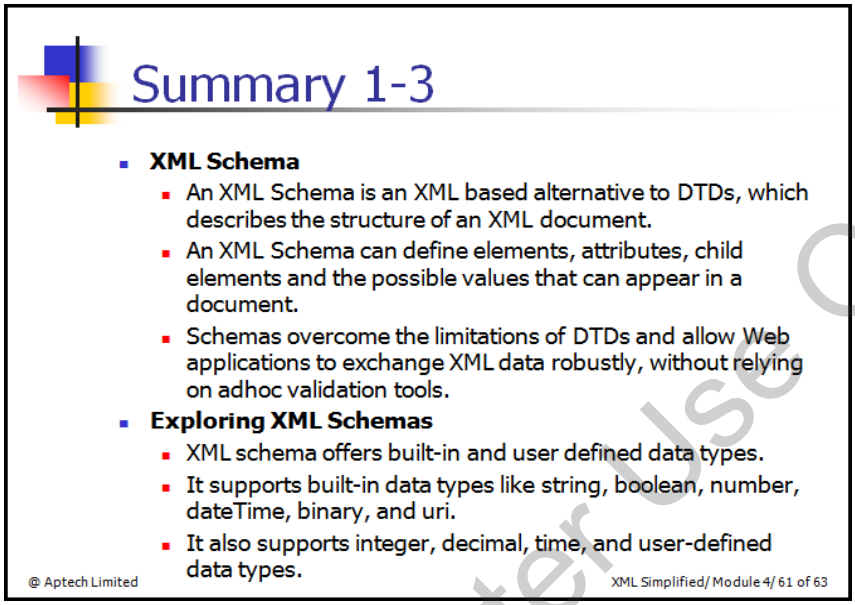
Explain the syntax for defining an attribute using slide 59.

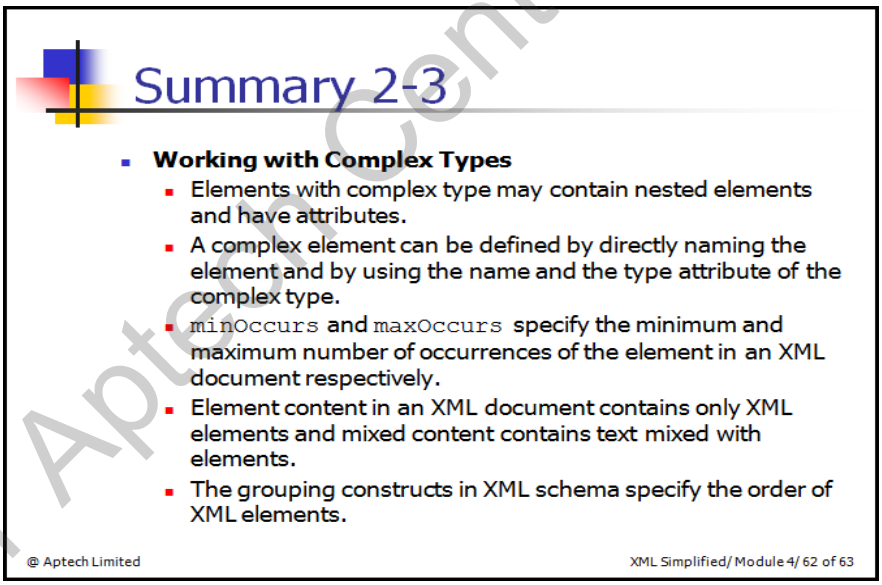
Explain the code on slide 60 which demonstrates the properties of attribute.

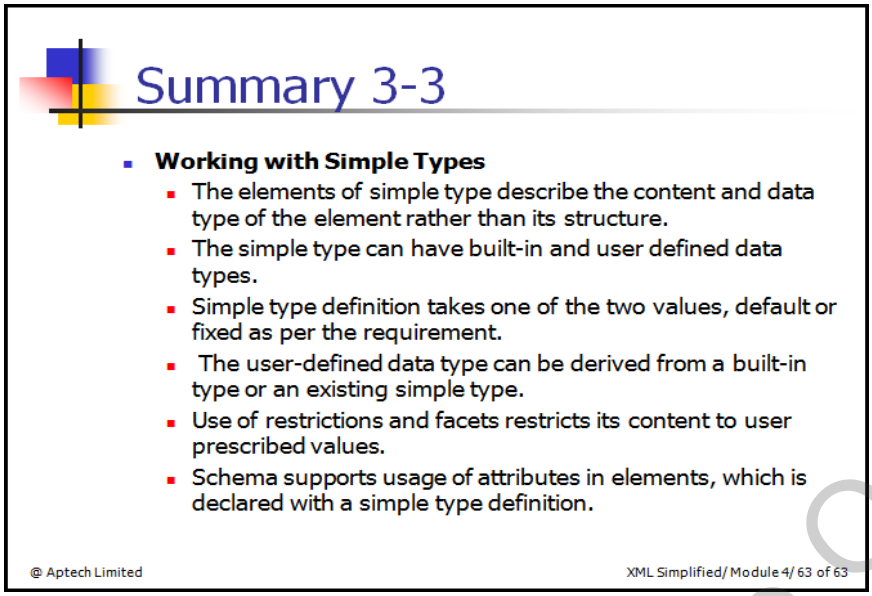
In the example, `xs:attribute` elements comes after `xs:sequence` and `xs:group` that forms the body of the element. The element `Name` can have an optional attribute named `age` with type `positiveInteger`.

Slides 61 to 63

Let us summarize the session.







Using slides 61 to 63, summarize the session. End the session, with a brief summary of what has been taught in the session.

4.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Style Sheets that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 5 - Style Sheets

5.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

5.1.1 Objectives

By the end of this session, the learners will be able to:

- Style Sheets
- Selectors in CSS
- Properties and values
- Inheritance and Cascades in CSS

5.1.2 Teaching Skills

To teach this session, you should be well-versed with style sheets. The topics such as how to use Cascading Style Sheets to format XML document, and the cascading order of style rules are also covered.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

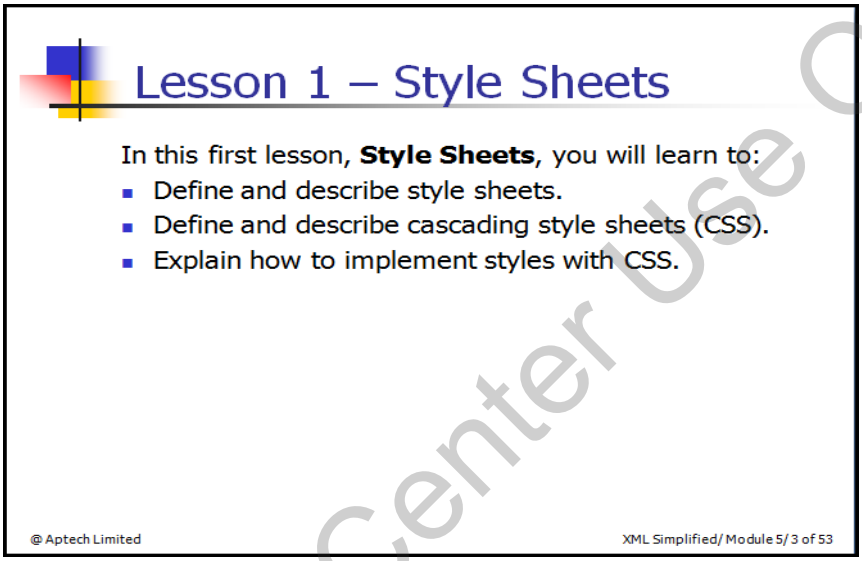
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces style sheets and how to create style sheets. They will also know understand how to use Cascading Style Sheets (CSS) to format XML document and the cascading order of style rules.

5.2 In-Class Explanations

Slide 3

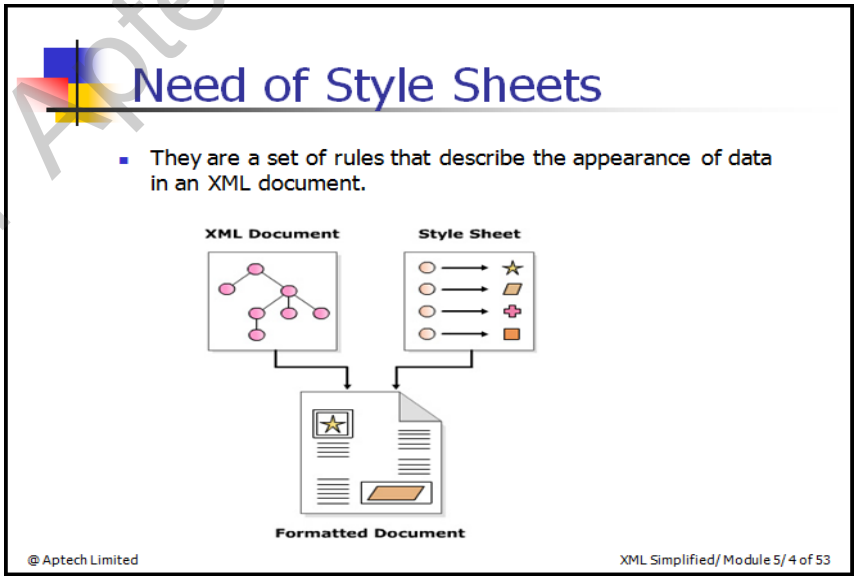
Let us understand the objectives of lesson 1.



Using slide 3, explain the objectives of lesson 1.

Slide 4

Let us understand the need for style sheets.



Using slide 4, explain the need for style sheets.

Style sheets are a set of rules that describe the appearance of data in an XML document.

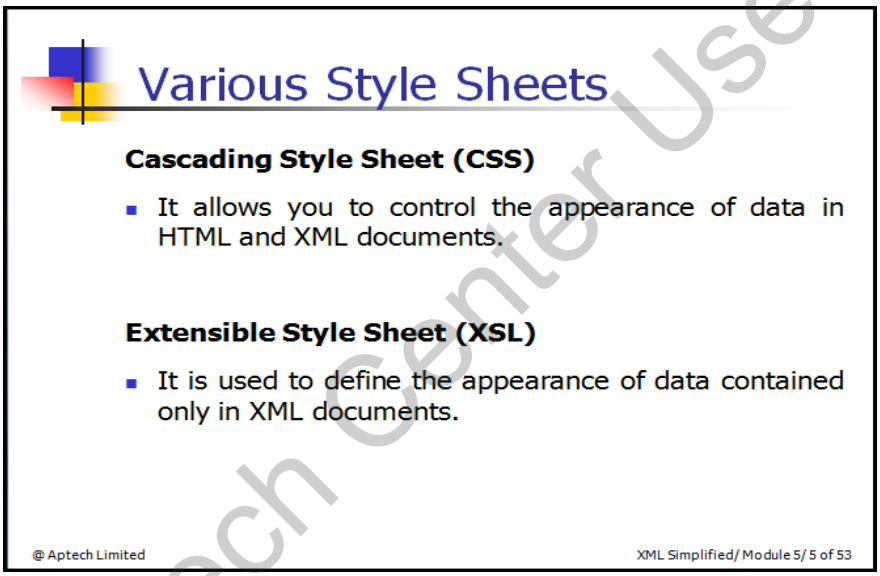
XML was inspired by the problems posed by presentational markup. Presentational markup does not describe data; it defines the appearance of data. In a document, if you had to change all proper nouns from bold to italics, you would have to do it manually for each proper noun. Presentation markup failed to provide same look and feel across multiple devices such as computers, Personal Digital Assistant (PDA) devices, and cell phones.

Style sheets and XML solve these problems. XML describes data. Style sheets define the appearance of data. However, both XML and style sheets are defined in separate files.

Explain the figure on slide 4 that depicts the formatted document with applied style and XML.

Slide 5

Let us understand the various style sheets.



Using slide 5, explain various style sheets.

There are several style sheets available. However, following two are the most popular style sheets:

- Cascading Style Sheet (CSS)
- EXtensible Stylesheet (XSL)

Explain the CSS that allows you to control the appearance of data in HTML and XML documents by providing various properties to define:

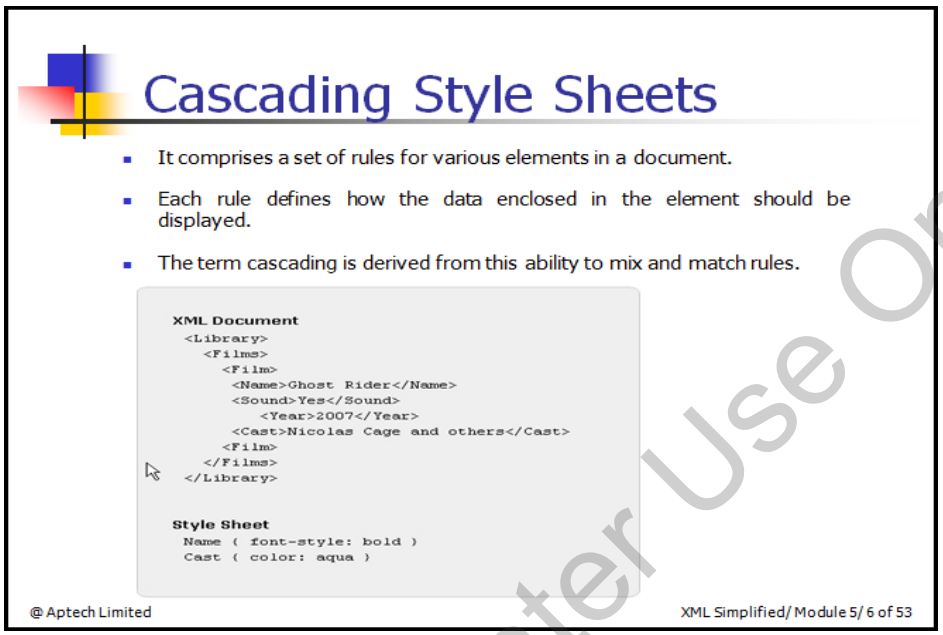
- Position and size of data to be displayed
- Foreground and background color of data
- Font to be used to display data
- Spacing between data

Explain EXtensible Stylesheet (XSL).

Mention XSL is a style sheet language used to define the appearance of data contained only in XML documents. However, it also allows you to transform XML documents.

Slide 6

Let us understand the structure of CSS.



Using slide 6, explain the structure of CSS.

CSS is a rule-based language invented in 1996 to write formatting instructions for data contained in HTML documents.

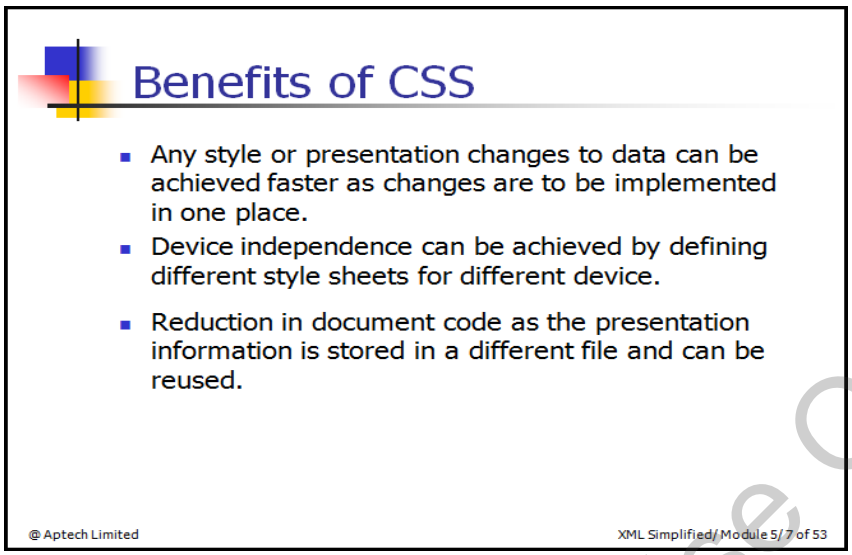
A CSS style sheet comprises a set of rules for various elements in a document. Each rule defines how the data enclosed in the element should be displayed.

Style sheet rules can be combined from different sources, or subsets can be created or the rules can be overridden. The term **cascading** is derived from this ability to mix and match rules.

Figure the code on slide 6 that shows an example of CSS.

Slide 7

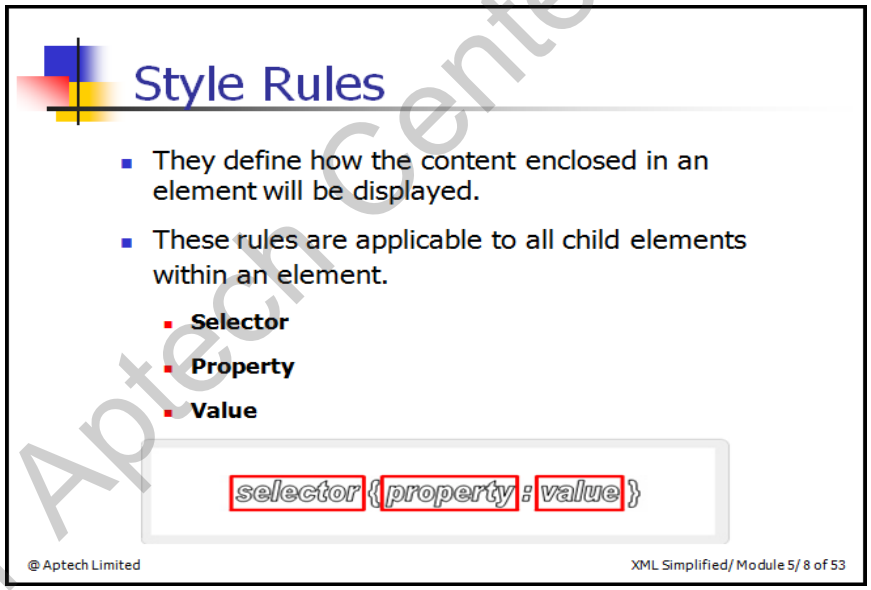
Let us understand the benefits of CSS.



Using slide 7, explain the benefits of CSS.

Slide 8

Let us understand the style rules.



Using slide 8, explain the style rules.

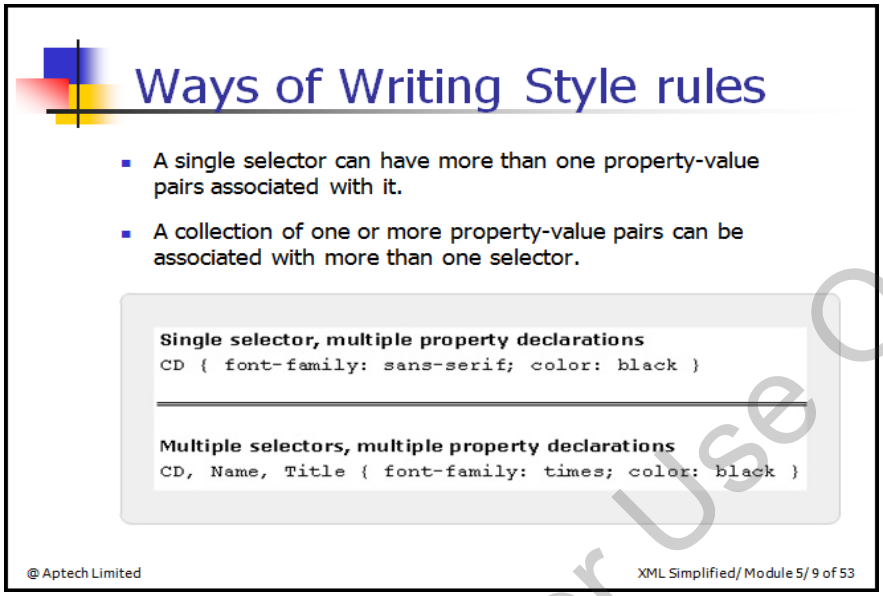
A style sheet in CSS comprises a set of style rules. These rules define how the content enclosed in an element will be displayed. These rules are applicable to all child elements within an element. A style rule comprises a selector, a property, and a value. A property and a value separated with a colon are referred as property declaration. Figure on slide 8 depicts the style rules.

Explain in detail about selector, property, and value.

In CSS, style rules can comprise more than one selector. To include multiple selectors or group multiple selectors, one needs to provide a comma-separated list of element names.

Slide 9

Let us understand the ways of writing style rules.



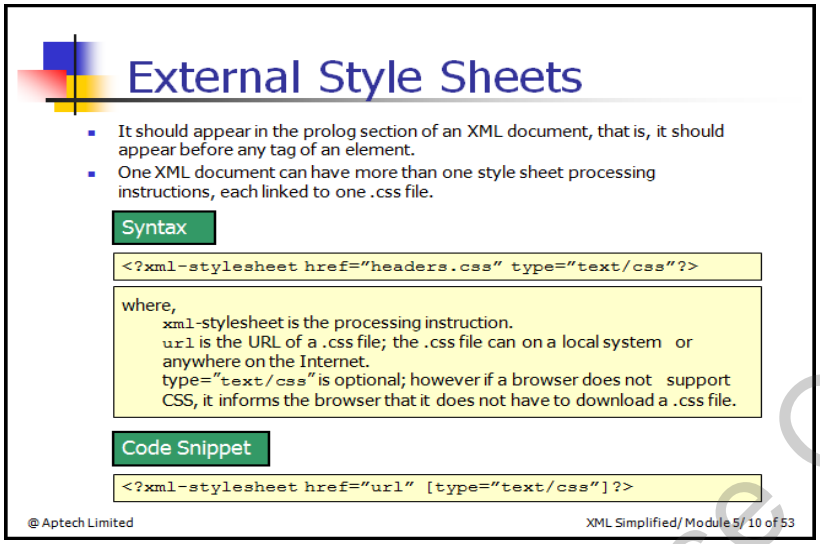
Using slide 9, explain the ways of writing style rules.

A single selector can have more than one property-value pairs associated with it. For example, figure on slide 9 shows a `CD` element having two property declarations – one to set the font family to sans-serif, and other to set the color of text to black. Notice the property-value pairs are separated by a semi-colon.

Similarly, a collection of one or more property-value pairs can be associated with more than one selector. For example, figure on slide 9 shows two property declarations assigned to three elements namely `CD`, `Title`, and `Name`.

Slide 10

Let us understand the external style sheets.



Using slide 10, explain the external style sheets.

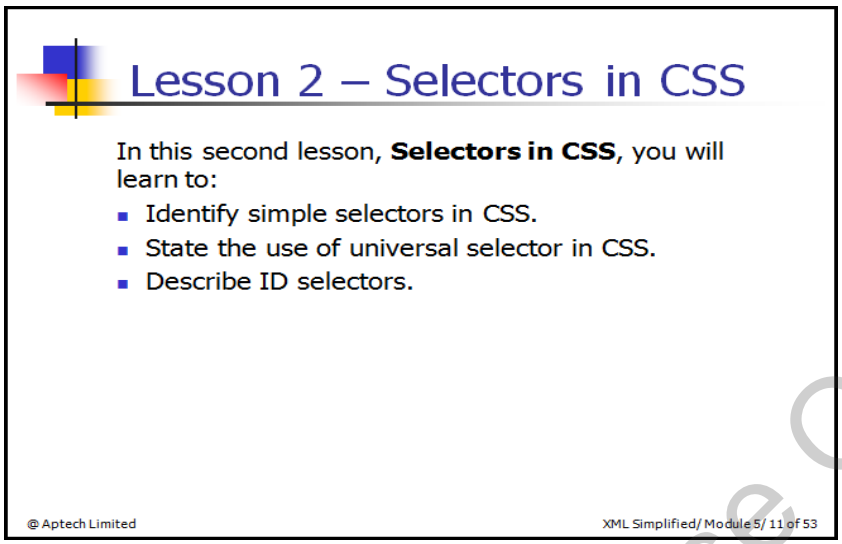
In CSS, the style rules are written in a file with the extension `.css`. This file is associated with an XML document using a style sheet processing instruction. A few points to note about style sheet processing instruction are:

- It should appear in the prolog section of an XML document, that is, it should appear before any tag of an element.
- One XML document can have more than one style sheet processing instructions, each linked to one `.css` file.

Explain the syntax and code snippet on slide 10 for of external style sheet.

Slide 11

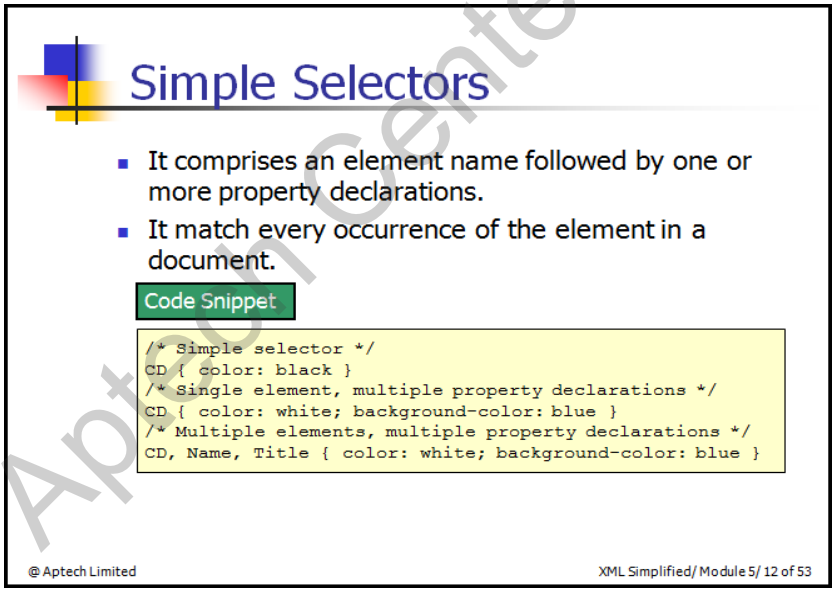
Let us understand the objectives of lesson 2.



Using slide 11, explain the topics covered in lesson 2.

Slide 12

Let us understand the simple selectors.



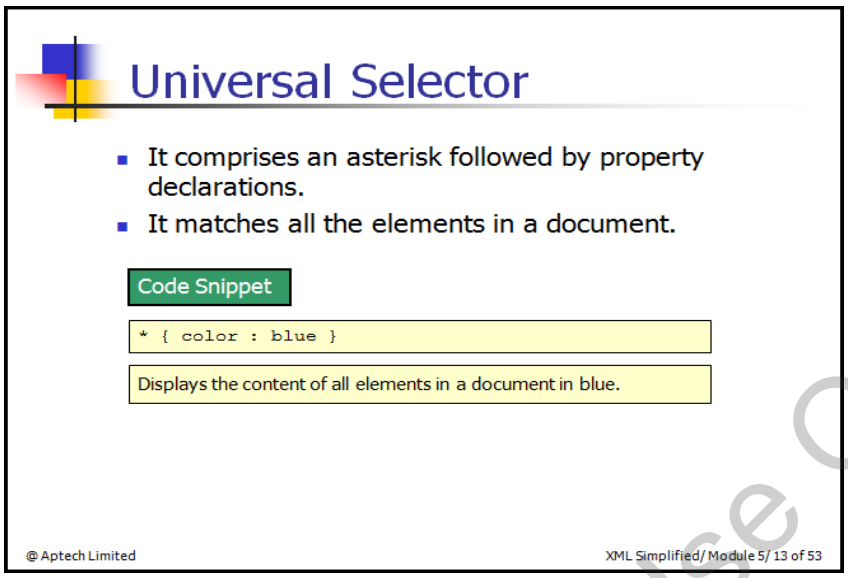
Using slide 12, explain the simple selectors.

Simple selector comprises an element name followed by one or more property declarations. Same property declarations can be assigned to multiple elements by separating element names with a comma. Simple selectors match every occurrence of the element in a document.

Explain the code snippet on slide 12 demonstrating the simple selectors.

Slide 13

Let us understand the universal selectors.



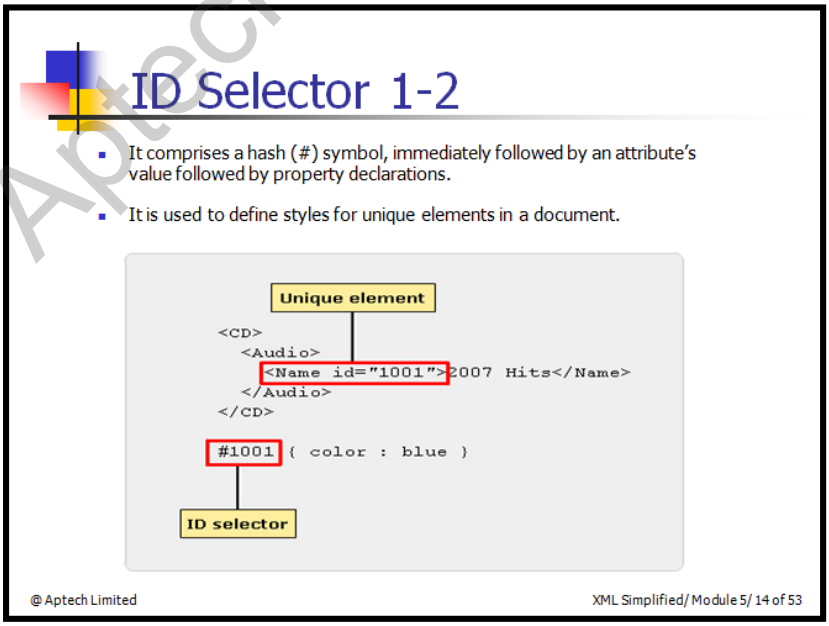
Using slide 13, explain the universal selectors.

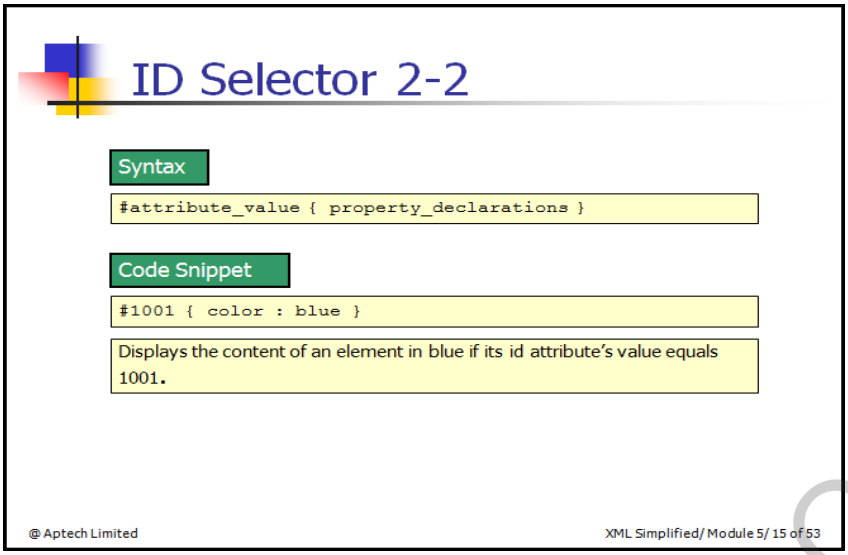
A universal selector comprises an asterisk followed by property declarations. It is used when you want to assign the same style rules to all the elements in a document. A universal selector matches all the elements in a document. The code shown on the slide displays the content of all elements in a document in blue.

Explain the code snippet for the same using slide 13.

Slides 14 and 15

Let us understand the ID selectors.





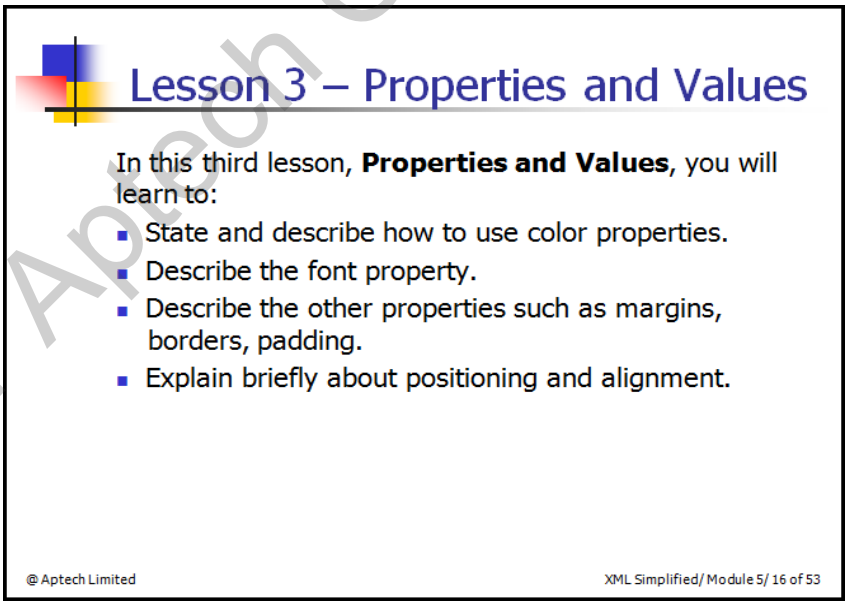
Using slides 14 and 15, explain the ID selectors.

An ID selector comprises a hash (#) symbol, immediately followed by an attribute's value followed by property declarations. It is used to define styles for unique elements in a document. For example, if you want the data of a unique element to be in a different style, you would define an ID selector for it. Unique element is one which has one of its attributes named as id as shown in figure on slide 14.

Explain the syntax and code snippet for ID selectors using slide 15.

Slide 16

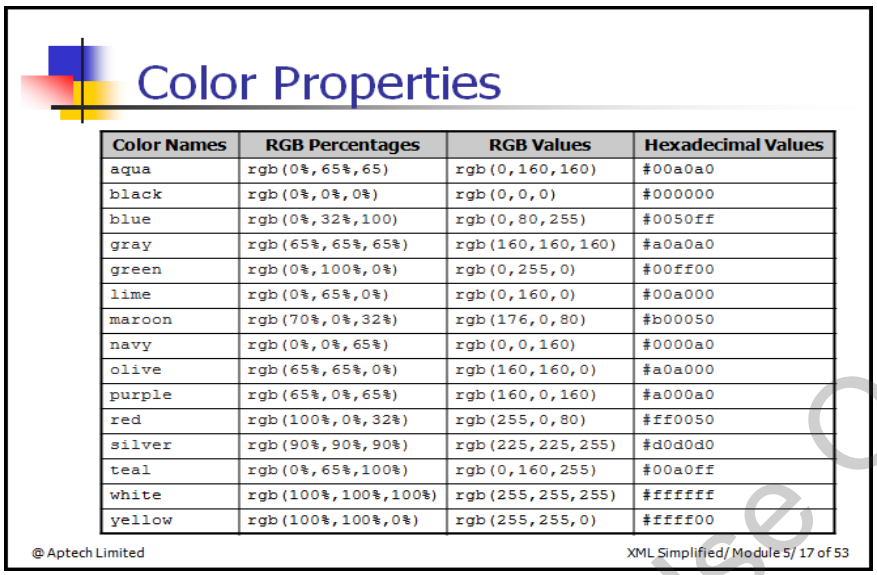
Let us understand the properties and values.



Using slide 16, explain the topics covered in lesson3.

Slide 17

Let us understand the color properties.

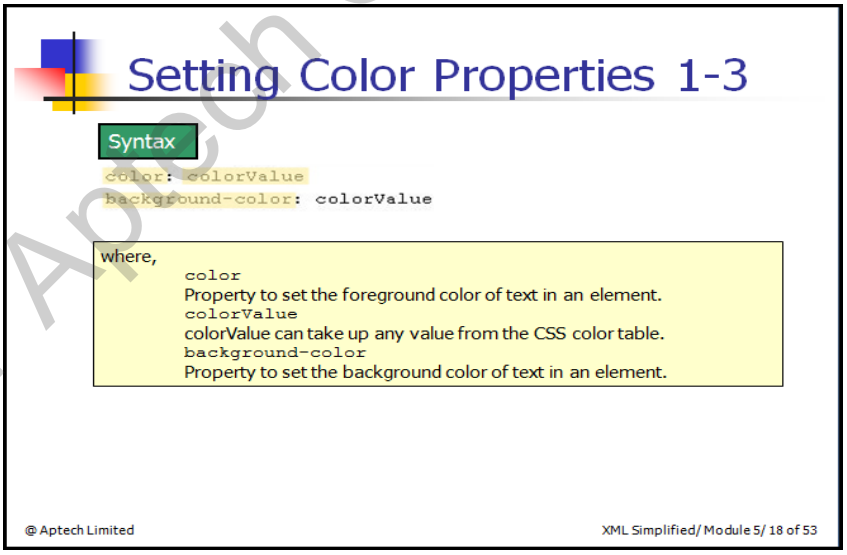


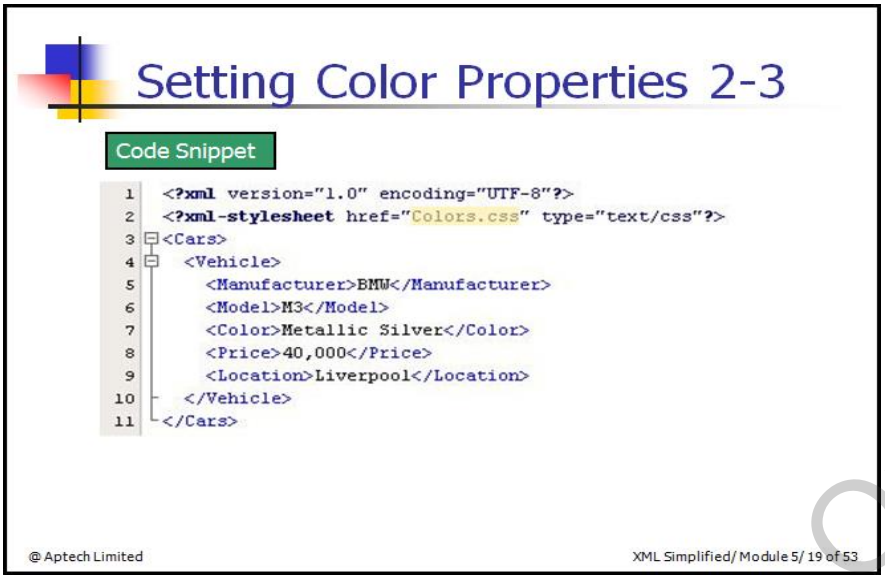
Using slide 17, explain the color properties.

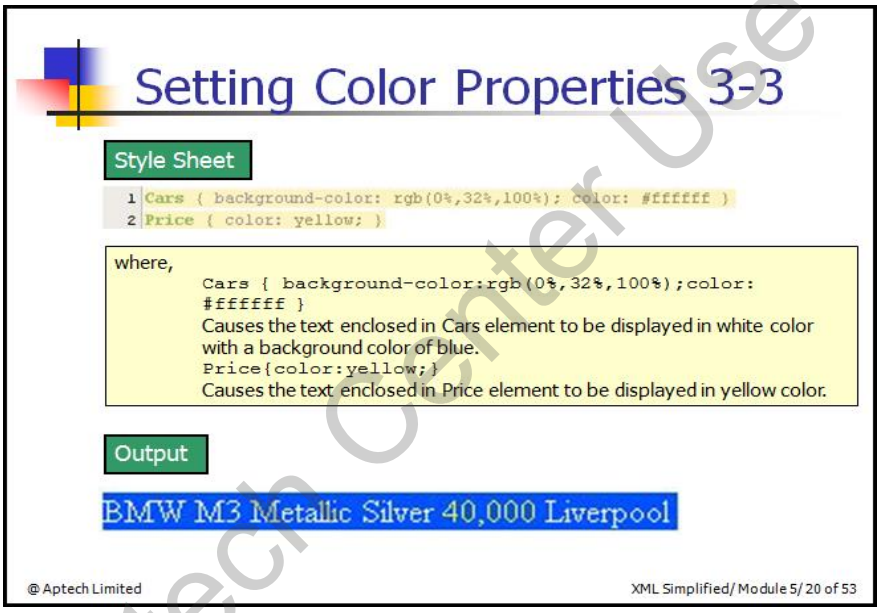
The CSS provides properties to set the foreground and background color of text. CSS uses color name, red-green-blue (RGB) values, RGB percentages and hexadecimal values to specify color values. The various ways in which color values are specified is shown in table on slide 17.

Slides 18 to 20

Let us understand how to set color properties.







Using slide 18, explain how to set color properties.

The CSS specification provides the properties color and background-color to set the foreground and background color of text enclosed in elements.

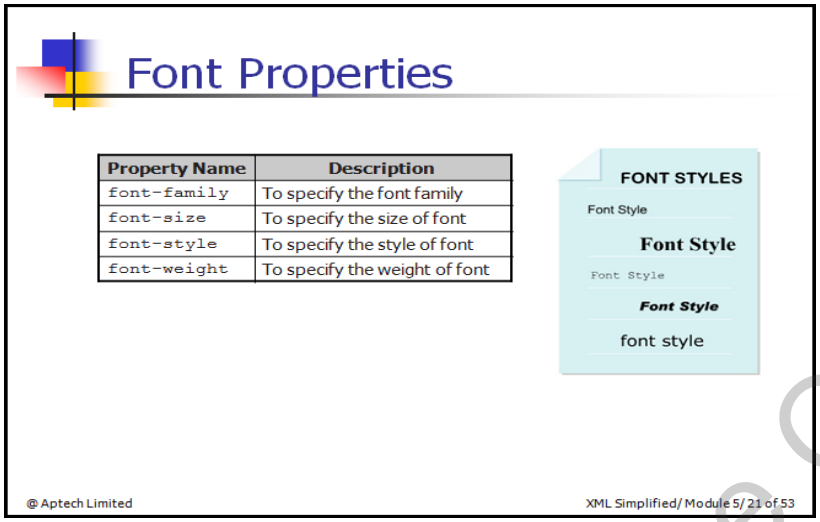
Explain the figure on slide 18 that depicts the syntax for setting properties.

Explain the figure on slide 19 that shows the code for style rules defined in `Colors.css` file.

Explain the figure on slide 20 depicts the style sheet for `Colors.css` file. Also, explain the output for the same.

Slide 21

Let us understand the font properties.

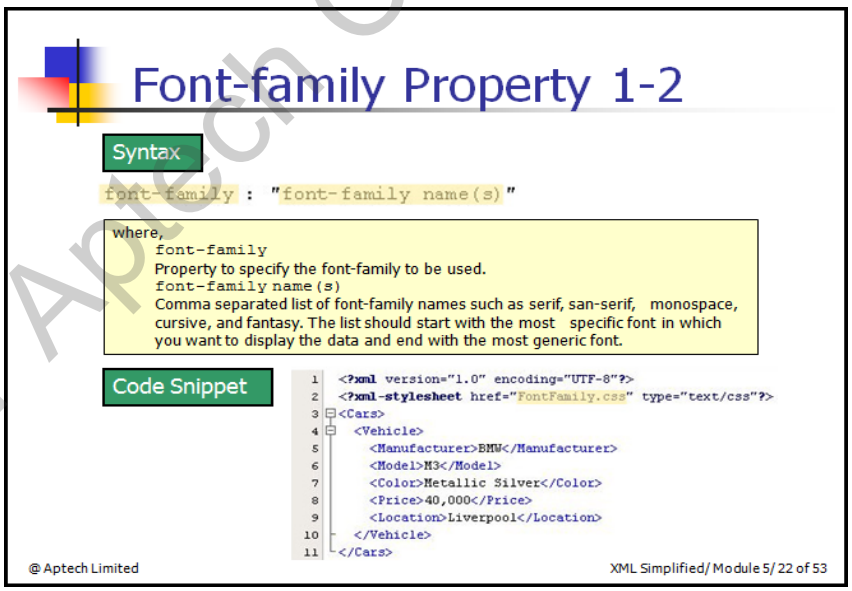


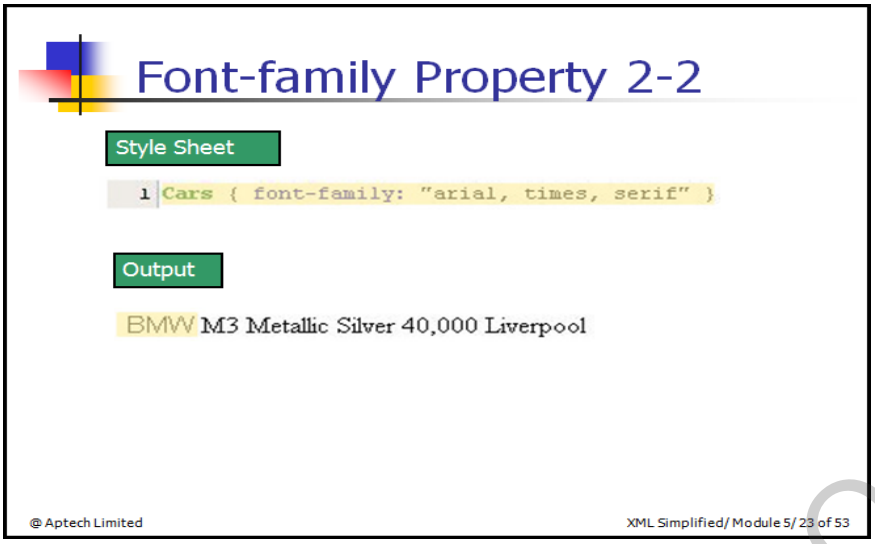
Using slide 21, explain the font properties.

The CSS specification defines several font properties to set the font family, font size, font style such as bold, italics, and so forth. Some of these properties are listed in table on slide 21.

Slides 22 and 23

Let us understand the `font-family` property.





Using slides 22 and 23, explain the `font-family` property.

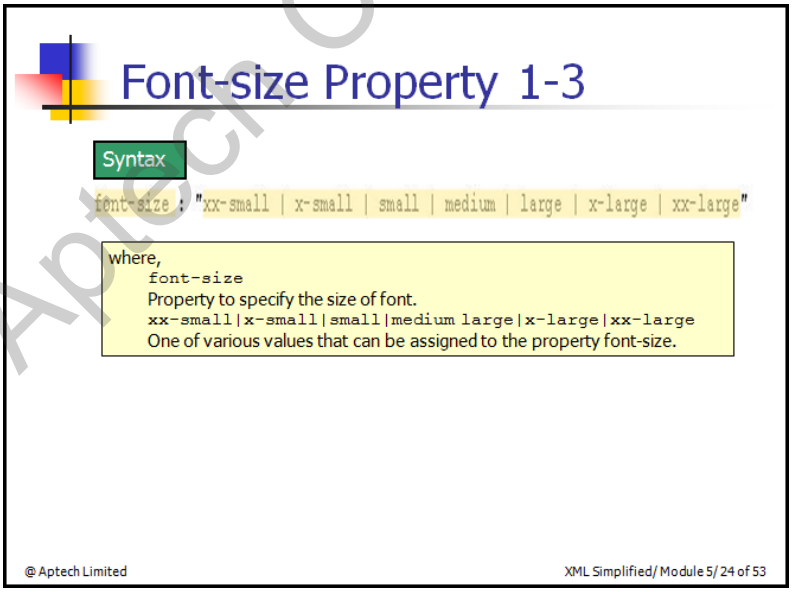
The `font-family` property is used to specify the name of the font family to be applied to an element.

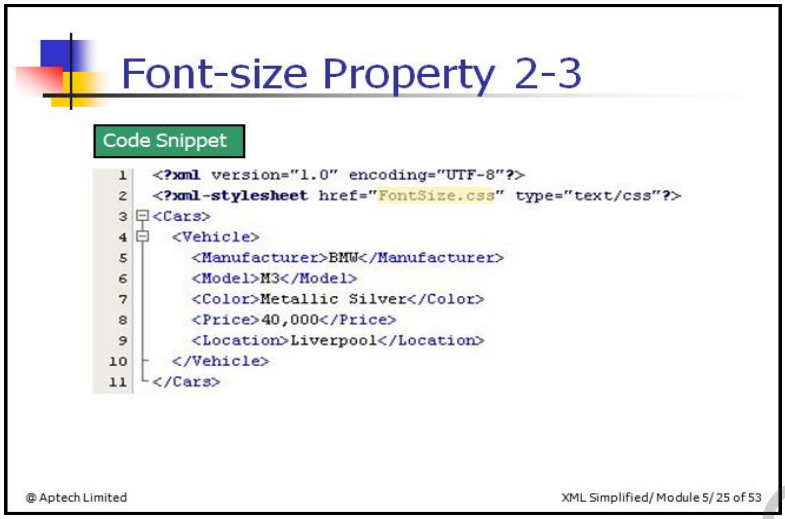
Explain the figure on slide 22 that shows the syntax for `font-family` property. Also, explain the code for style rules stored in `FontFamily.css` file.

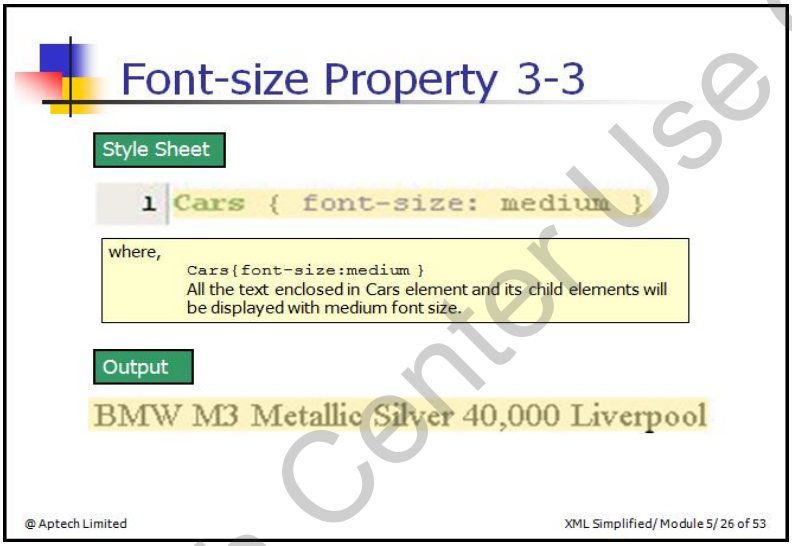
Then, explain slide 23 that depicts the style sheet for `FontFamily.css` file.

Slides 24 to 26

Let us understand the `font-size` property.







Using slides 24 to 26, explain the `font-size` property.

The `font-size` property is used to specify the size of font used to display element data. Explain the figure on slide 24 that shows the syntax for `font-size` property.

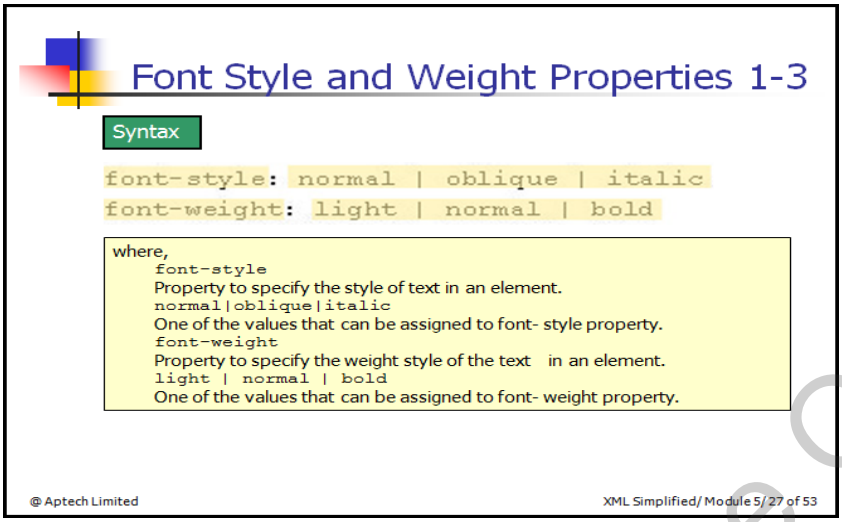
Then, explain the figure on slide 25 that shows the code for style rules defined in `FontSize.css` file.

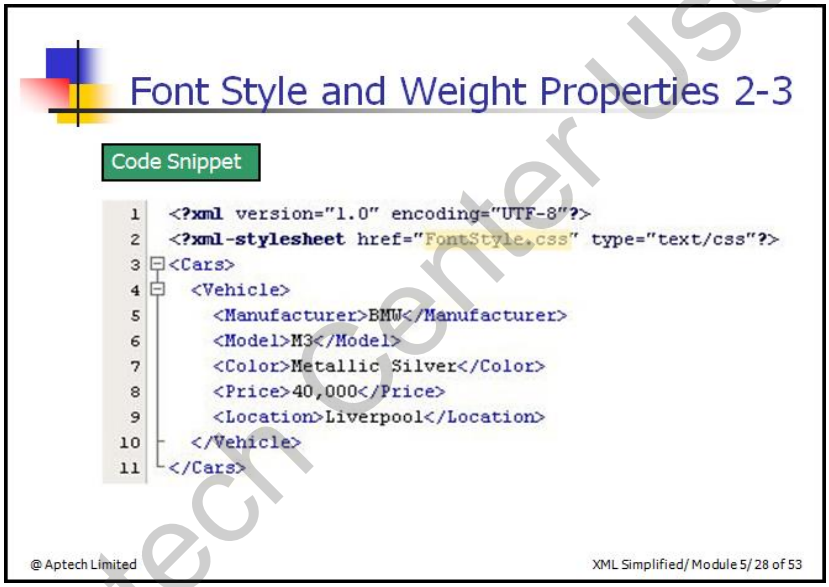
Finally, explain the style sheet for `FontSize.css` file on slide 26.

The text enclosed in the element `Cars` will be displayed in either **Arial**, **Times New Roman**, or **Serif** font. The rule becomes applicable to all the elements enclosed in `Cars` element. If the system does not support **Arial** font, then next fonts will be selected in sequence.

Slides 27 to 29

Let us understand the font style and weight properties.





Using slides 27 to 29, explain the font style and weight properties.

CSS provides two properties namely, `font-style` and `font-weight` to add emphasis or meaning to parts of data.

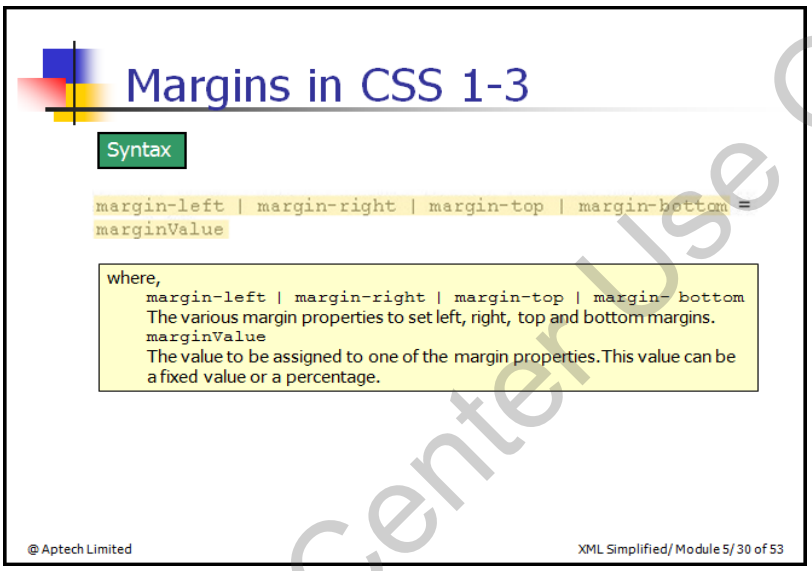
Explain the figure on slide 27 that shows the syntax for font style and weight properties.

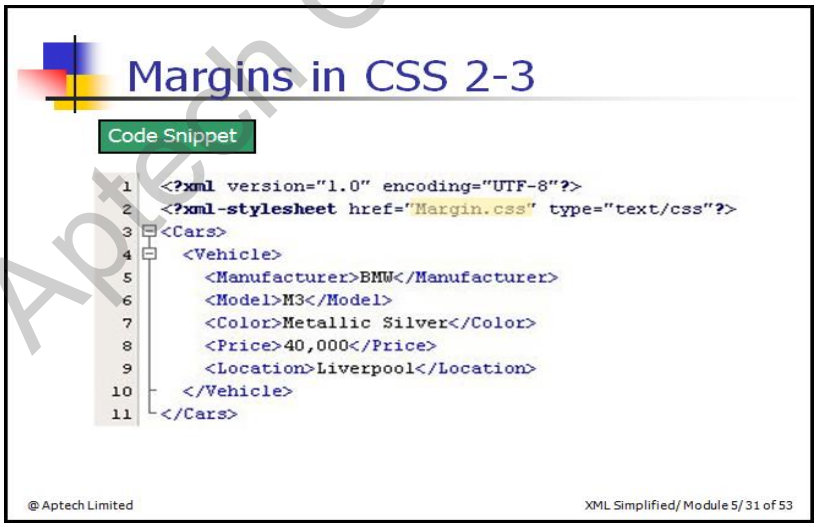
Explain the figure on slide 28 that shows the code for style rules defined in `FontStyle.css` file.

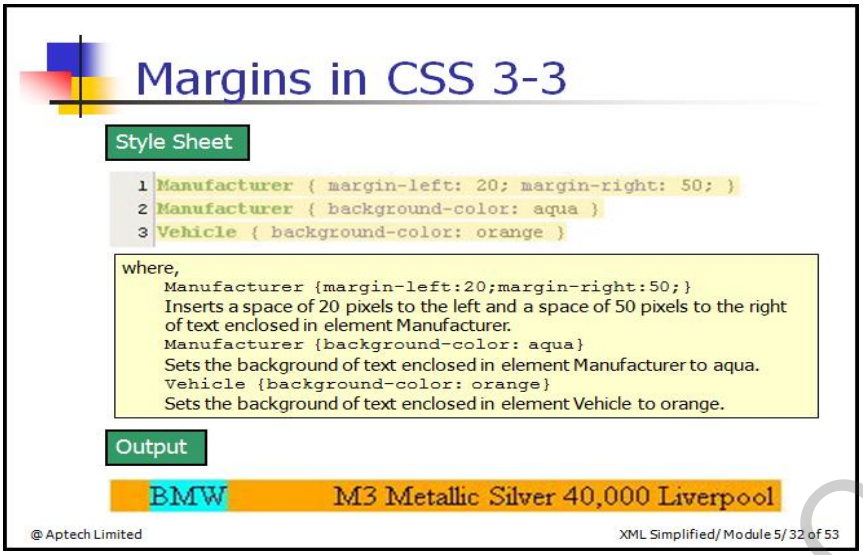
Explain the figure on slide 29 that depicts the style sheet for `FontStyle.css` file.

Slides 30 to 32

Let us understand the margins in CSS.







Using slide 30, explain margins in CSS.

Every element in an XML document is displayed in its own box. CSS provides four properties namely `margin-left, margin-right, margin-top,` and `margin-bottom` to insert space around the element's box.

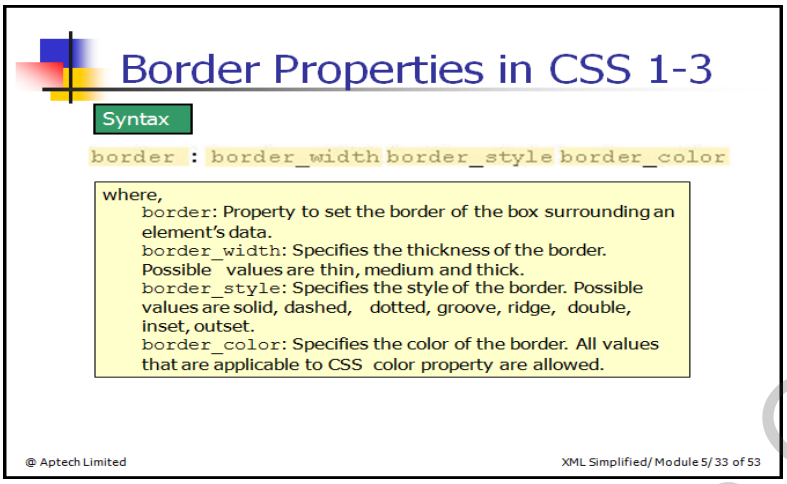
Explain the figure on slide 30 that shows the syntax for margins in css.

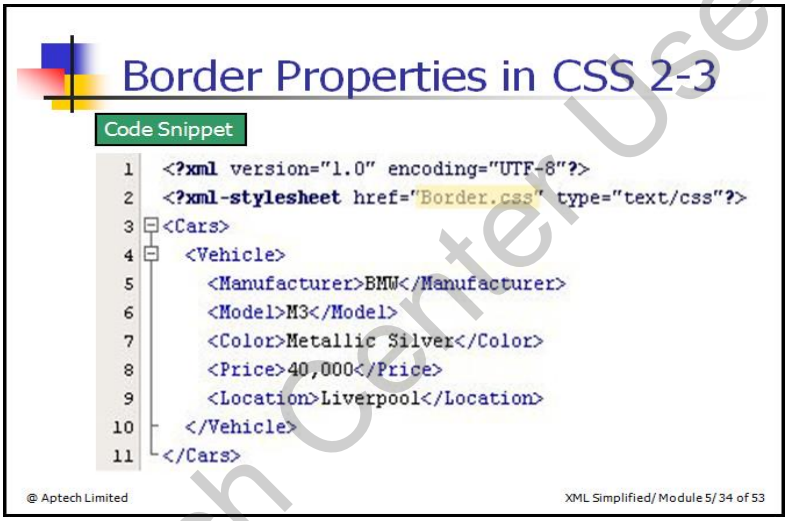
Explain the figure on slide 31 that shows the code for style rules defined in `Margin.css` file.

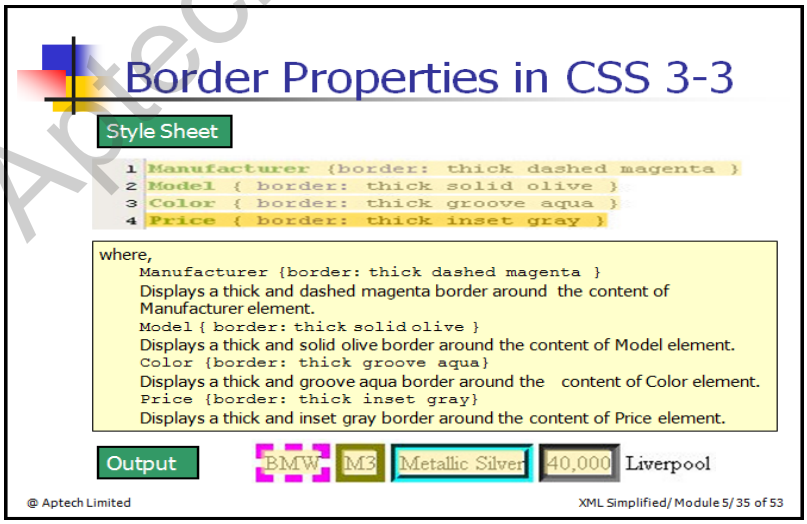
Explain the figure on slide 32 that depicts the style sheet for `Margin.css` file.

Slides 33 to 35

Let us understand the `border` properties in CSS.







Using slides 33 to 35, explain the `border` properties in CSS.

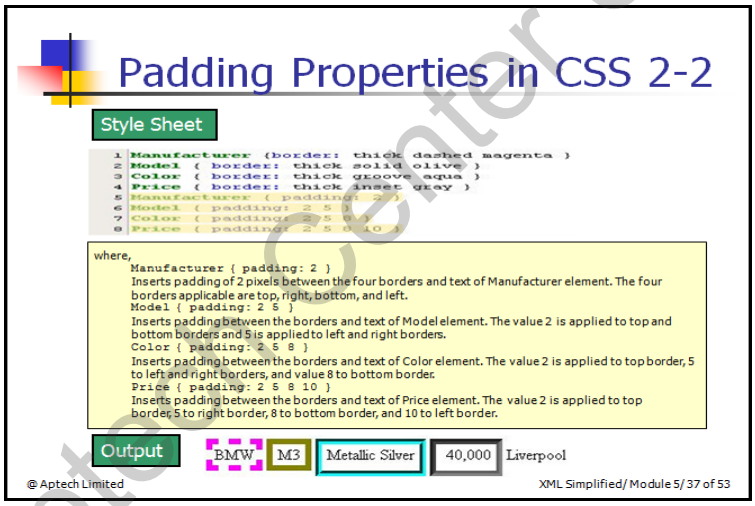
Borders are rectangular outlines surrounding an element's box. CSS provides properties to create dotted, solid, and groove borders to name a few.

Explain the syntax and the style rules for border properties explained on slides 33 and 34.

Then, explain the figure on slide 35 that depicts the style sheet for `Border.css` file.

Slides 36 and 37

Let us understand the `padding` properties in CSS.



Using slides 36 and 37, explain the `padding` properties in CSS.

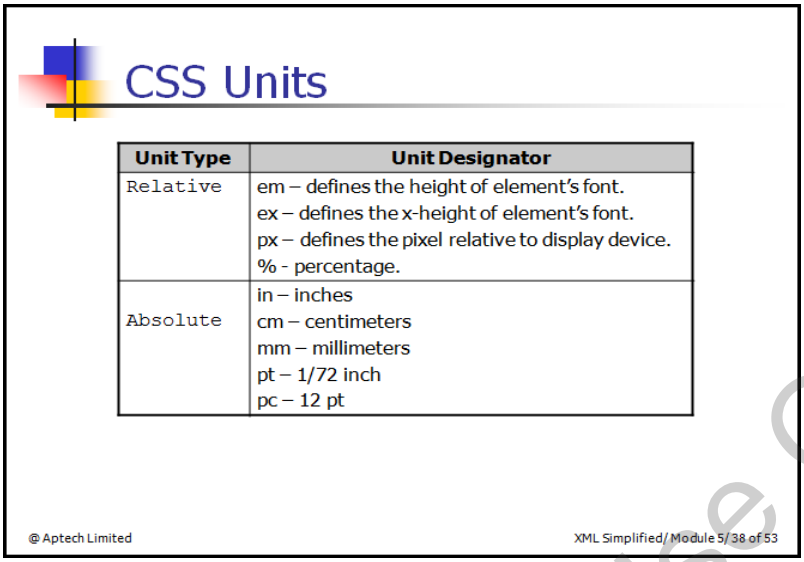
The border property surrounds the text in an element with an outline. To insert space between the border and the text of an element, CSS provides the `padding` property.

Explain the figure showing the syntax for `padding` property and the code for style rules defined in `Padding.css` file explained on slides 36 and 37.

Then, explain the figure on slide 37 that depicts the style sheet for `Padding.css` file.

Slide 38

Let us understand the CSS units.



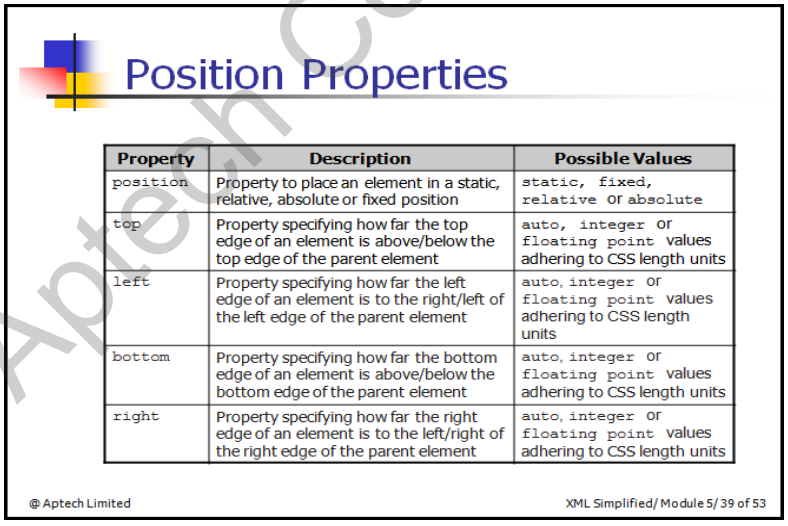
Using slide 38, explain the units in CSS.

The values assigned to CSS properties are expressed in length units.

Explain the table that lists the units, in slide 38.

Slide 39

Let us understand the `position` properties.

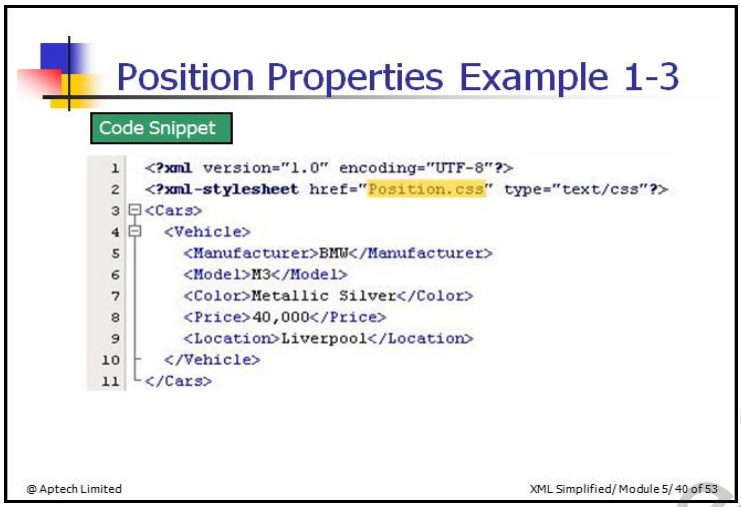


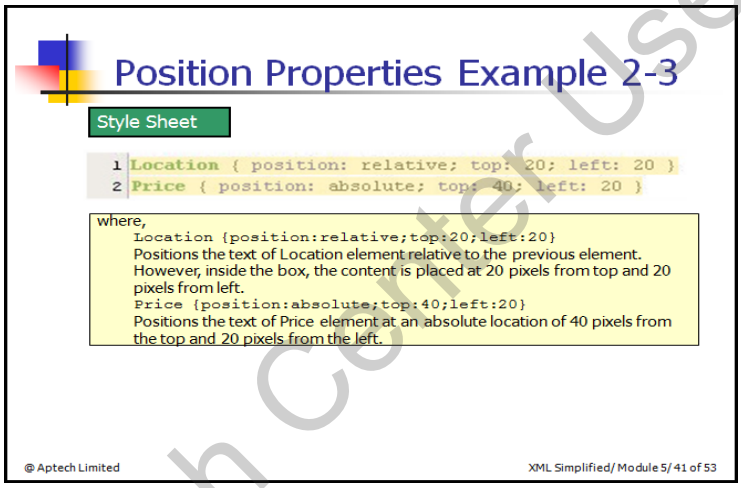
Using slide 39, explain the `position` properties.

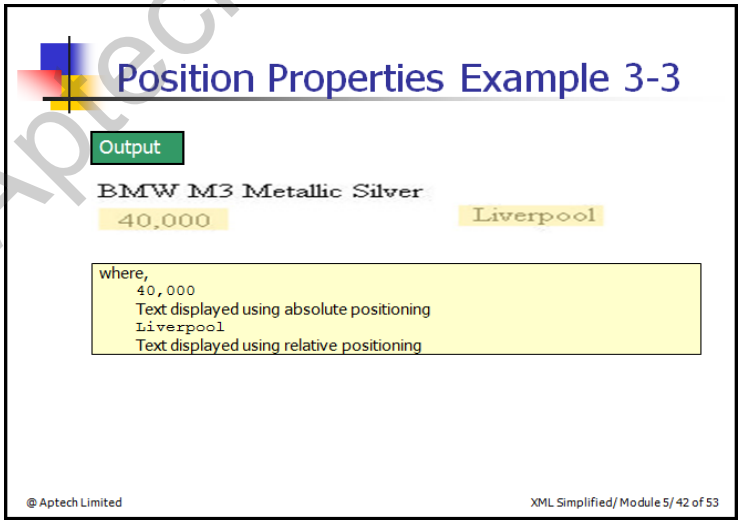
Every element's text is placed in a box of its own. Table lists the CSS positioning properties to position the text inside the box. Note that the properties `top`, `left`, `bottom`, and `right` are used only if value of position property is not static.

Slides 40 to 42

Let us understand the `position` properties example.





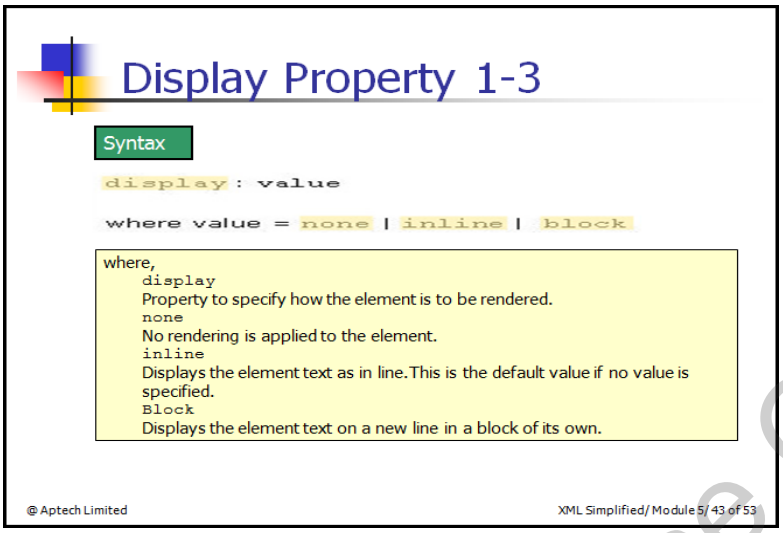


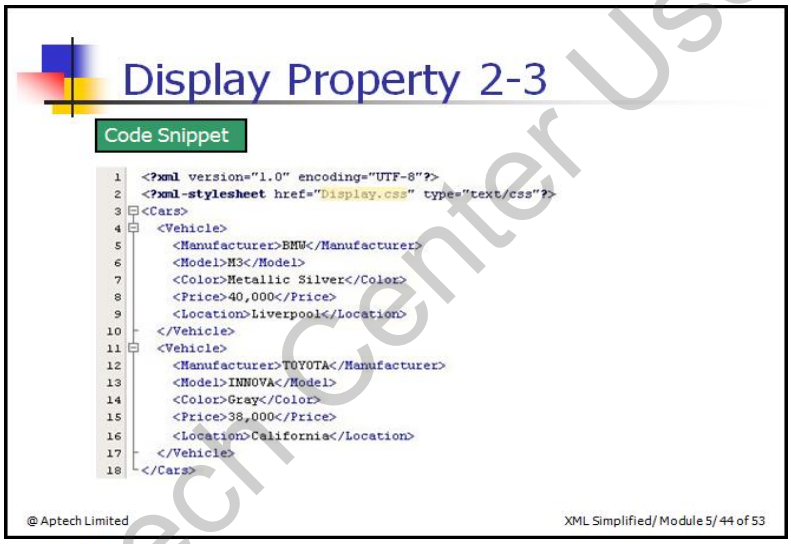
Using slides 40 to 42, explain the example of `position` properties.

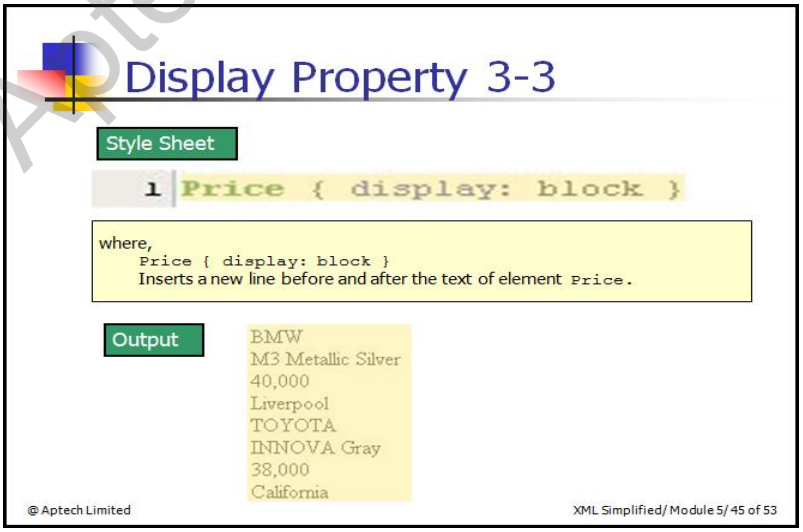
Explain the figure on slide 40 that explains the style rules defined in `Position.css` file. Also, explain the figure on slide 41 that depicts the style sheet for `Position.css` file.

Slides 43 to 45

Let us understand the `display` properties.







Using slides 43 to 45, explain the `display` properties.

In HTML, if you wanted the text to appear as new paragraph, you would use the <P> tag. The same can be achieved in XML by using CSS `display` property.

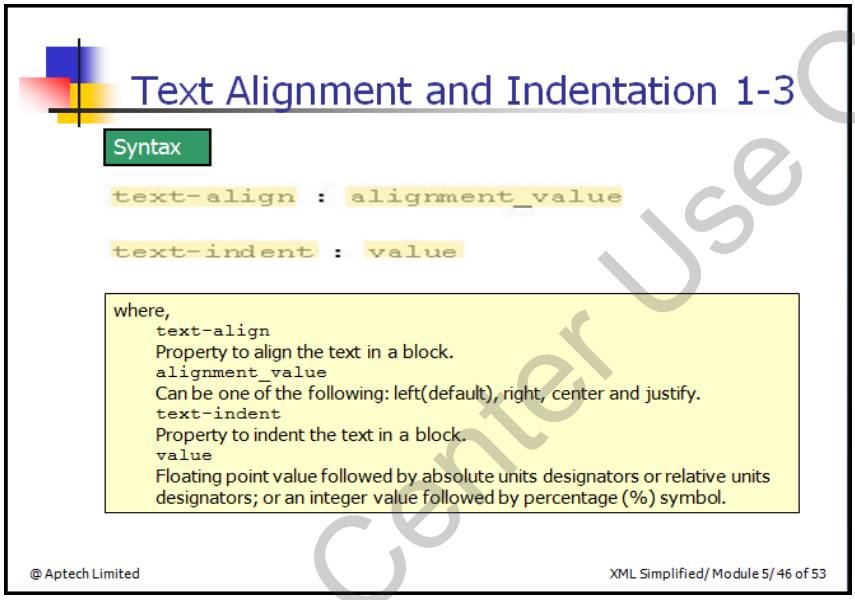
Explain the figure that shows the syntax for `display` property and also explain the code for style rules defined in `Display.css`.

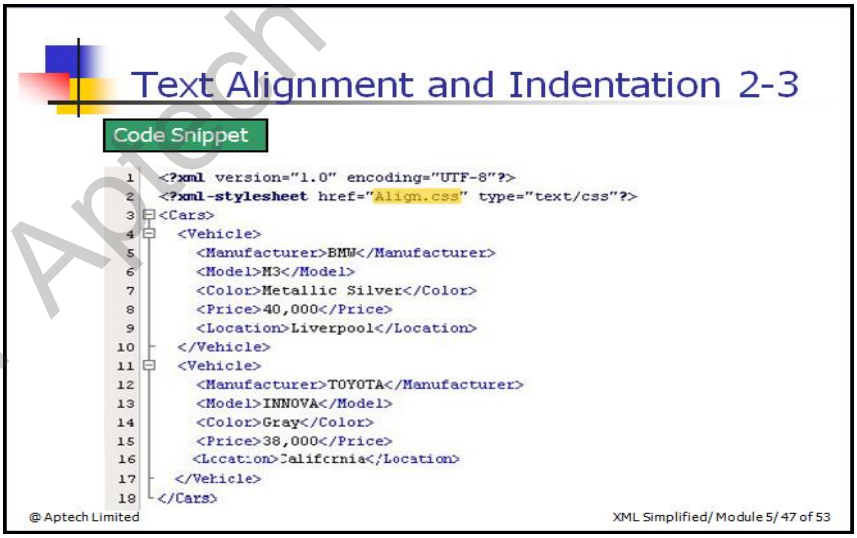
Then, explain the figure on slide 45 that depicts the style sheet for `Display.css` file.

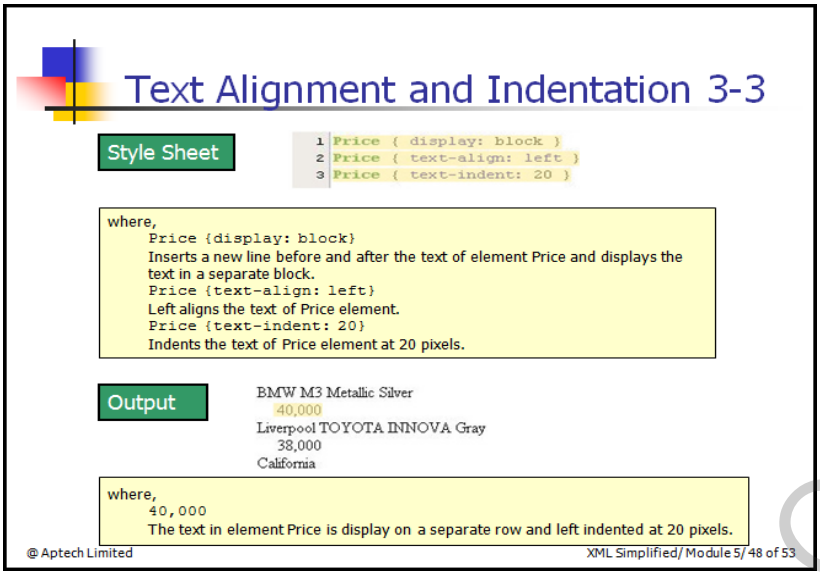
Data in each element is displayed on a separate row in block of its own.

Slides 46 to 48

Let us understand the text alignment and indentation.







Using slides 46 to 48, explain the text alignment and indentation.

The `display:block` property renders the text of an element in a separate block. Inside this block, you can align and indent the text using the CSS properties `text-align` and `text-indent` respectively.

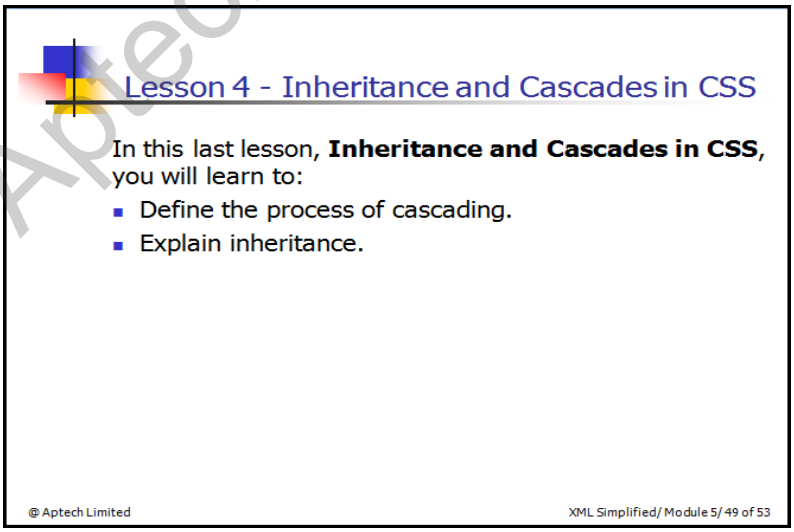
Explain the figure on slide 46 that shows the syntax for text alignment and indentation.

Then, explain the figure on slide 47 shows the code for style rules defined in `Align.css`. Also, the figure on slide 48 depicts the style sheet for `Align.css` file.

The text in element `Price` is displayed on a separate row and left indented at 20 pixels.

Slide 49

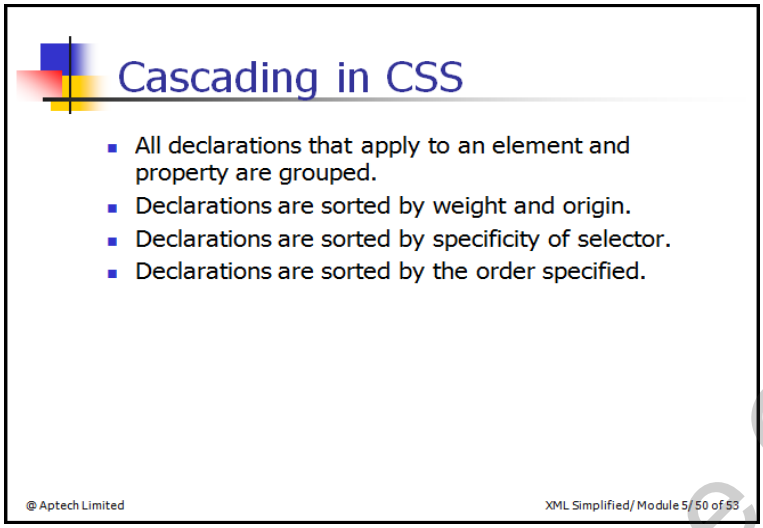
Let us understand the objectives of lesson 4.



Using slide 49, explain the objectives of lesson 4.

Slide 50

Let us understand the cascading in CSS.



Using slide 50, explain the cascading in CSS.

There are several ways in which style sheets can be defined. For example, one document can have more than one style sheet defined for it. A browser has its own style sheet. In such cases, it is possible that more than one style rule may exist for an element. Hence, the World Wide Web Consortium (W3C) defines the following rules to determine what style to apply to an element. The sorting of style rules begins with rule number 1. If the style rule matches rule number 1, then the search is over; otherwise the search continues with rule number 2 and so on.

1. Find all property declarations for an element in question. Apply the style rules if the element name matches the element in the selector. If there is no style rule defined for an element, the element inherits the style rules defined for the parent element. However, if the parent element does not have any style rules defined, then the element is rendered with default values.

2. Style rules declared as important are considered next. A style rule can be declared important in the following manner:

```
Threat { display: block ! important }
```

Important property declarations have a higher precedence over normal property declarations.

3. Next, the origin of style sheet is determined. A style sheet can have following sources:
   - **Author**: The author of the XML document can define a style sheet in an external document to format the XML document.
   - **User/Reader**: The end-user viewing the XML document can specify his/her personal style sheet to be used to format the XML document.

- **User-agent**: The browser is referred to as user-agent. A browser has its own default style sheet.

The style rules defined in the author's style sheet override the style rules defined in the user's style sheet. The author's and the user's style sheet both override the user-agent's style sheet.

4. Next, the specificity of a selector is determined in that a more specific selector will override the general selector. The specificity is determined by carrying out the following three activities:
   - Count the number of ID attributes in the selector.
   - Count the number of class attributes in a selector.
   - Count the number of element names in the selector.

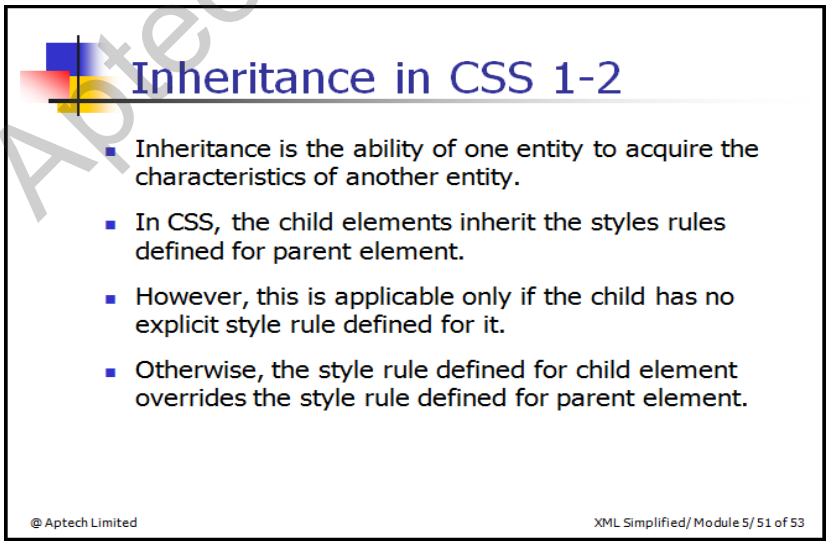
5. Next, write the three numbers with no commas or spaces and in the same sequence as shown earlier. Higher the number, higher is the specificity. Rules with higher specificity override the ones with lower specificity. For example, following is a list of selectors sorted by specificity:

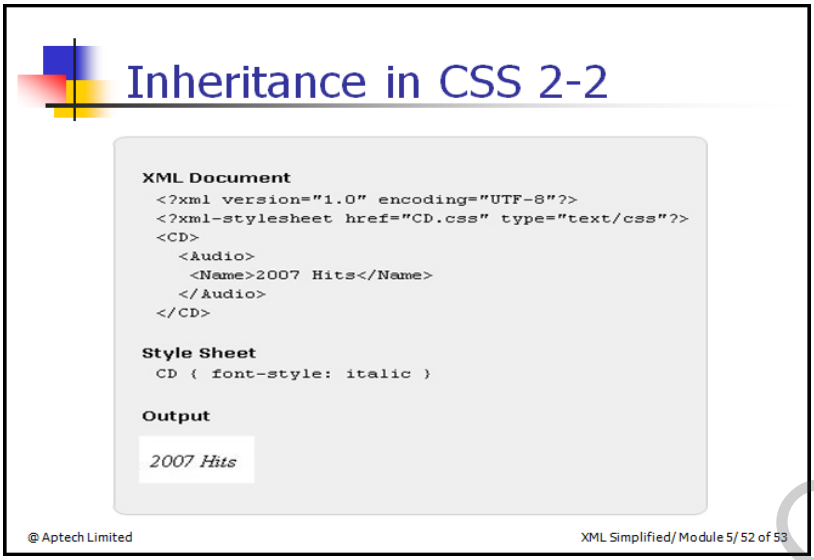
```
#favorite  {...}            /* a=1;b=0;c=0; specificity =
100 */ Name, CD, Artist.caps {...} /* a=0;b=1;c=3;
specificity = 013 */ Artist.caps  {...}          /*
a=0;b=0;c=1; specificity = 001 */
```

6. If two rules have the same weight, then the one specified last wins.

Slides 51 and 52

Let us understand the inheritance in CSS.





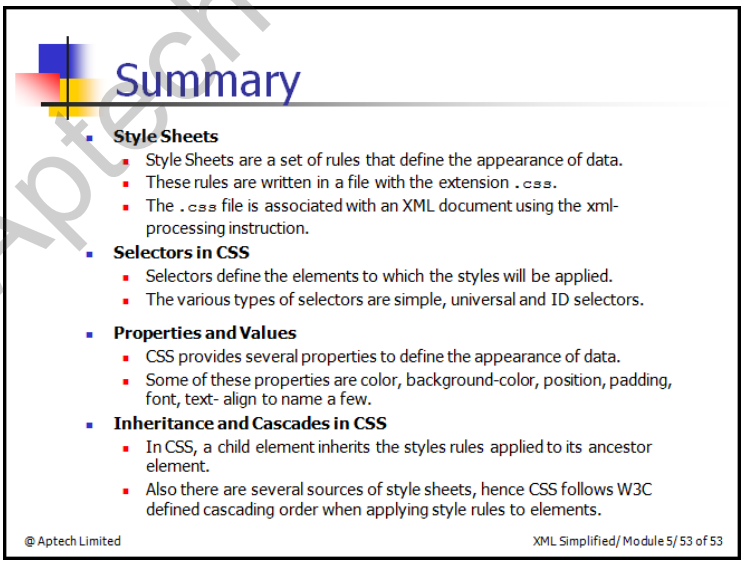
Using slides 51 and 52, explain the inheritance in CSS.

Inheritance is the ability of one entity to acquire the characteristics of another entity. In CSS, the child elements inherit the style rules defined for parent element. However, this is applicable only if the child has no explicit style rule defined for it. Otherwise, the style rule defined for child element overrides the style rule defined for parent element.

Figure on slide 52 shows that the text in element `Name` is displayed in italics. This is because there are no style rules defined for elements `Audio` and `Name`. Hence, the style rule defined for `CD` is inherited by child element `Name`.

Slide 53

Let us summarize the session.



In slide 53, summarize the session. End the session, with a brief summary of what has been taught in the session.

5.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the XSL and XSLT that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 6 - XSL and XSLT

6.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

6.1.1 Objectives

By the end of this session, the learners will be able to:

- Introduction to XSL
- Working with XSL

6.1.2 Teaching Skills

To teach this session, you should be well-versed with XSL and XSLT. You should aware yourself with the techniques of transforming XML documents into XML, HTML, and Text documents using XSL stylesheets.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

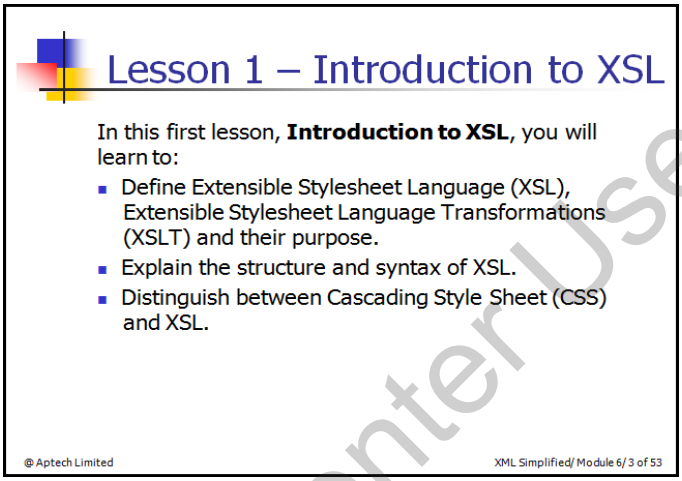
Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

Tell the students that this session introduces the process of transforming XML documents into XML, HTML, and text documents using Extensible Stylesheet Language (XSL). It also explains the difference between XSL and CSS.

6.2 In-Class Explanations

Slide 3

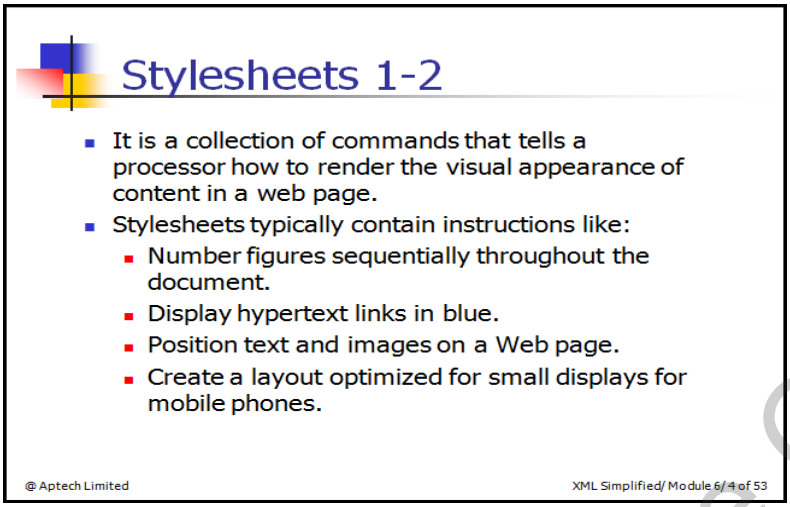
Let us understand the objectives of lesson 1.

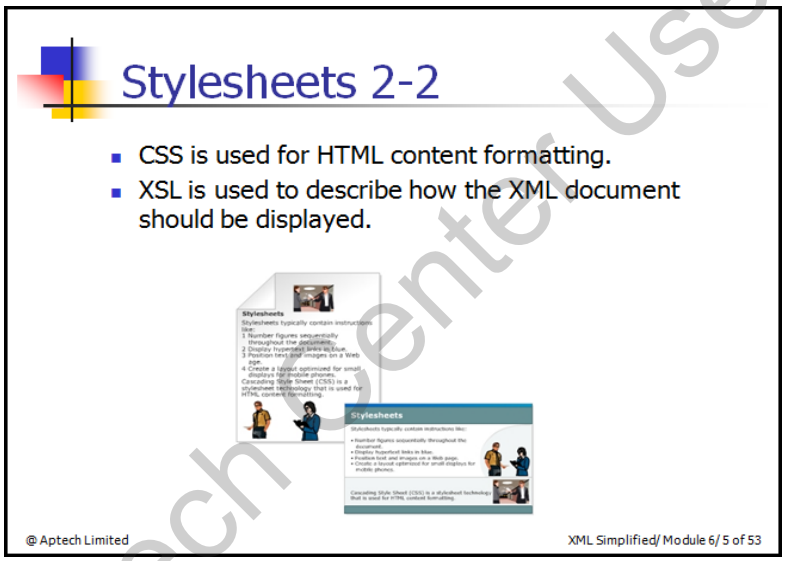


Using slide 3, explain the topics covered in lesson1.

Slides 4 and 5

Let us understand the stylesheets.





Using slides 4 and 5, explain the stylesheets.

A stylesheet is a collection of commands that tells a processor (such as a Web browser, or a document reader) how to render the visual appearance of content in a Web page.

Stylesheets typically contain instructions such as:

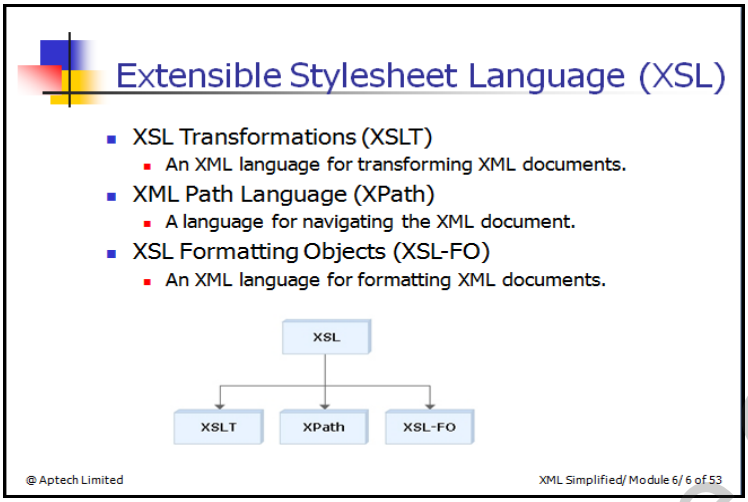
- Number figures sequentially throughout the document.
- Display hypertext links in blue.
- Position text and images on a Web page.
- Create a layout optimized for small displays for mobile phones.

CSS is a stylesheet technology that is used for HTML content formatting.

Extensible Stylesheet Language (XSL) is developed by W3C to describe how the XML document should be displayed.

Slide 6

Let us understand the XSL.



Using slide 6, explain XSL.

XSL is an XML-based language used to create stylesheets. XSL is designed to format XML content for display purposes and also has the ability to completely transform XML documents. XSL is a specification from the World Wide Web Consortium (W3C). XSL Transformation Recommendation describes the process of transforming an XML document, using a transformation engine and XSL. The XML document and XSL stylesheet are provided as input to the XML transformation engine, also known as the XSLT processor. The output of the processor is in the form of a result tree in which the elements of the XML document are converted into nodes having specific text values and attributes. XSLT processors use XSL stylesheets to gather instructions for transforming source XML documents to output XML documents. Stylesheets are used to change the structure of the XML document and modify the output.

XSL consists of three languages as follows:

- XSL Transformations (XSLT)

XSLT transforms an XML document into another XML document that can actually be understood and displayed by a computer.

- XML Path Language (XPath)

XPath is used as the navigator for XSL. XSL uses XPath to find parts of the source document that should match a certain predefined template. When XPath finds the node, XSLT takes over and performs a transformation, turning the source document into a result document.

- XSL Formatting Objects (XSL-FO)

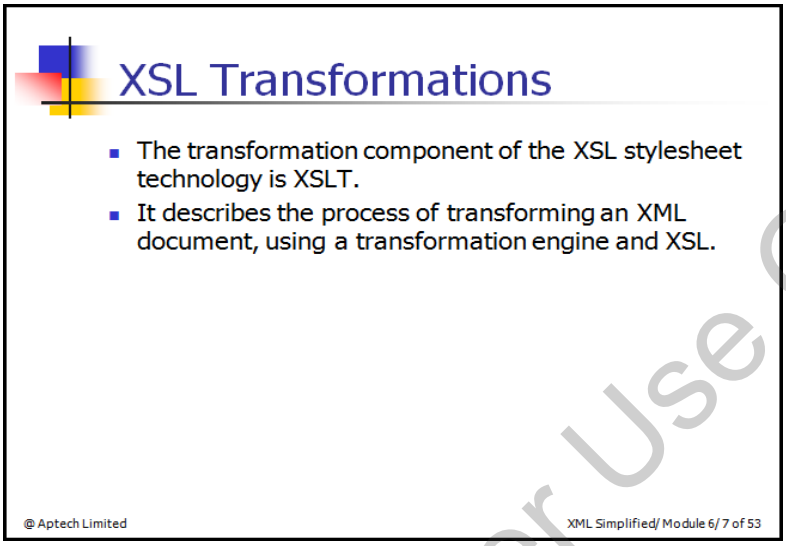
An XML language for formatting XML documents. Once XPath has searched through the source document and used XSLT to transform the source document into the result document, the document then needs to be formatted so that the Web browser will be able

to present the document with the proper layout and structure. Simply put, XSL-FO is the part of XSL that produces the final output.

Figure on slide 6 depicts the structure of XSL.

Slide 7

Let us understand the XSL transformations.

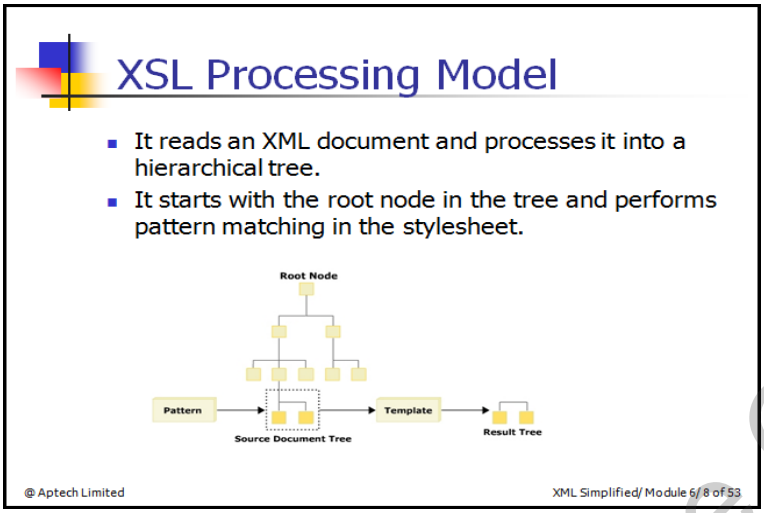


Using slide 7, explain the XSL transformations.

The transformation component of the XSL stylesheet technology is XSLT. Its purpose is to transform XML documents. It describes the process of transforming an XML document, using a transformation engine and XSL. The XML document and XSL stylesheet are provided as input to the XML transformation engine, also known as the XSLT processor. The output of the processor is in the form of a result tree in which the elements of the XML document are converted into nodes having specific text values and attributes.

Slide 8

Let us understand the XSL processing model.



Using slide 8, explain the XSL processing model.

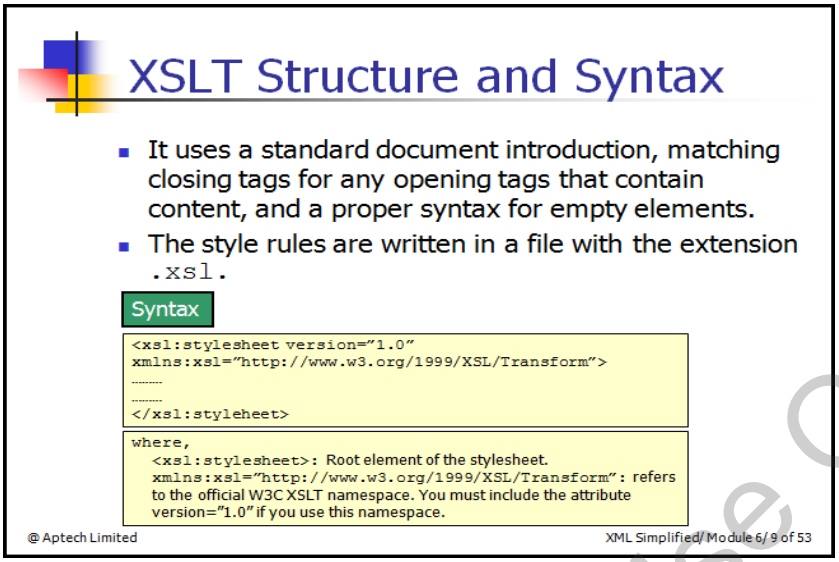
The XML processor reads an XML document and processes it into a hierarchical tree containing nodes for each piece of information in a document. After a document has been processed into a tree, the XSL processor begins applying the rules of an XSL stylesheet to the Document tree.

The XSL processor starts with the root node in the tree and performs pattern matching in the stylesheet. These patterns are stored within constructs known as templates in XSL stylesheets. The XSL processor analyzes templates and the patterns associated with them to process different parts of the Document tree. When a match is made, the portion of the tree matching the given pattern is processed by the appropriate stylesheet template. At this point, the rules of the template are applied to the content to generate a result tree. The result tree is itself a tree of data and the data has been transformed by the stylesheet.

Figure on slide 8 displays the XSL processing model.

Slide 9

Let us understand the XSLT structure and syntax.



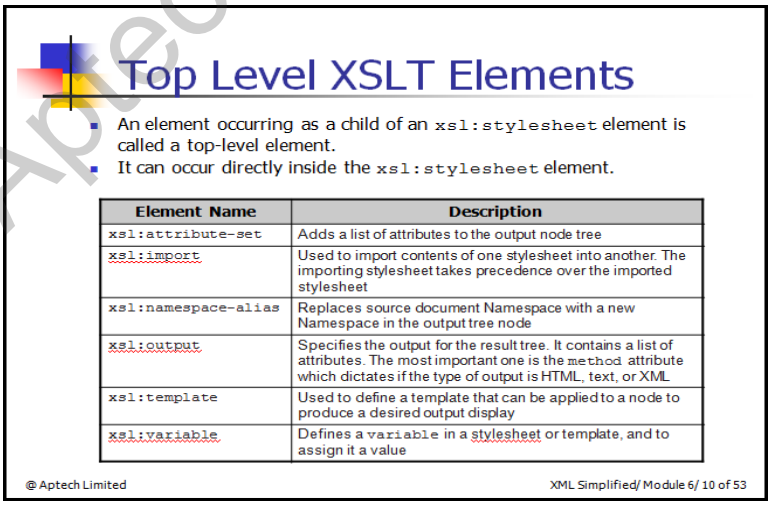
Using slide 9, explain the XSLT structure and syntax.

XSLT follows normal rules of XML syntax. It uses a standard document introduction, matching closing tags for any opening tags that contain content, and a proper syntax for empty elements.

In XSL, the style rules are written in a file with the extension `.xsl`. This file is associated with an XML document by using the statement `<?xml-stylesheet href="xsl file" type="text/xsl"?>`. Here, the xml-stylesheet is the processing instruction.

Slide 10

Let us understand the top-level XSLT elements.



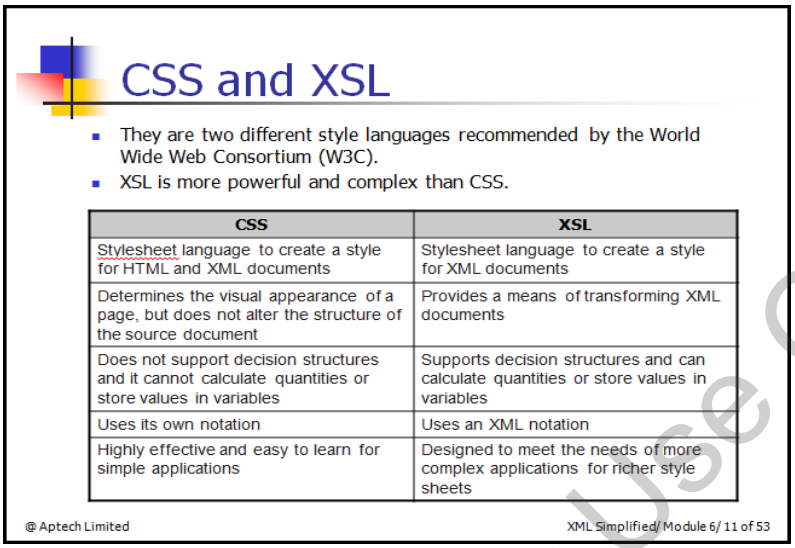
Using slide 10, explain the top-level XSLT elements.

The top-level XSLT elements can occur directly inside the `xsl:stylesheet` element. An element occurring as a child of an `xsl:stylesheet` element is called a top-level element. These elements provide the building blocks for creating XSLT documents.

The top-level elements in the XSLT syntax are listed in table on slide 10. These elements can occur directly inside the `xsl:stylesheet` or `xsl:transform` elements.

Slide 11

Let us understand the CSS and XSL.

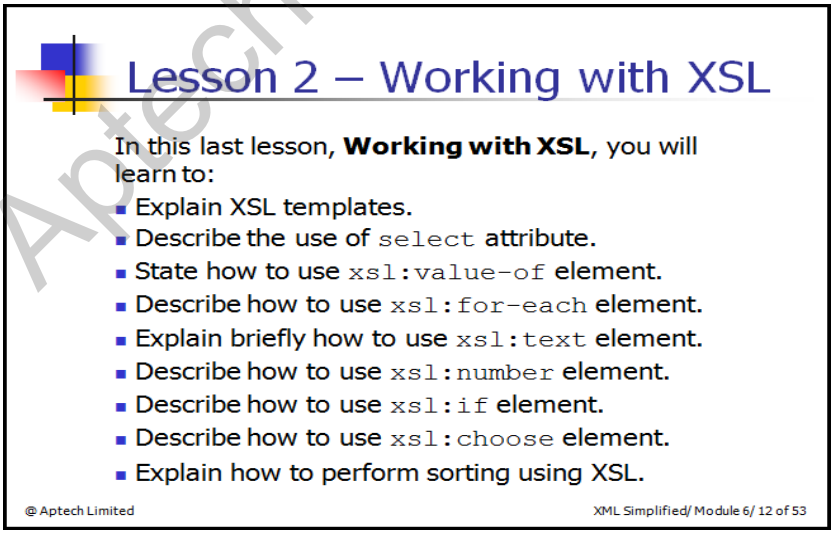


Using slide 11, explain the CSS and XSL.

XSL and CSS are two different style languages recommended by the W3C. XSL is more powerful and complex than CSS. Some of the differences between CSS and XSL are listed in table on slide 11.

Slide 12

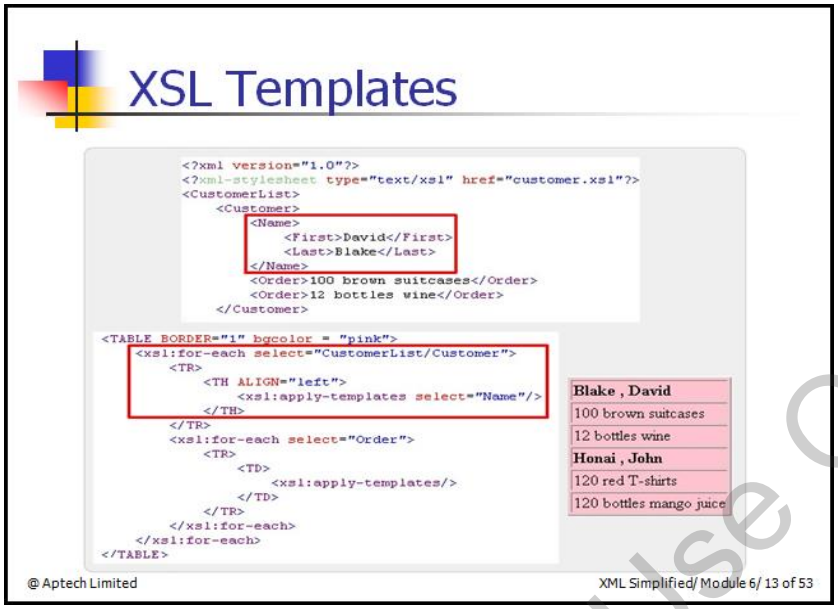
Let us understand the objectives of lesson 2.



Using slide 12, explain the topics covered in lesson 2.

Slide 13

Let us understand the XSL templates.



Using slide 13, explain the XSL templates in details.

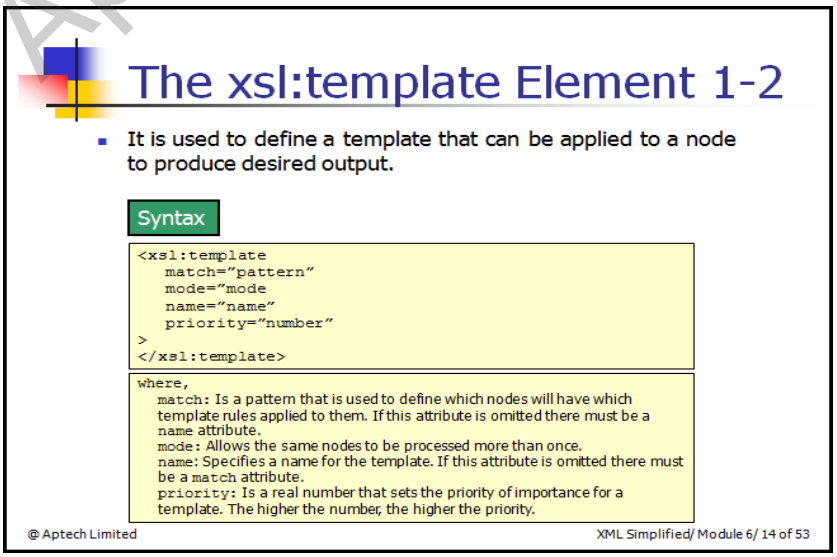
A template is the main component of a stylesheet. Templates are defined with the help of rules. A template rule is used to control the output of the XSLT processor. It defines the method by which an XML element node is transformed into an XSL element node. A template rule consists of a pattern that identifies the XML node and an action that selects and transforms the node.

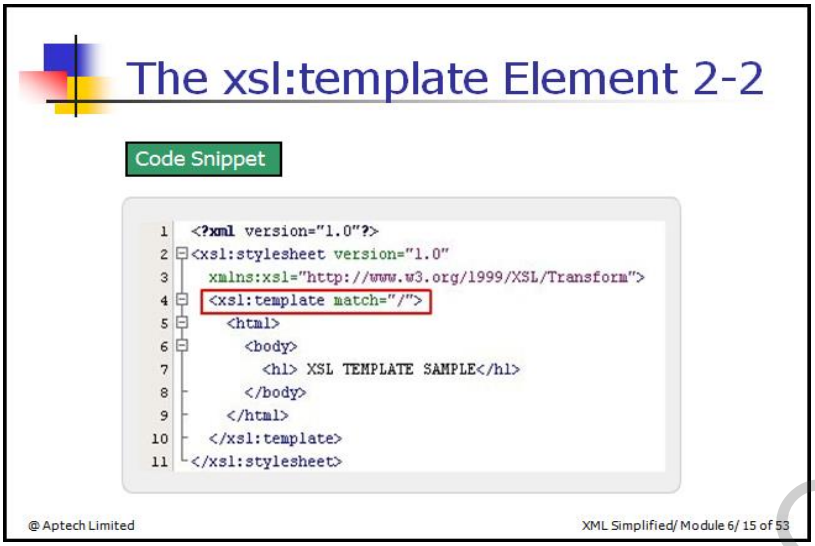
Each template rule is represented by the `xsl:template` element. The `xsl:template` is an element that defines an action for producing output from a source document.

Explain the figure on slide 13 shows an example of XSL template.

Slides 14 and 15

Let us understand the `xsl:template` element.





Using slides 14 and 15, explain the `xsl:template` element.

The `xsl:template` element is used to define a template that can be applied to a node to produce desired output.

The match attribute in `xsl:template` is used to associate the template with an XML element. You can also define a template for a whole branch of the XML document by using the match attribute (for example, `match="/"` defines the whole XML document).

Explain the syntax of the `xsl:template` element using slide 14.

Explain the code snippet which demonstrates the usage of `xsl:template` element in stylesheets on slide 15.

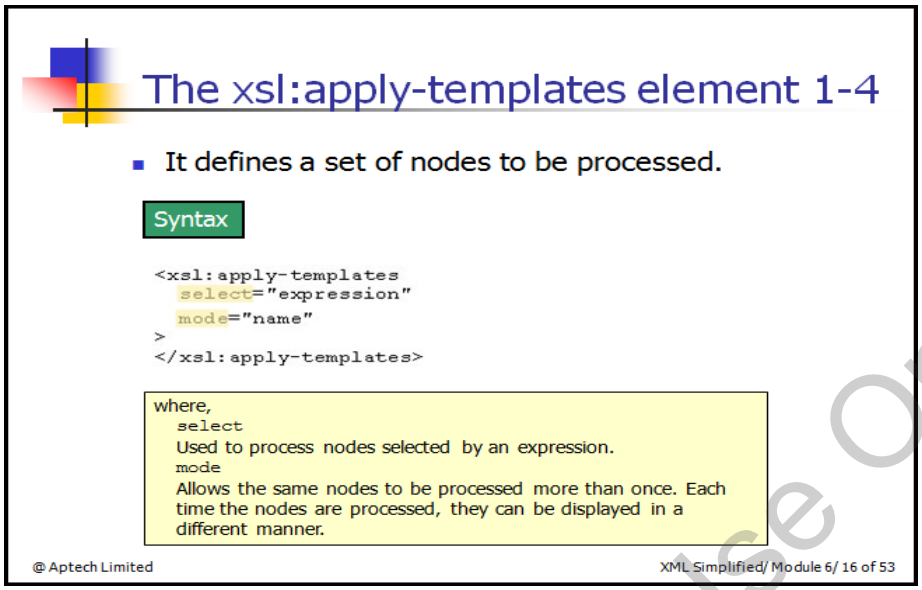
An XSL stylesheet is an XML document itself, it always begins with the XML declaration : `<?xml version="1.0" ?>`. The next element, `<xsl:stylesheet>`, defines that this document is an XSLT stylesheet document. It also contains the version number and XSLT namespace attributes.

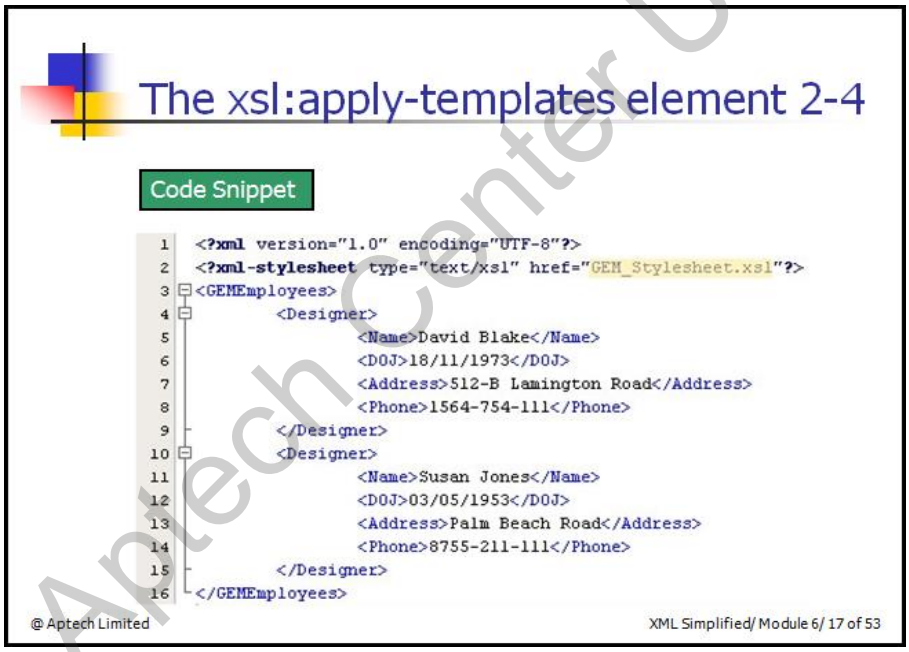
The `<xsl:template>` element defines a template. The `match="/"` attribute associates the template with the root of the XML source document.

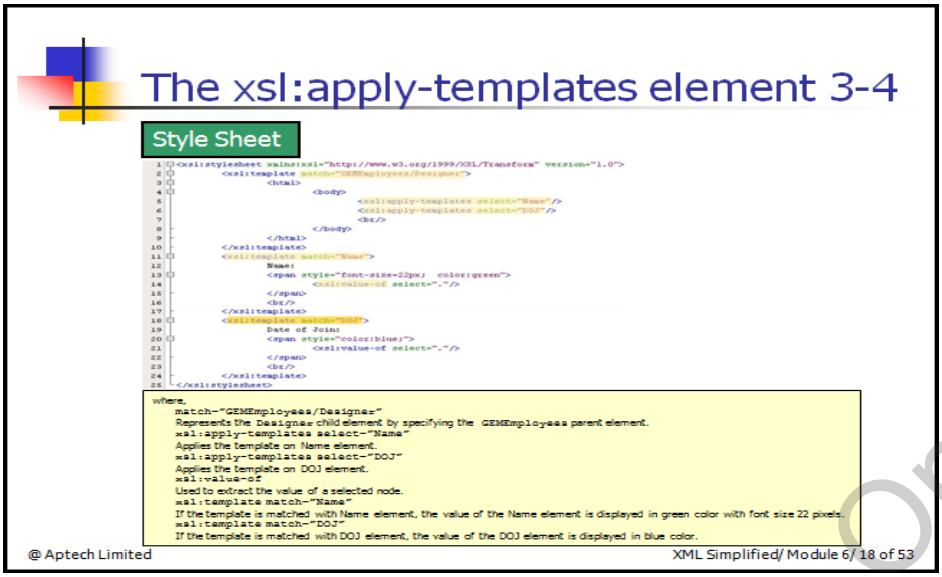
The content inside the `<xsl:template>` element defines some HTML to write the output. The last two lines define the end of the template and the end of the stylesheet.

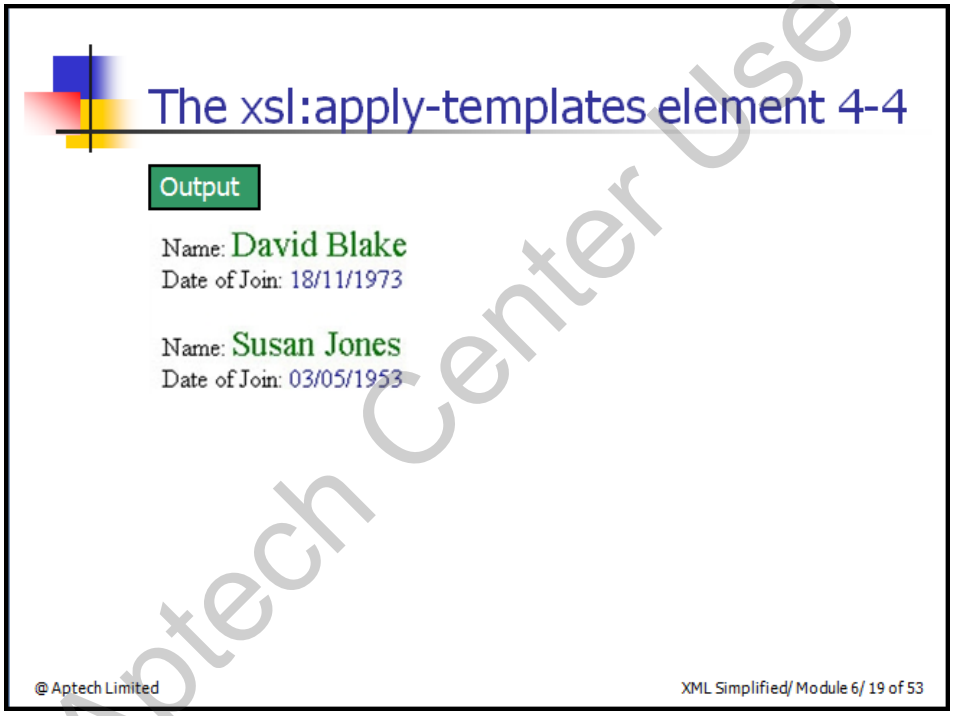
Slides 16 to 19

Let us understand the `xsl:apply-templates` element.









Using slides 16 to 19, explain the `xsl:apply-templates` element.

The `xsl:apply-templates` element defines a set of nodes to be processed. This element, by default, selects all child nodes of the current node being processed, and finds a matching template rule to apply to each node in the set.

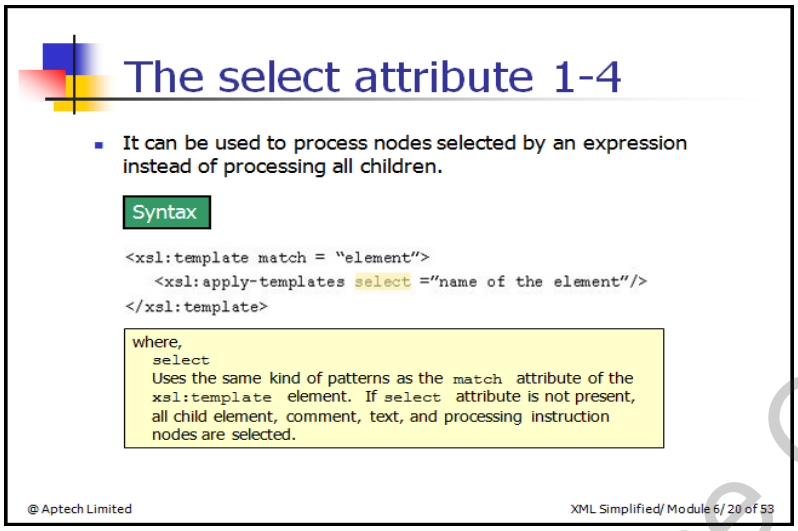
Explain the figure on slide 16 that shows the syntax for `xsl:apply-templates`.

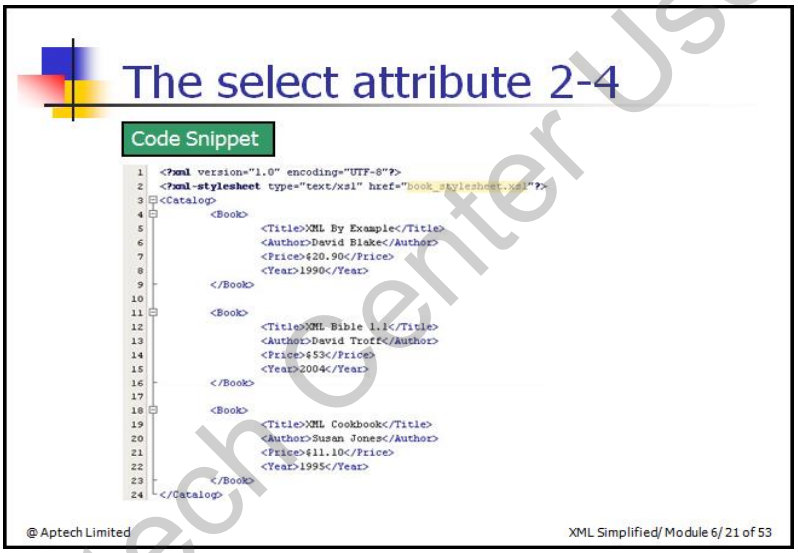
Explain the figure on slide 17 that shows the code for style rules that are defined in `GEM_Stylesheet.xsl` file.

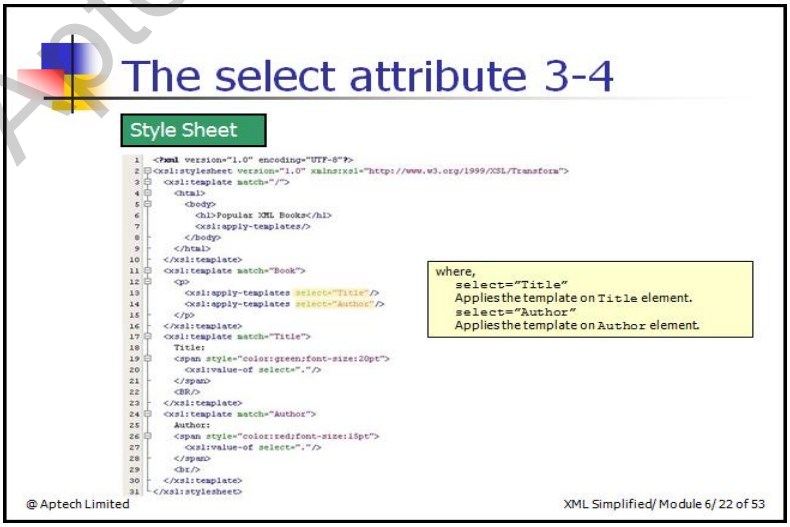
Explain the figure on slide 18 depicts the `GEM_Stylesheet.xsl` file.

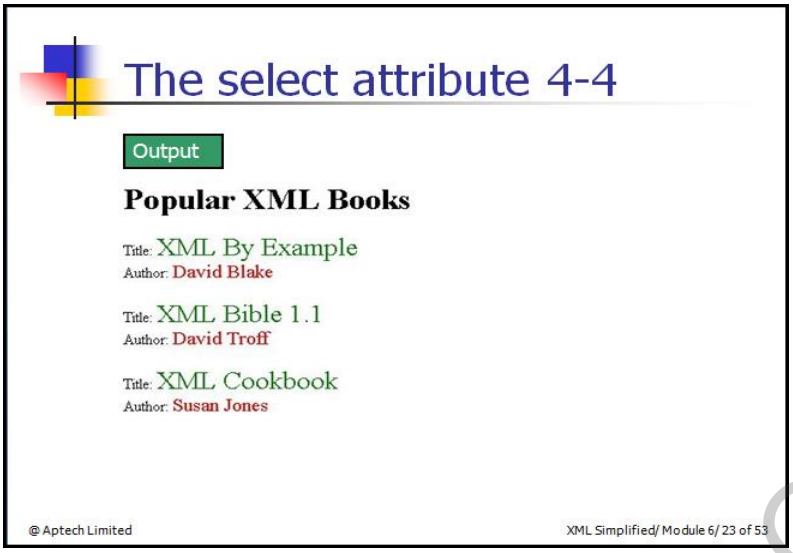
Slides 20 to 23

Let us understand the `select` attribute.









Using slide 20, explain the `select` attribute.

The `select` attribute can be used to process nodes selected by an expression instead of processing all children. The `xsl:apply-templates` with a `select` attribute can be used to choose a particular set of children instead of all children.

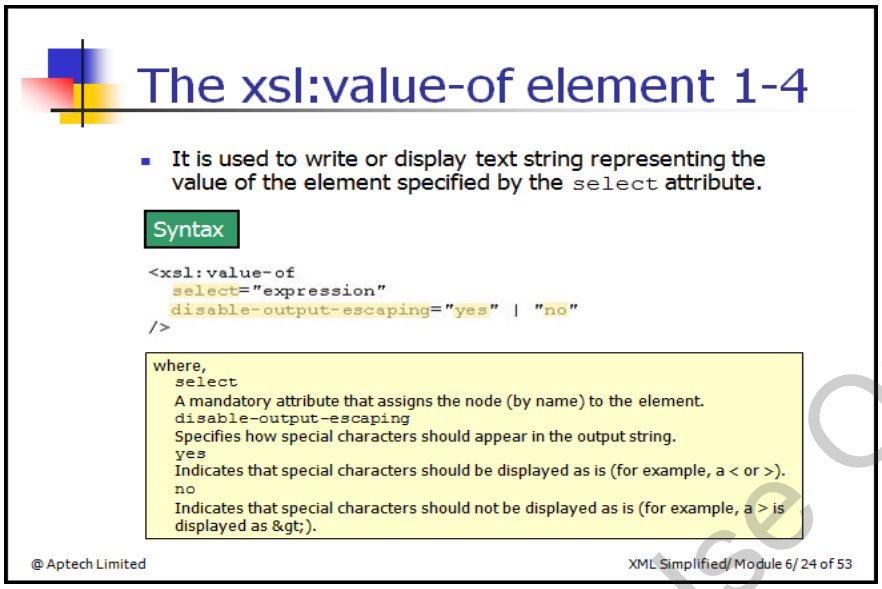
Explain the figure on slide 20 shows the syntax for `select` attribute.

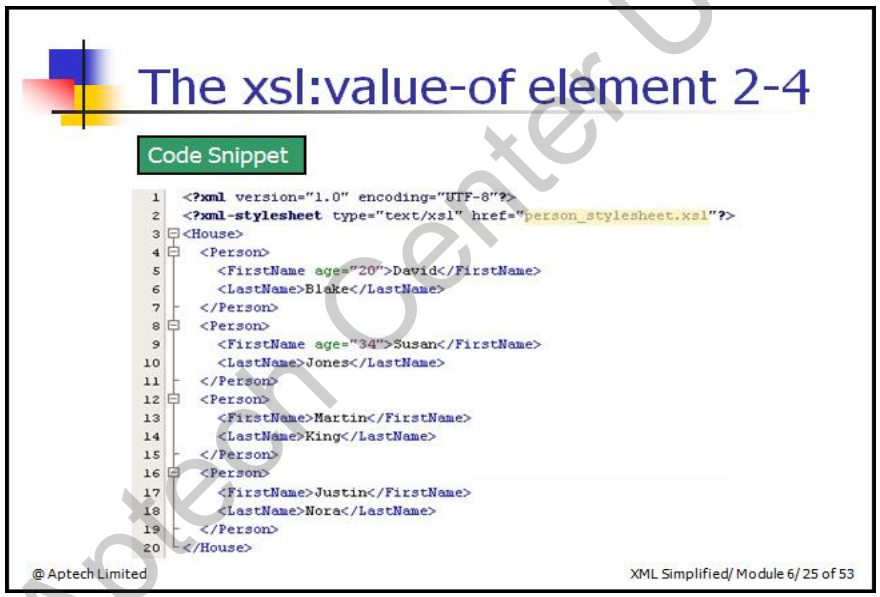
It uses the same kind of patterns as the `match` attribute of the `xsl:template` element. If `select` attribute is not present, all child element, comment, text, and processing instruction nodes are selected.

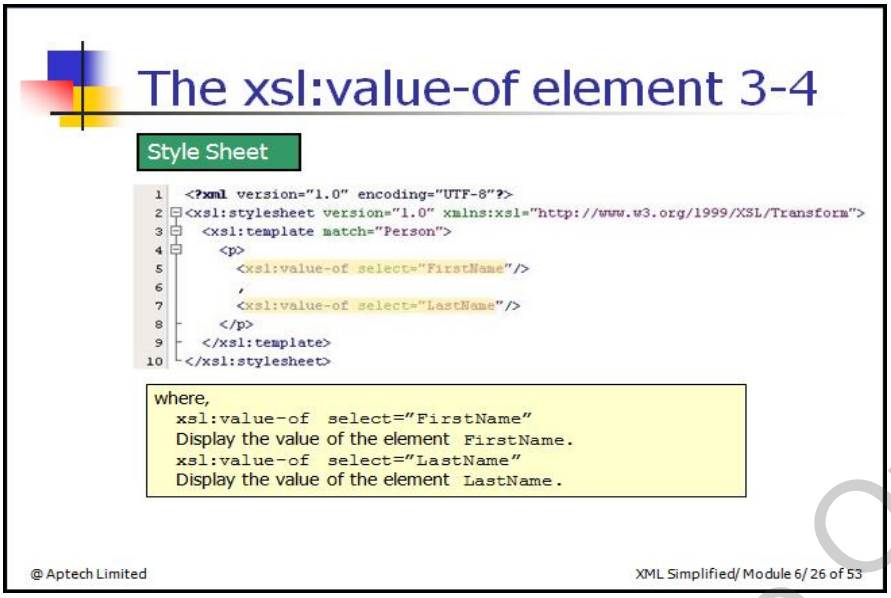
Explain the figure on slide 21 shows the code for style rules that are defined in `book_stylesheet.xsl` file. Also the figure on slide 22 that depicts the stylesheet for `book_stylesheet.xsl` file. Explain the code.

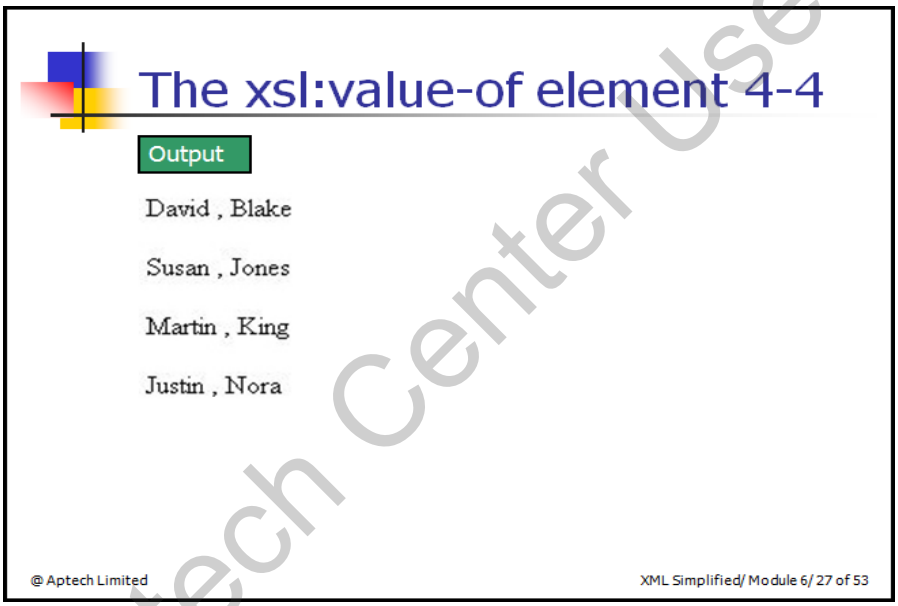
Slides 24 to 27

Let us understand the `xsl:value-of` element.









Using slides 24 to 27, explain the `xsl:value-of` element.

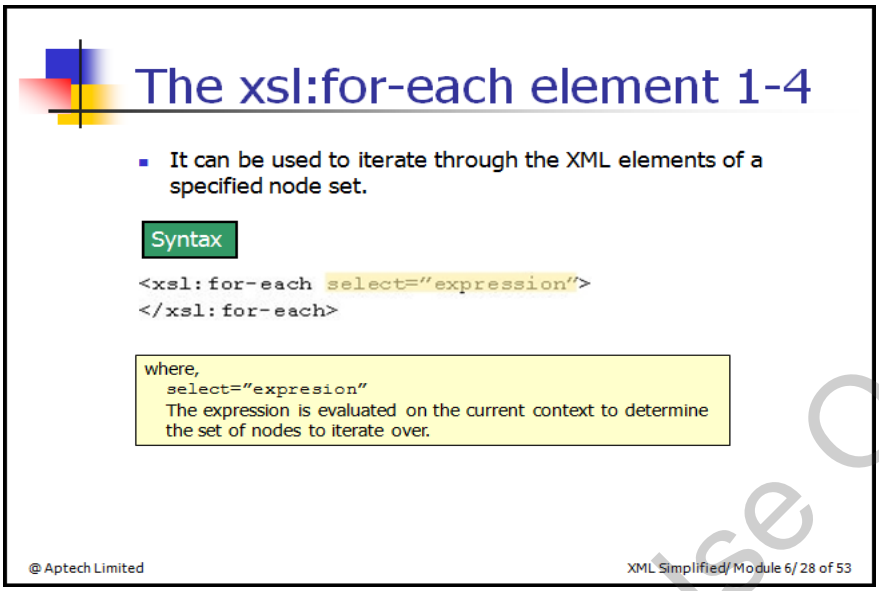
The `xsl:value-of` element is used to write or display in the result tree a text string representing the value of the element specified by the `select` attribute. A `xsl:value-of` element can only have one node assigned to it.

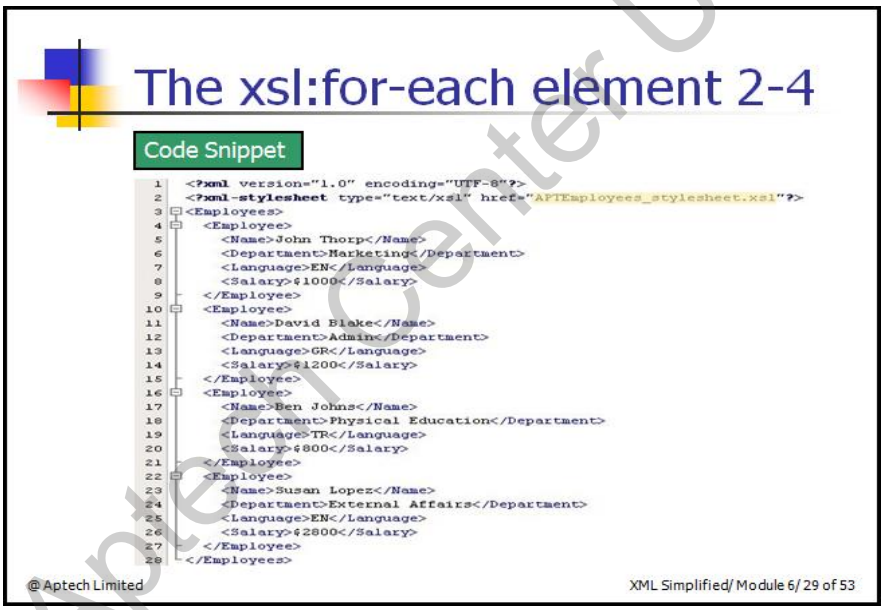
Explain the figure on slide 24 that shows the syntax for `xsl:value-of` element.

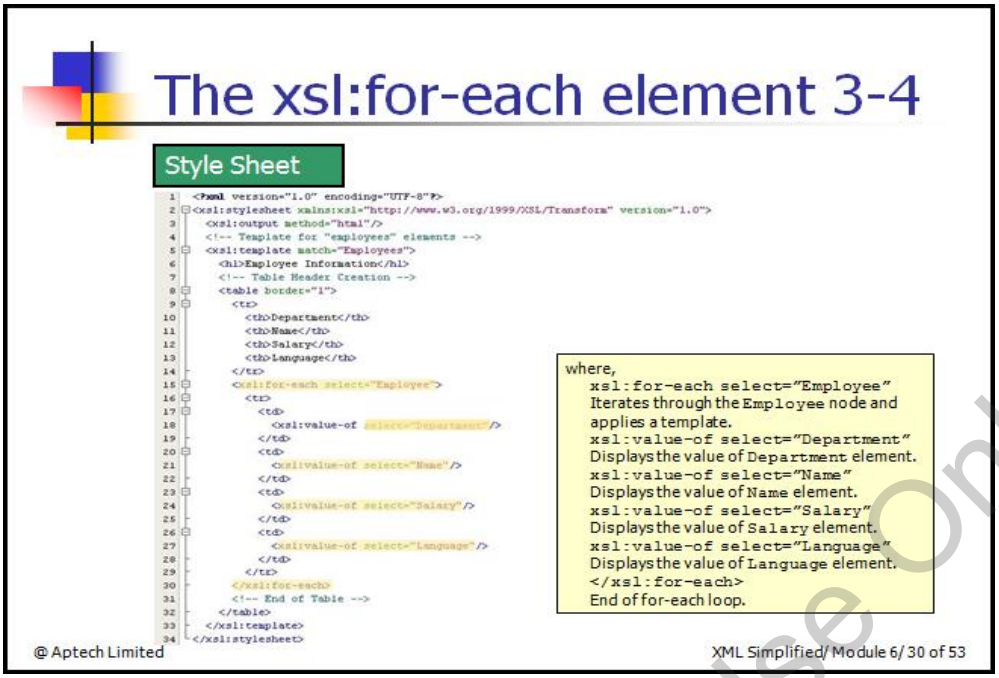
Explain the figure on slide 25 shows the code for style rules that are defined in `person_stylesheet.xsl` file. Explain the figure on slide 26 depicts the stylesheet for `person_stylesheet.xsl` file.

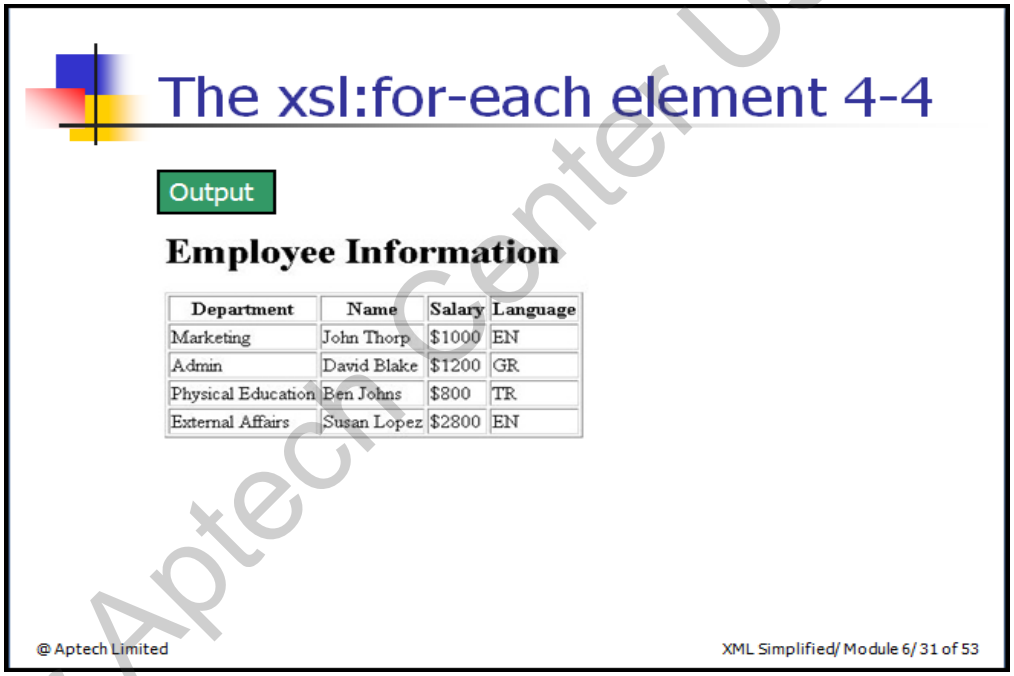
Slides 28 to 31

Let us understand the `xsl:for-each` element.









Using slides 28 to 31, explain the `xsl:for-each` element.

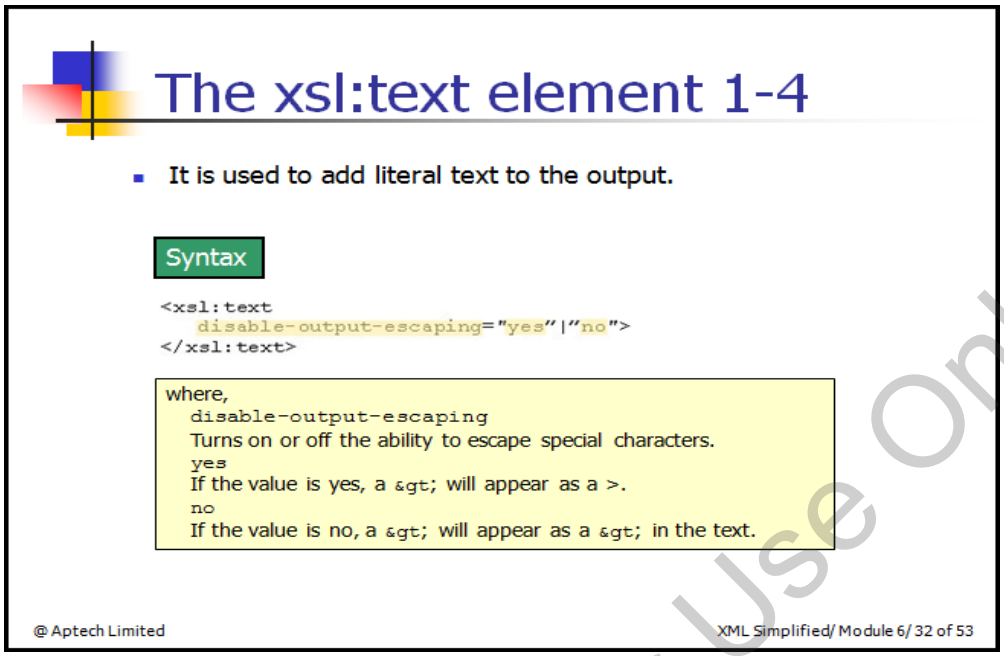
The `xsl:for-each` element can be used to iterate through the XML elements of a specified node set. It applies a template repeatedly to each node in a set.

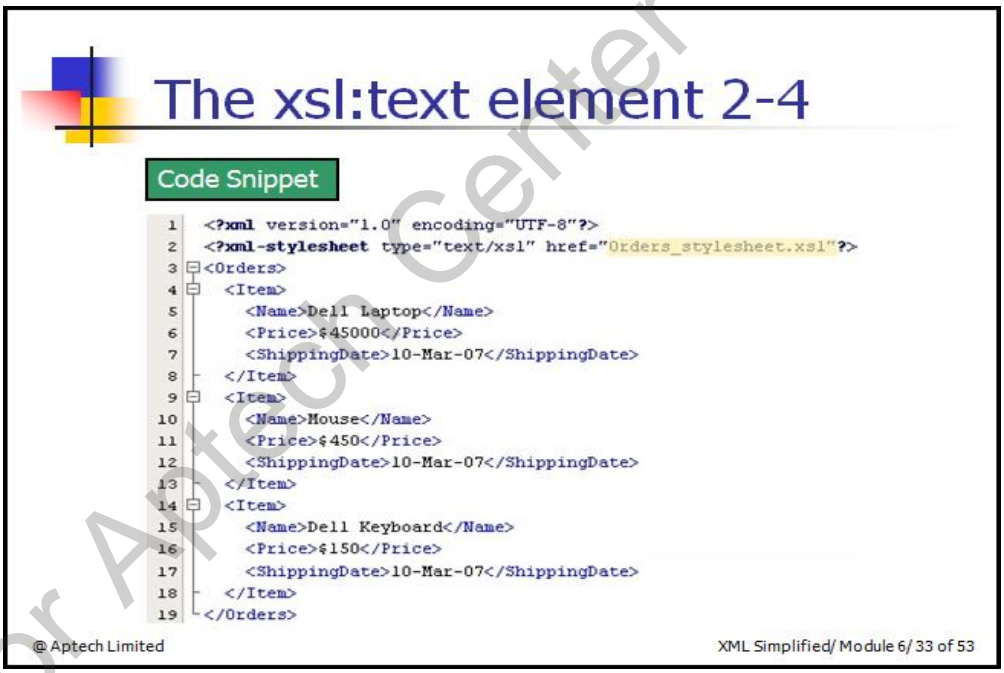
Explain the figure on slide 28 that shows the syntax for `xsl:for-each` element. Mention the expression is evaluated on the current context to determine the set of nodes to iterate over.

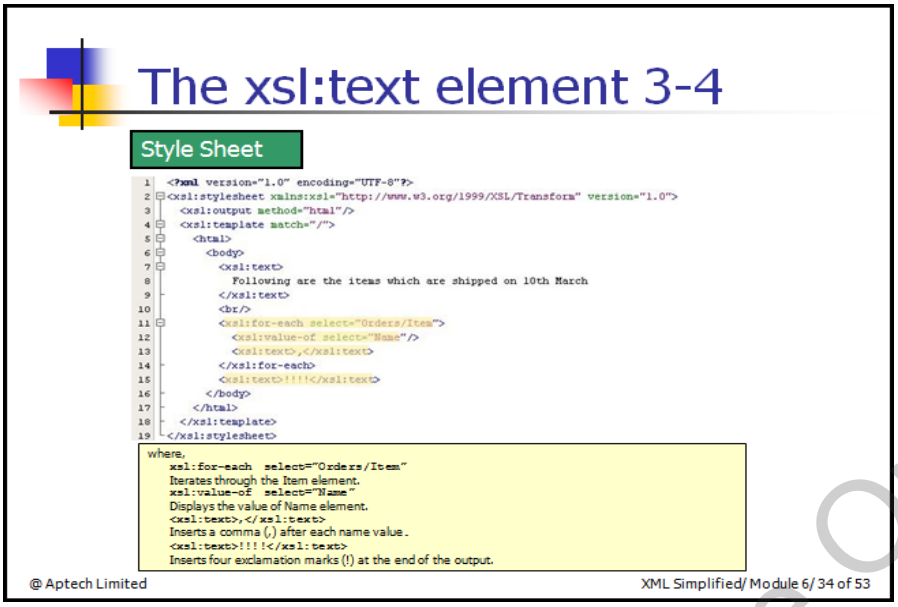
Explain the figure on slide 29 shows the code for style rules that are defined in `APTEmployees_stylesheet.xsl` file. Explain the figure on slide 30 that depicts the stylesheet for `APTEmployees_stylesheet.xsl` file.

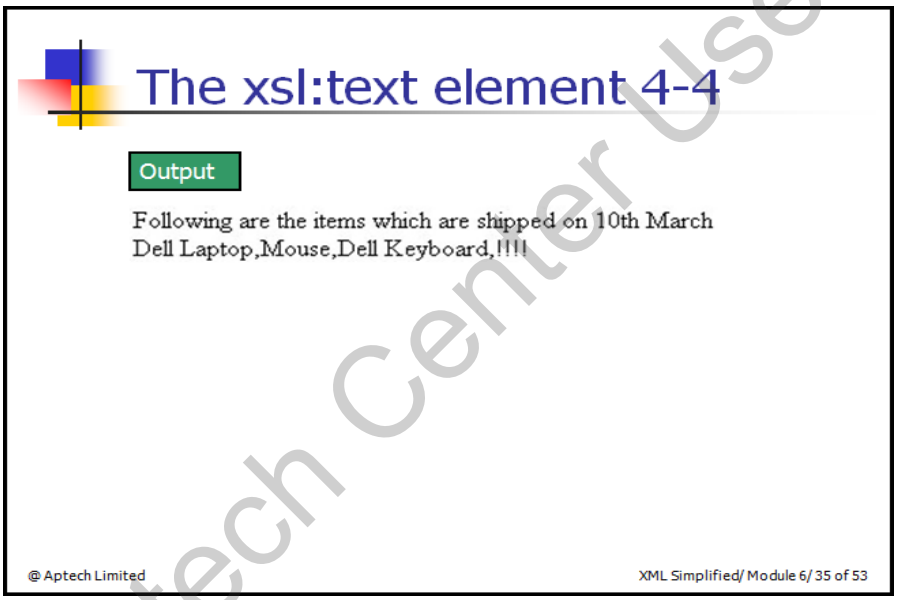
Slides 32 to 35

Let us understand the `xsl:text` element.









Using slides 32 to 35, explain the `xsl:text` element.

The `xsl:text` element is used to add literal text to the output. This element cannot contain any other XSL elements. It can contain only text.

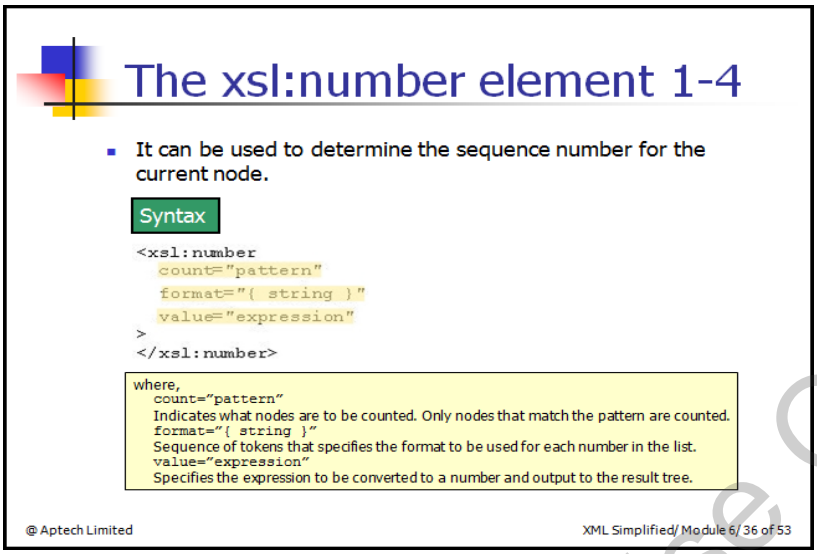
Explain the figure on slide 32 that shows the syntax for `xsl:text` element.

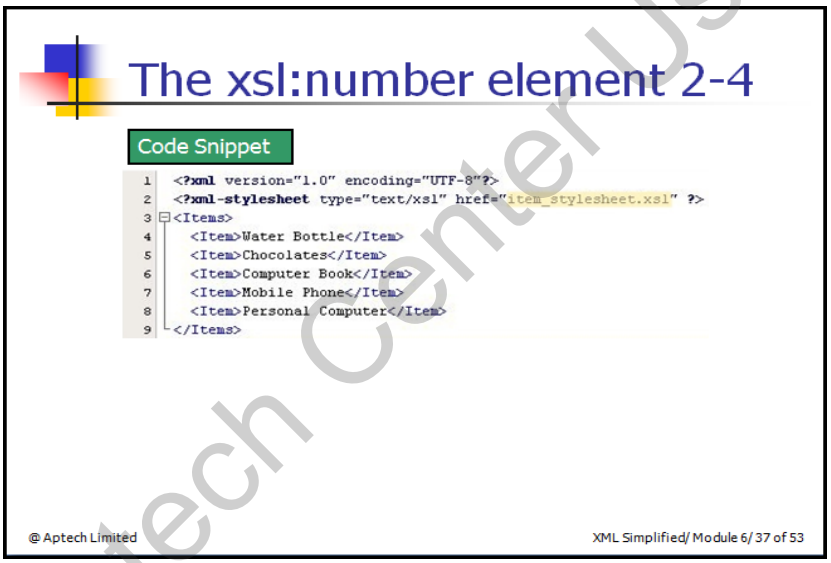
Explain the figure on slide 33 that shows the code for style rules that are defined in `Orders_stylesheet.xsl` file.

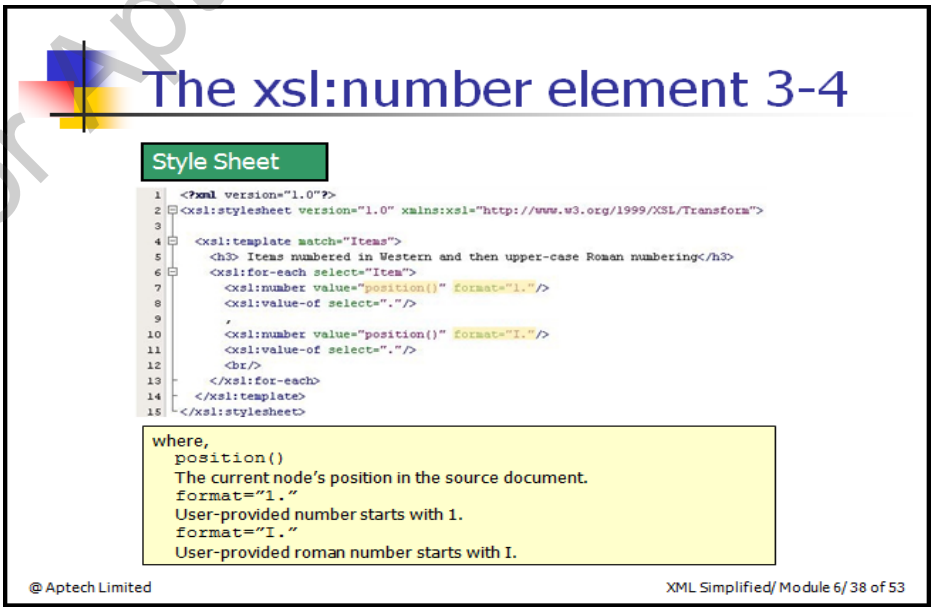
Explain the figure on slide 34 depicts the stylesheet for `Orders_stylesheet.xsl` file.

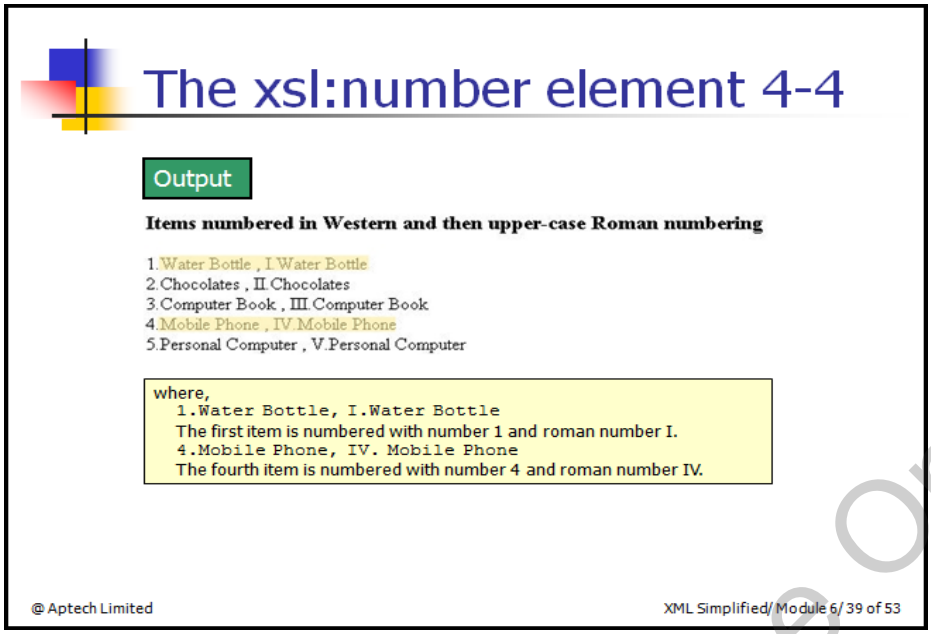
Slides 36 to 39

Let us understand the `xsl:number` element.









Using slides 36 to 39, explain the `xsl:number` element.

The `xsl:number` element can be used to determine the sequence number for the current node. It can also be used to format a number for display in the output.

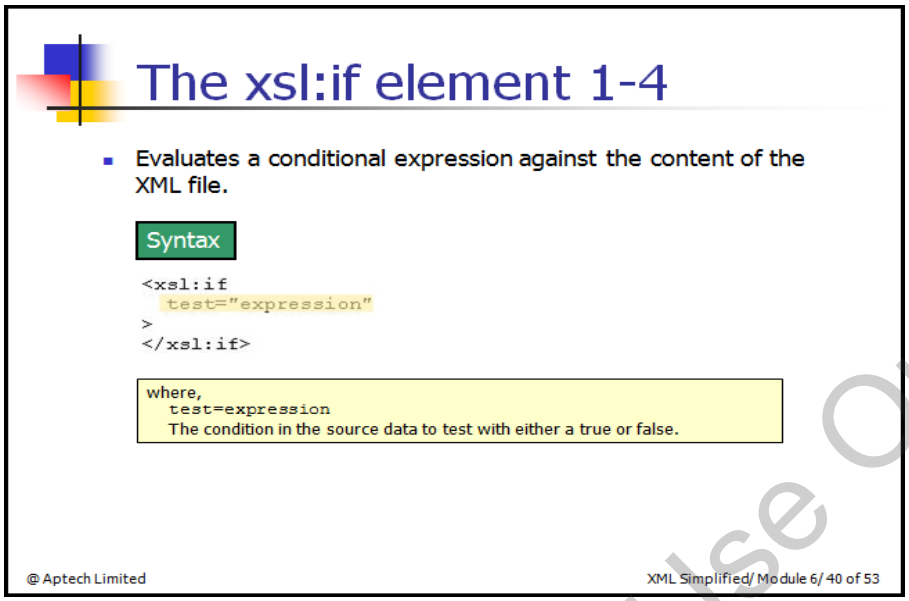
Explain the figure on slide 36 shows the syntax for `xsl:number` element.

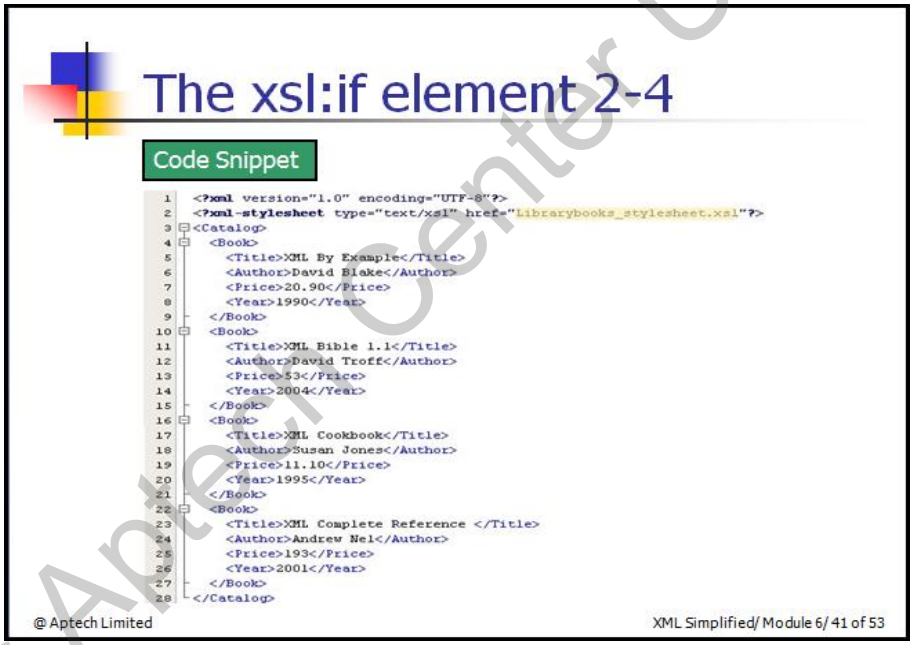
Explain the figure on slide 37 shows the code for style rules that are defined in `item_stylesheet.xsl` file.

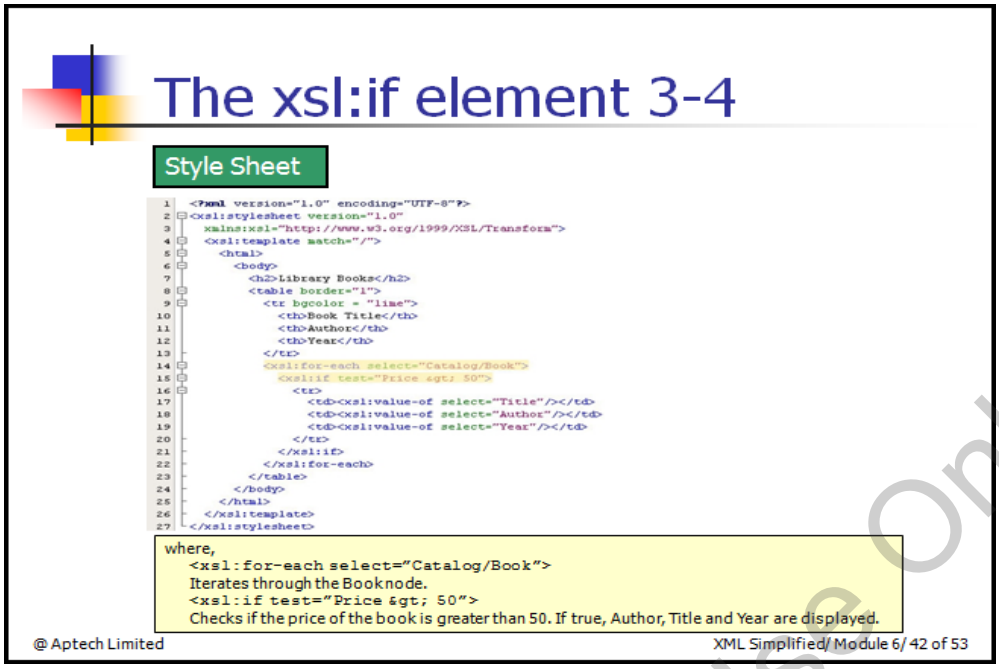
Explain the figure on slide 38 that depicts the stylesheet for `item_stylesheet.xsl` file. Explain the code snippet.

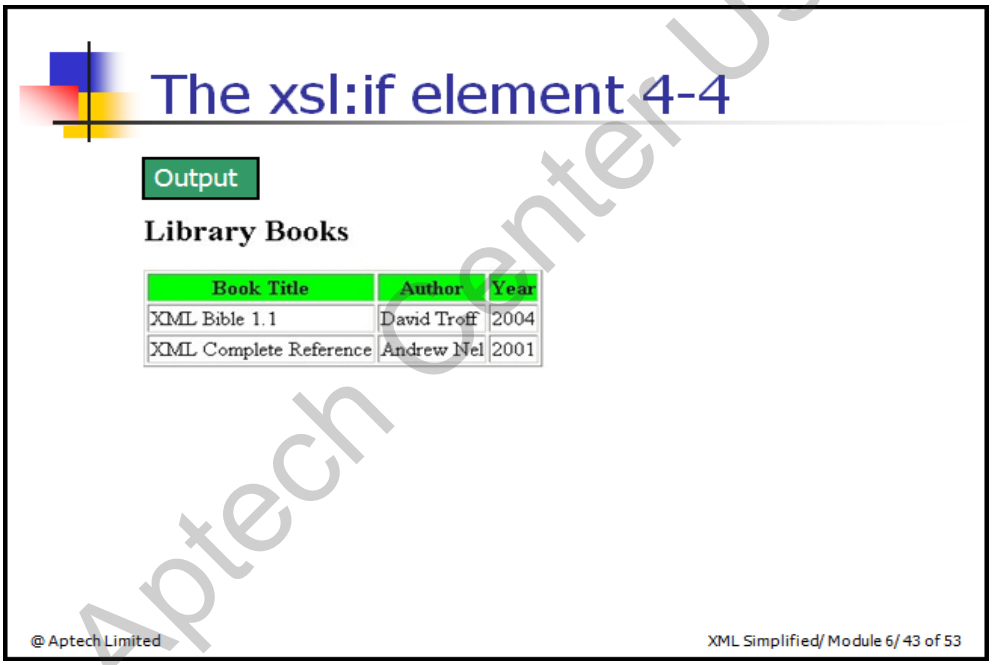
Slides 40 to 43

Let us understand the `xsl:if` element.









Using slides 40 to 43, explain the `xsl:if` element.

The `xsl:if` element evaluates a conditional expression against the content of the XML file. The `test` attribute of `xsl:if` element contains a conditional expression that evaluates to a boolean value of **true** or **false**.

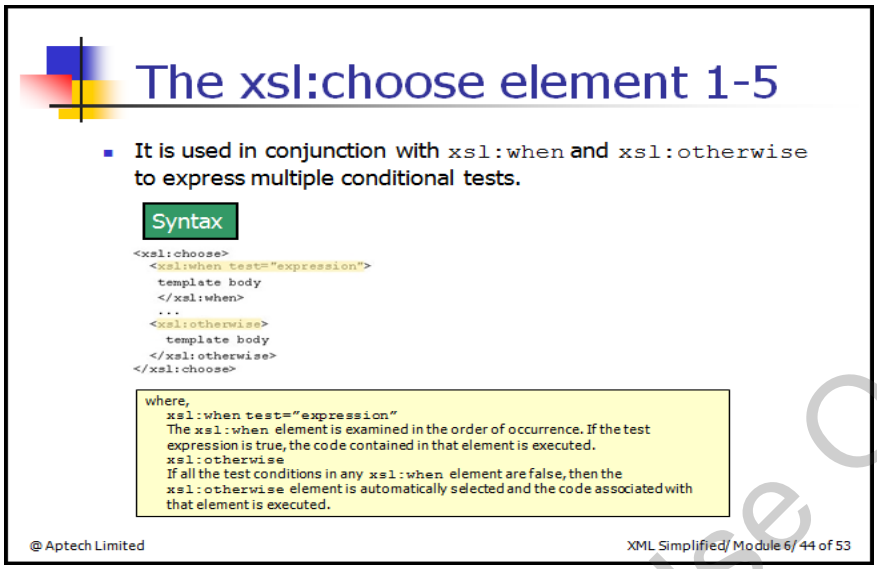
Explain the figure on slide 40 shows the syntax for `xsl:if` element.

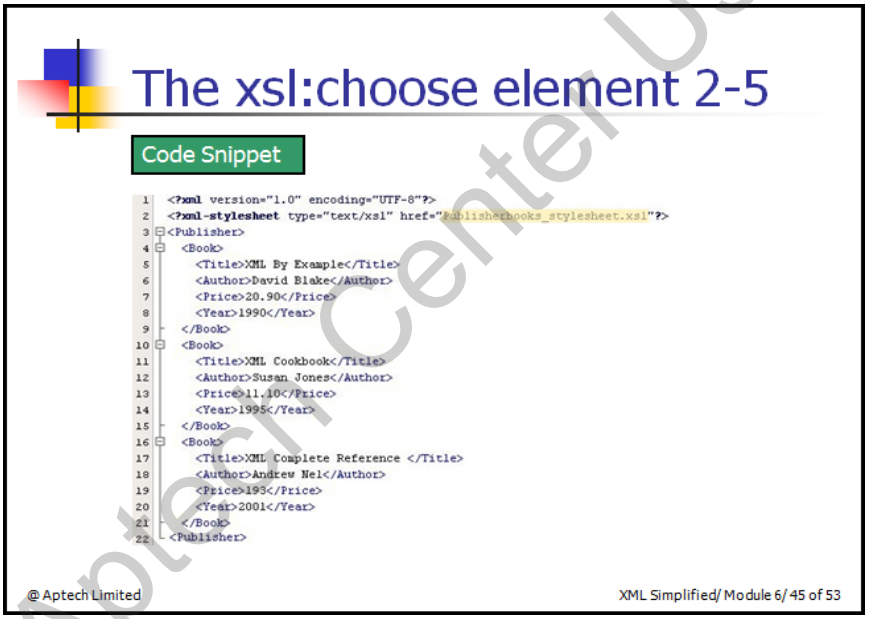
Explain the figure on slide 41 shows the code for style rules are that defined in `Librarybooks_stylesheet.xsl` file.

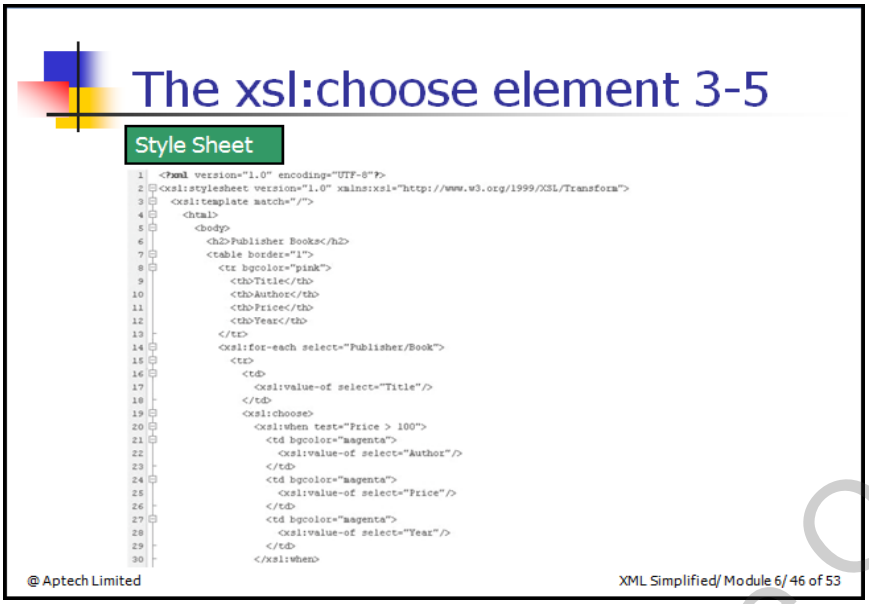
Explain the figure on slide 42 depicts the stylesheet for `Librarybooks_stylesheet.xsl` file.

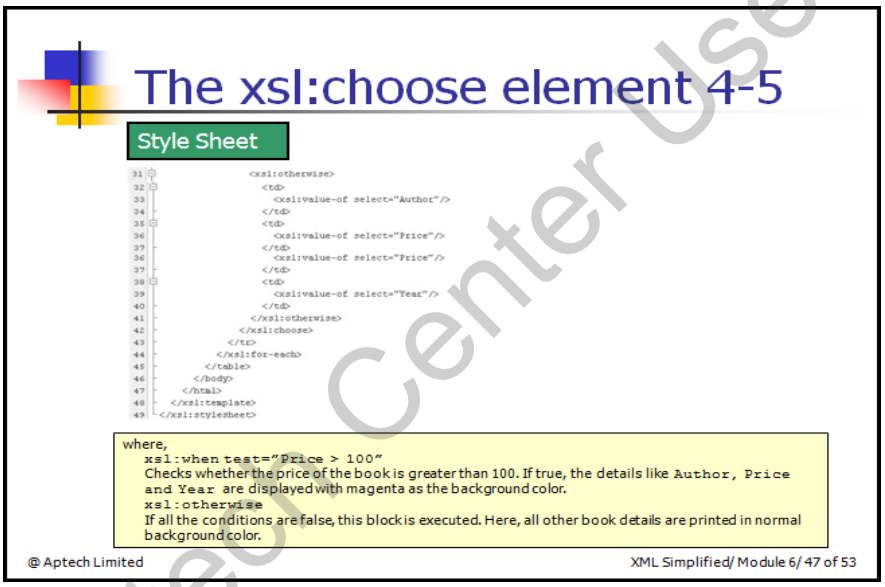
Slides 44 to 48

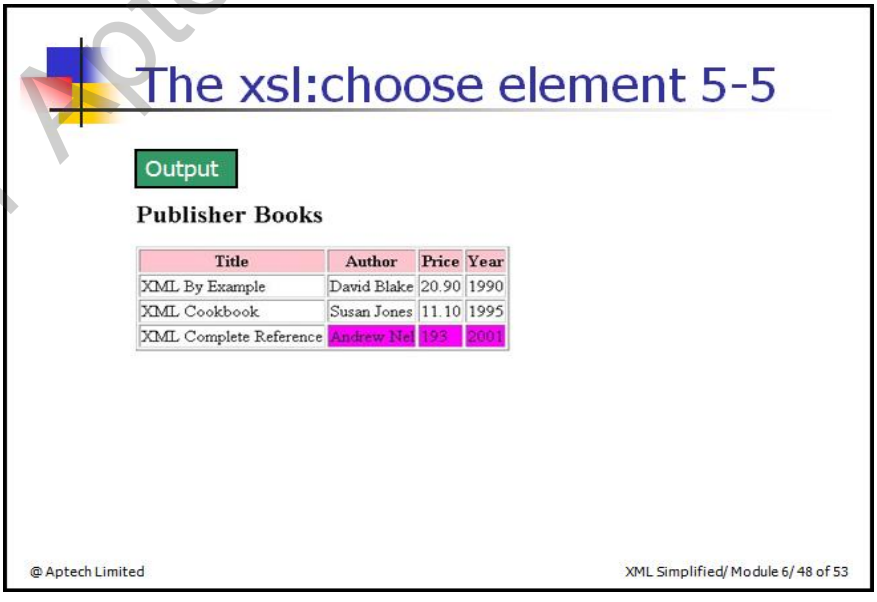
Let us understand the `xsl:choose` element.











Using slides 44 to 48, explain the `xsl:choose` element.

The `xsl:choose` element is used to make a decision when there are two or more possible courses of action. The `xsl:choose` element is used in conjunction with `xsl:when` and `xsl:otherwise` to express multiple conditional tests.

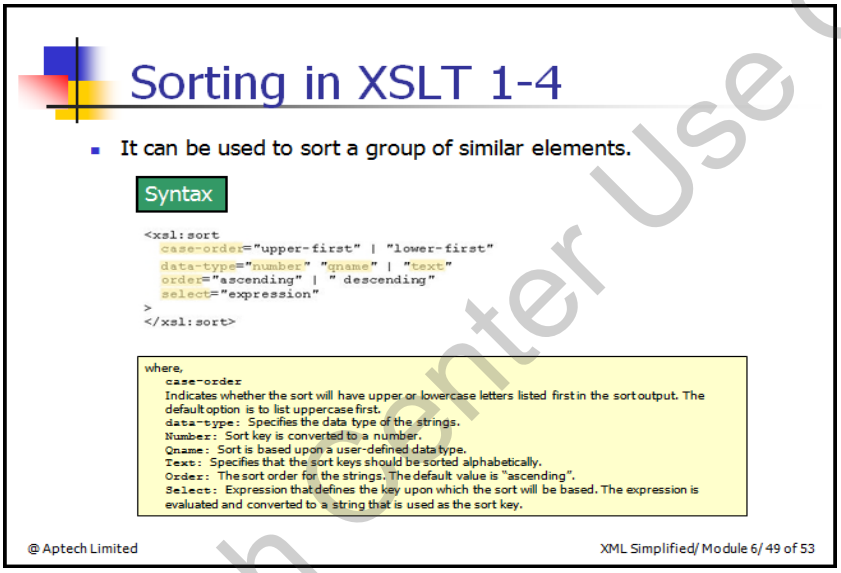
Explain the figure on slide 44 shows the syntax for `xsl:choose` element.

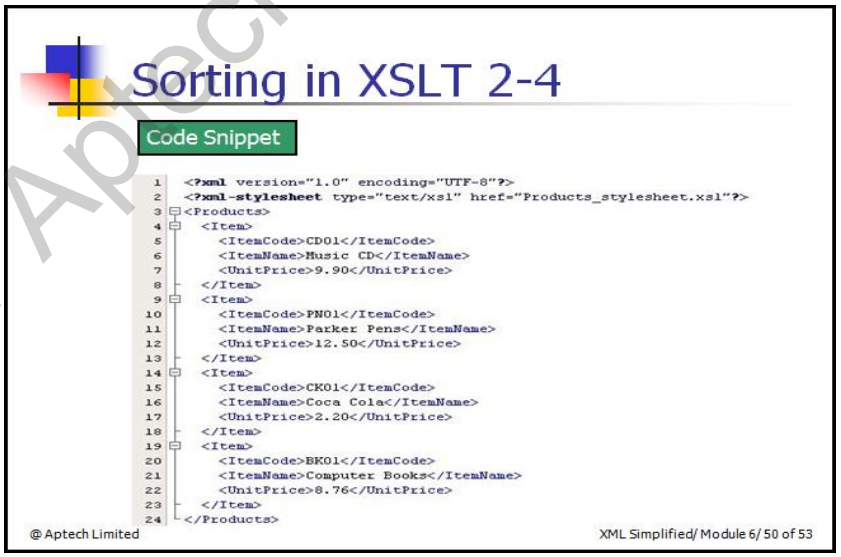
Explain the figure on slides 45 and 46 shows the code for style rules that are defined in `Publisherbooks_stylesheet.xsl` file.

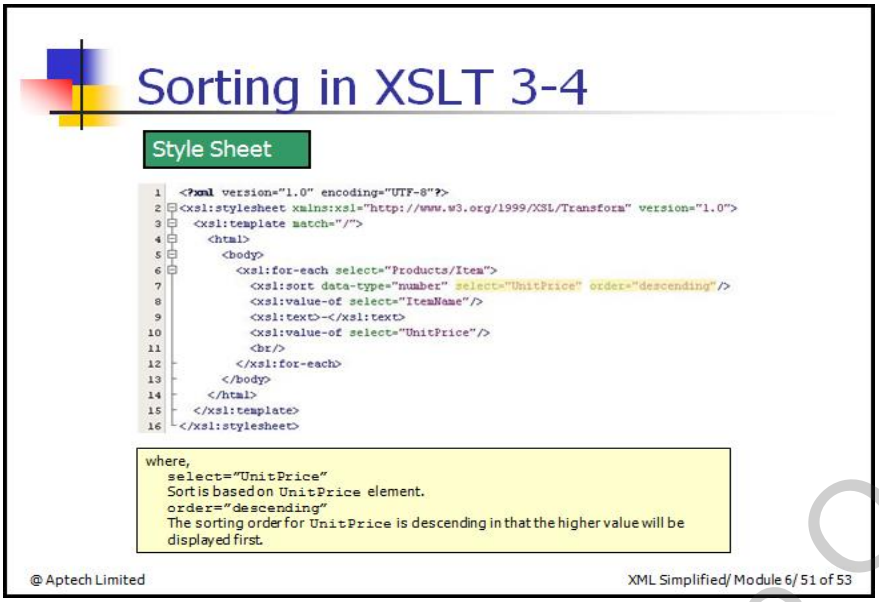
Then, explain the figure on slide 47 depicts the stylesheet for `Publisherbooks_stylesheet.xsl` file.

Slides 49 to 52

Let us understand the sorting in XSLT.







Using slides 49 to 52, explain the sorting in XSLT.

The `xsl:sort` element in XSLT can be used to sort a group of similar elements. The sorting can be done in various ways by using the attributes of this element.

Explain the figure on slide 49 that shows the syntax for `xsl:sort` element.

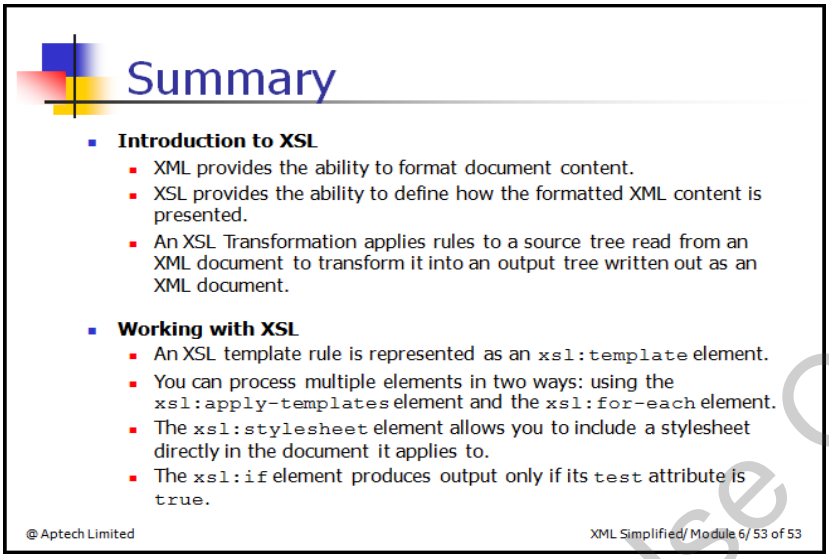
Explain the figure on slide 50 shows the code for style rules that are defined in `Products_stylesheet.xsl` file.

Then, explain the figure on slide 51 depicts the stylesheet for `Products_stylesheet.xsl` file.

The sorting order for `UnitPrice` is descending in that the higher value will be displayed first.

Slide 53

Let us summarize the session.



Using slide 53, summarize the session. End the session, with a brief summary of what has been taught in the session.

6.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the more on XSLT that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 7 - More on XSLT

7.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

7.1.1 Objectives

By the end of this session, the learners will be able to:

- XPath
- XPath Expressions and Functions
- Working with different Styles

7.1.2 Teaching Skills

To teach this session, you should be well-versed with XPath and identifying the various nodes of XPath. You should also aware yourself with the different operators used with XPath and learn the various XPath expressions and functions. Finally, this session explains how to switch between styles and how to transform XML documents into HTML using XSLT.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Then give the students the overview of the current session in the form of Session Overview. Show the students slide 2 of the presentation.

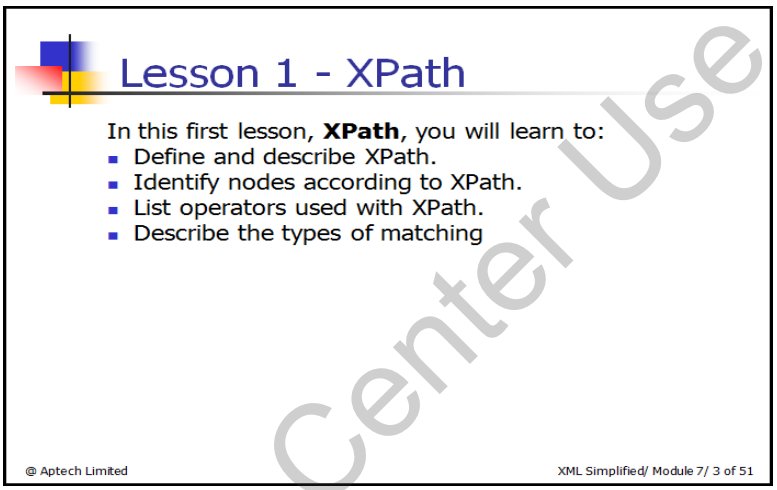
Tell the students that this session introduces XPath and identifying the various nodes of XPath. They will also learn the different operators used with XPath. They will understand and will be able to work with the various XPath expressions and functions.

They will also learn how to switch between styles and how to transform XML documents into HTML using XSLT.

7.2 In-Class Explanations

Slide 3

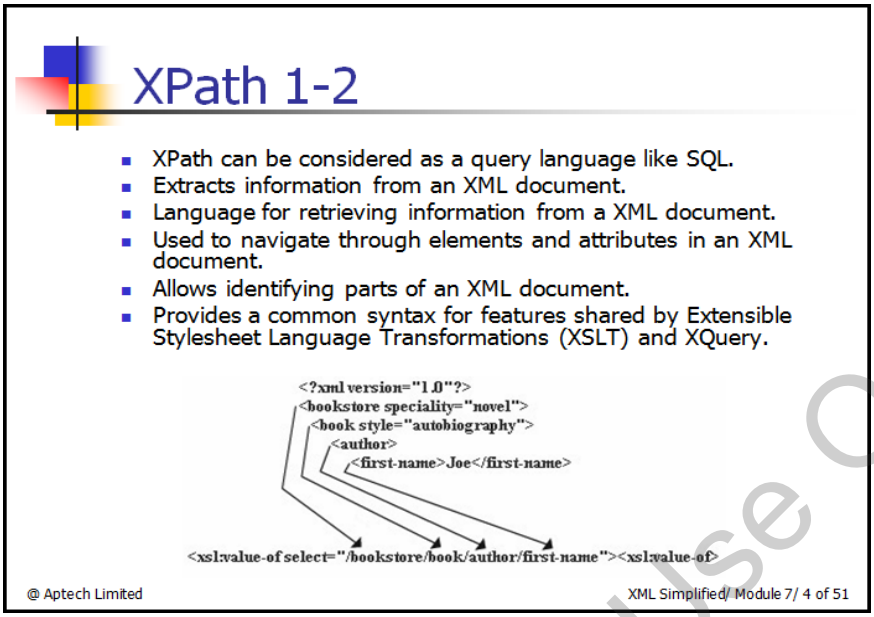
Let us understand the objectives of lesson 1.

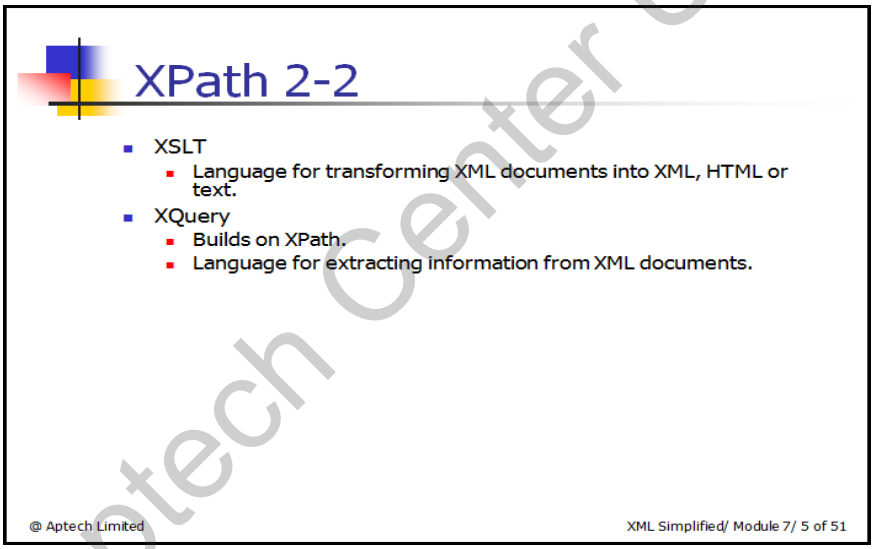


Using slide 3, explain the topics covered in the lesson 1.

Slides 4 and 5

Let us understand the XPath.





Using slides 4 and 5, explain the XPath.

XPath is used to navigate through elements and attributes in an XML document. It contains a library of standard functions that form a major element in XSLT. It is a major element in XSLT standards recommended by W3C.

XPath can be thought of as a query language like SQL. However, rather than extracting information from a database, it extracts information from an XML document. XPath is a language for retrieving information from a XML document. XPath is used to navigate through elements and attributes in an XML document. Thus, XPath allows identifying parts of an XML document.

XPath provides a common syntax as shown in figure on slide 4 for features shared by XSLT and XQuery.

- **XSLT** – It is a language for transforming XML documents into XML, HTML, or text.
- **XQuery** – It is built on XPath and is a language for extracting information from XML documents.

In-Class Question:

After you finish explaining the XPath, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

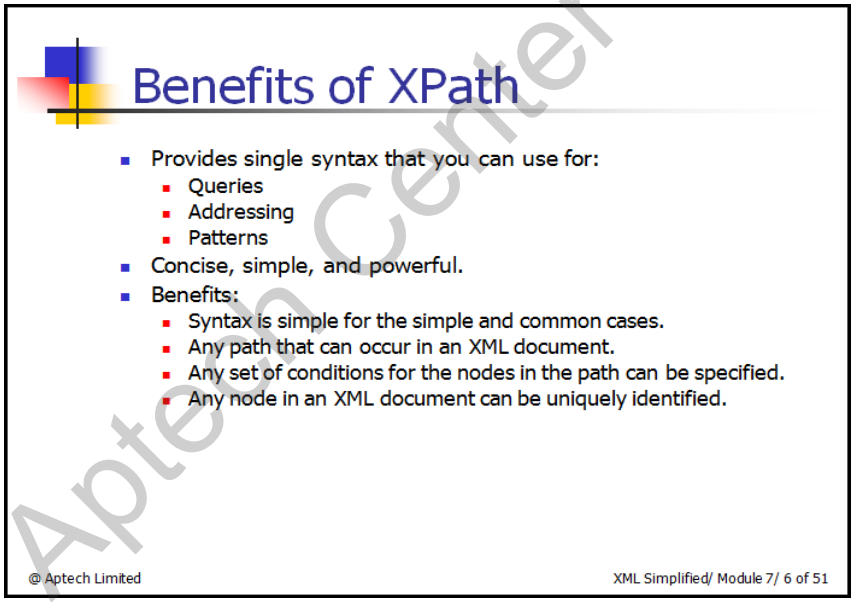
Which language can be used for querying XML data?

Answer:

XQuery.

Slide 6

Let us understand the benefits of XPath.



Using slide 6, explain the benefits of XPath.

XPath is designed for XML documents. It provides a single syntax that you can use for queries, addressing, and patterns. XPath is concise, simple, and powerful.

Explain the benefits of XPath in details.

XPath is designed to be used in many contexts. It is applicable for providing links to nodes, searching repositories, and for many other applications.

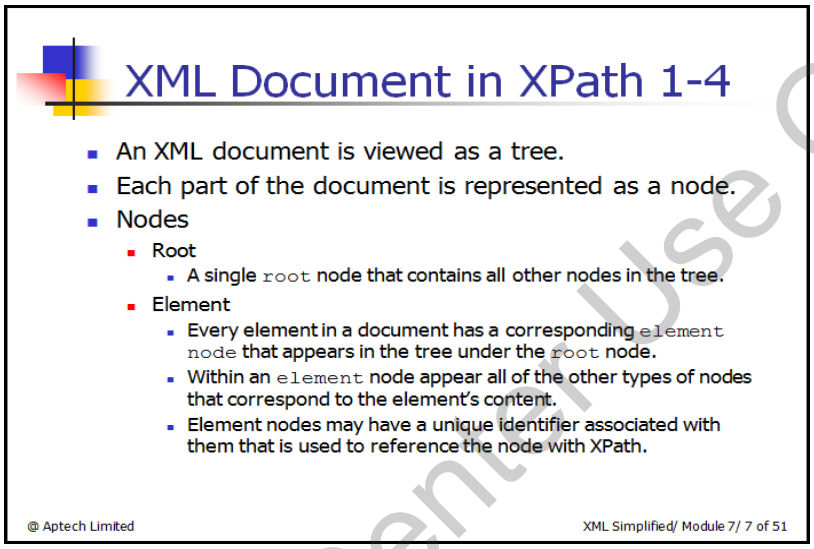
Tips:

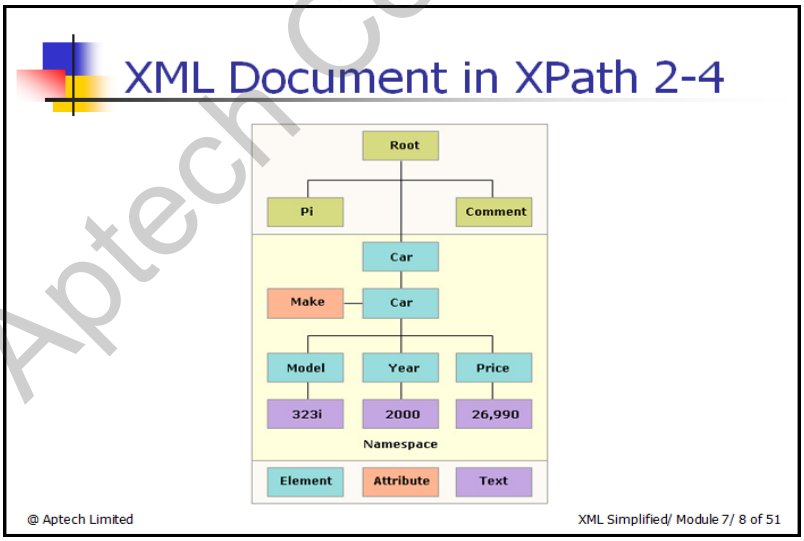
Knowledge of XPath is important for:

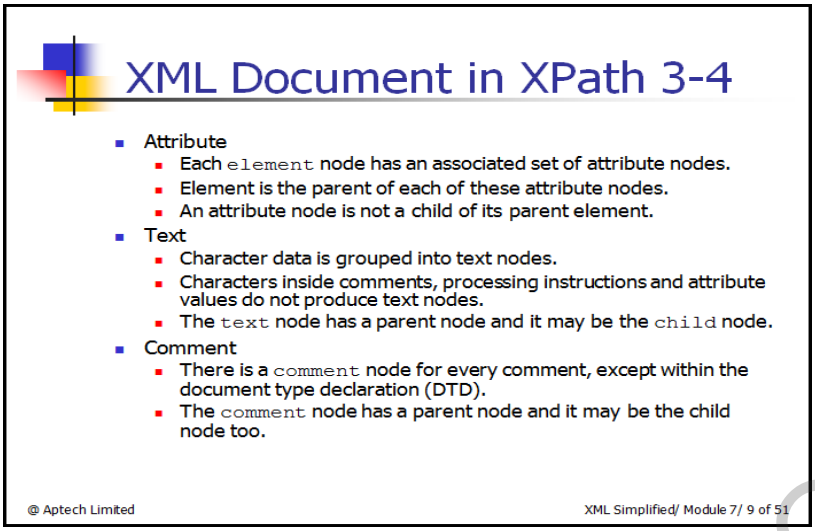
- Business analysts
- Project managers
- Report developers
- Programmers

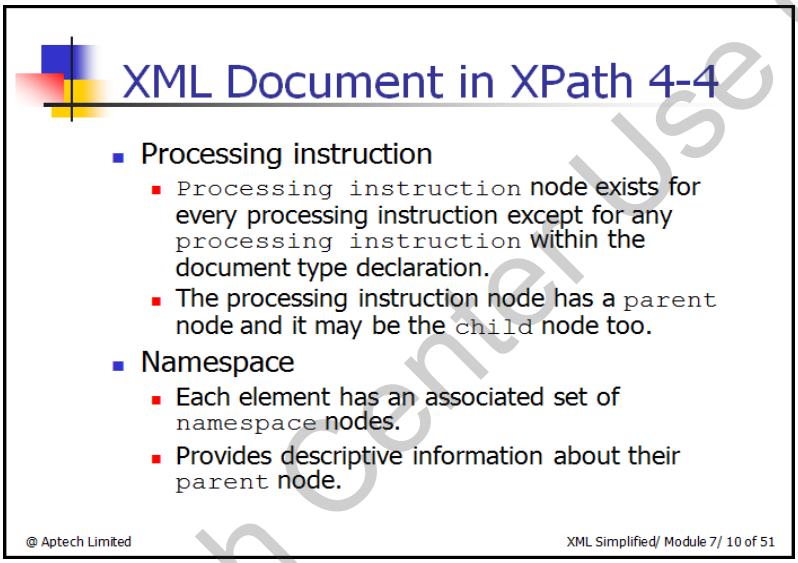
Slides 7 to 10

Let us understand the viewing of XML document in XPath.









Using slides 7 to 10, explain the viewing of XML document in XPath.

In XPath, an XML document is viewed conceptually as a tree in which each part of the document is represented as a node as shown in figure on slide 8.

XPath have seven types of nodes. Explain each of them in detail using slides 7, 9, and 10.

Following table summarizes the various nodes in XPath tree:

Node Type	String Value	Expanded Name
root	Determined by concatenating the string-values of all text-node descendants in document order.	None.
element	Determined by concatenating the string-values of all text-node descendants in document order.	The element tag, including the namespace prefix (if applicable).
attribute	The normalized value of the attribute.	The name of the attribute, including the namespace prefix (if applicable).

Node Type	String Value	Expanded Name
text	The character data contained in the text node.	None.
comment	The content of the comment (not including <!-- and -->).	None.
processing instruction	The part of the processing instruction that follows the target and any whitespace.	The target of the processing instruction.
namespace	The URI of the namespace.	The namespace prefix.

Tips:

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Relationship in Nodes:

XML documents are treated as trees of nodes. The topmost element of the tree is called the root element.

Look at the following XML document:

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

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Examples of nodes in the XML document are:

```
<bookstore> (root element node)

<author>J K. Rowling</author> (element node)

lang="en" (attribute node)
```

Parent - Each element and attribute has one parent.

In the bookstore XML document, the `book` element is the parent of the `title`, `author`, `year`, and `price`.

Children - Element nodes may have zero, one, or more children.

In the bookstore XML document, the `title`, `author`, `year`, and `price` elements are all children of the `book` element.

Siblings - Nodes that have the same parent.

In the bookstore XML document, the `title`, `author`, `year`, and `price` elements are all siblings.

Ancestors - A node's parent, parent's parent, and so on.

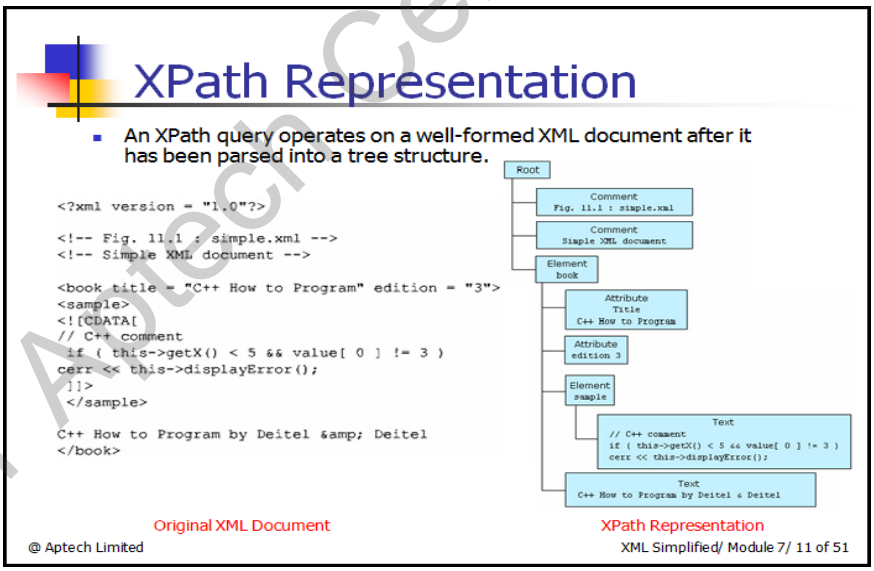
In the bookstore XML document, the ancestors of the `title` element are the `book` element and the `bookstore` element.

Descendants - A node's children, children's children, and so on.

In the bookstore XML document, descendants of the `bookstore` element are the `book`, `title`, `author`, `year`, and `price` elements.

Slide 11

Let us understand the XPath representation.



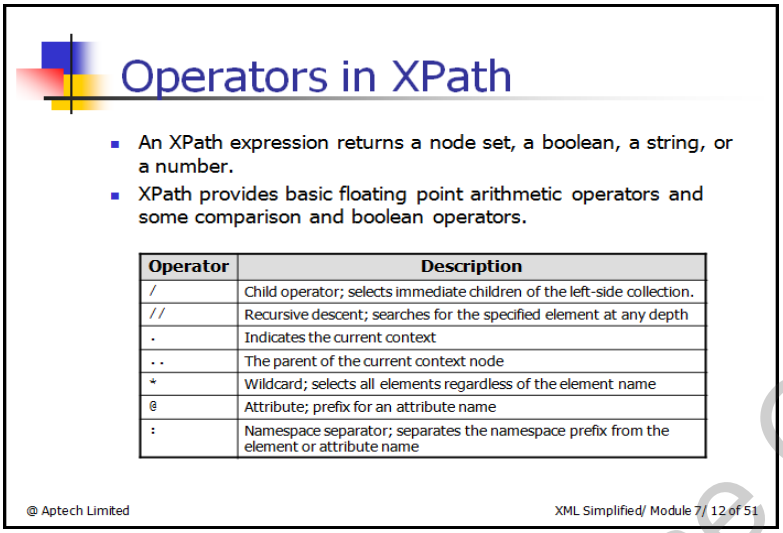
Using slide 11, explain the XPath representation.

An XPath query operates on a well-formed XML document after it has been parsed into a tree structure. Figure on slide 11 depicts a simple XML document along with its generated XPath tree.

Explain the figure showing the corresponding XPath tree that is generated after it has been parsed.

Slide 12

Let us understand the operators in XPath.



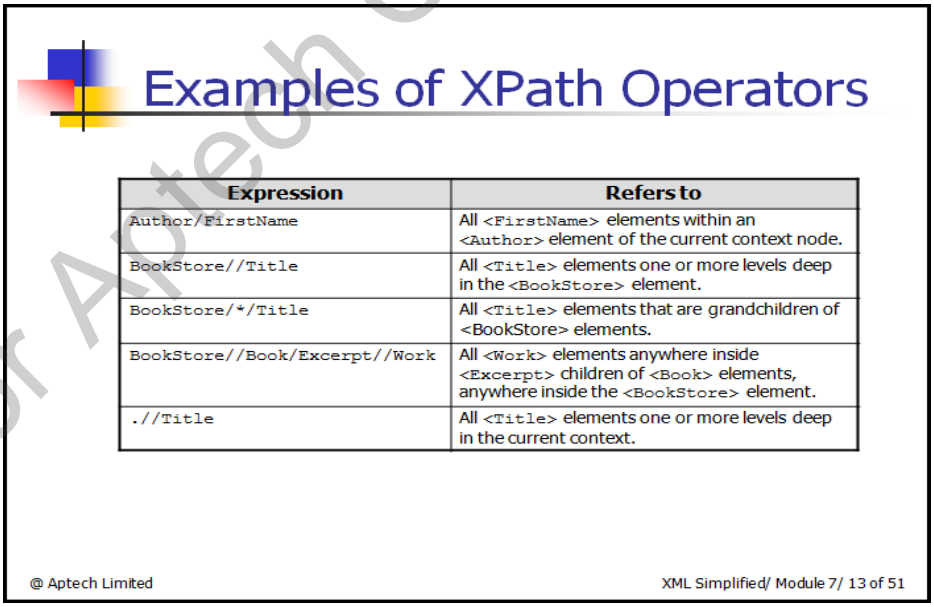
Using slide 12, explain the operators in XPath.

An XPath expression returns a node set, a boolean, a string, or a number. XPath provides basic floating point arithmetic operators and some comparison and boolean operators.

The XPath expressions are constructed using the operators and special characters as shown in the table shown on slide 12.

Slide 13

Let us understand the XPath operators with examples.

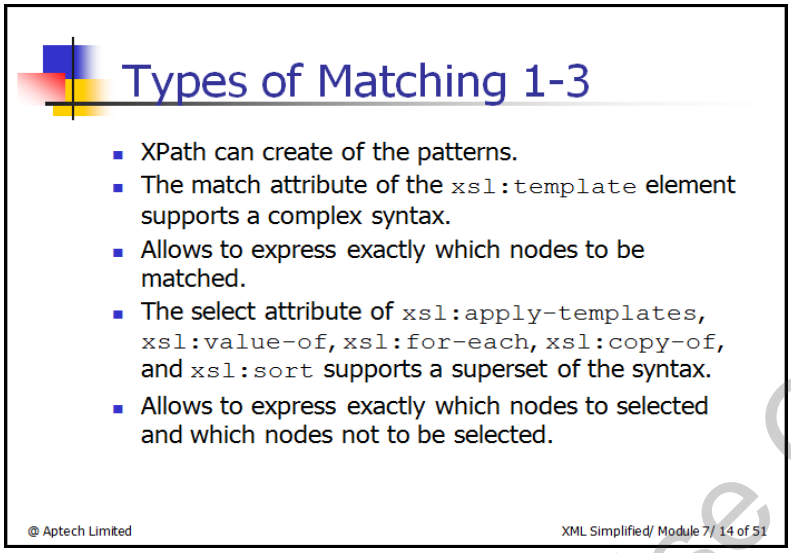


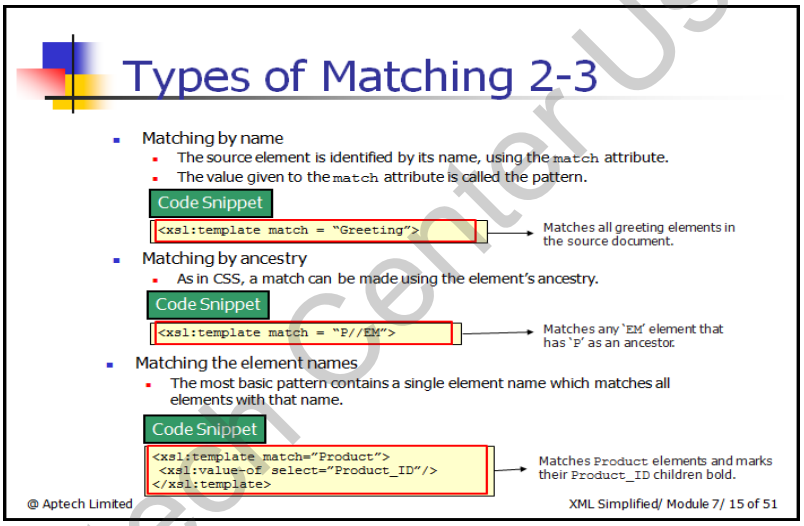
Using slide 13, explain the XPath operators with examples.

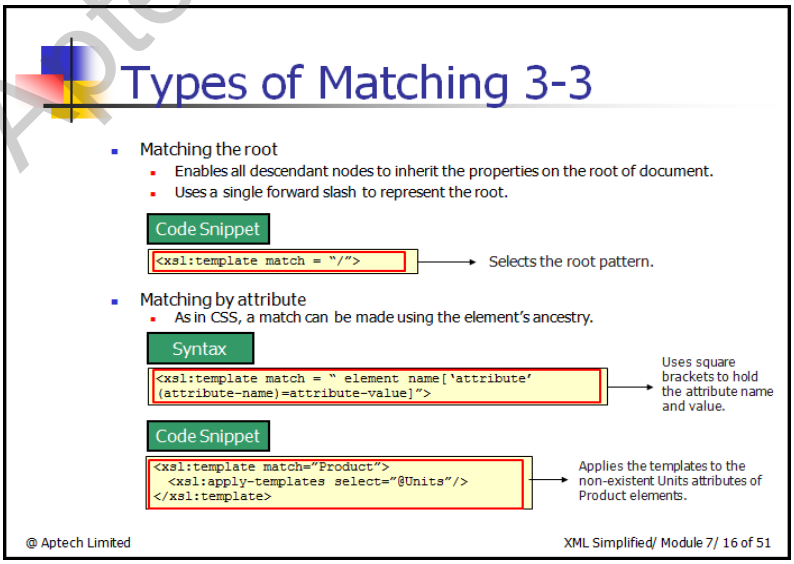
Table on slide 13 shows few examples along with their description that use XPath operators in an expression.

Slides 14 to 16

Let us understand the types of matching.







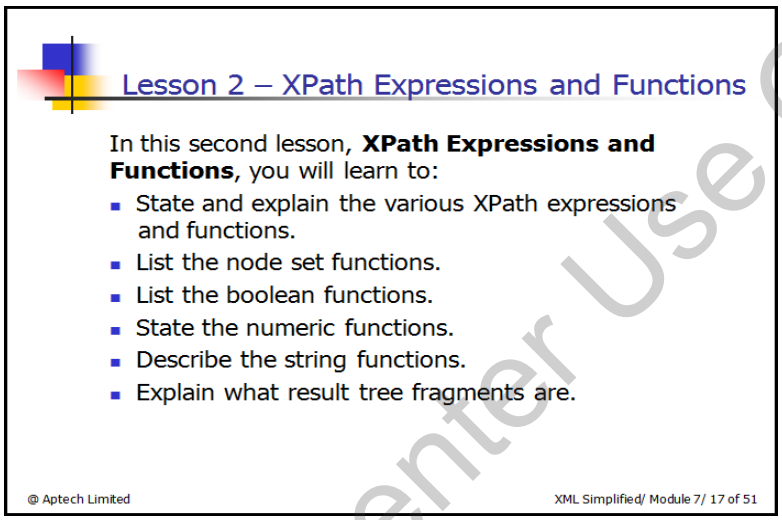
Using slides 14 to 16, explain types of matching.

XPath is used in the creation of the patterns. The match attribute of the `xsl:template` element supports a complex syntax that allows to express exactly which nodes to be matched. The `select` attribute of `xsl:apply-templates`, `xsl:value-of`, `xsl:for-each`, `xsl:copy-of`, and `xsl:sort` supports an even more powerful superset of this syntax that allows to express exactly which nodes to select and which nodes not to be selected.

Explain the types of matching in detail using slides 15 and 16.

Slide 17

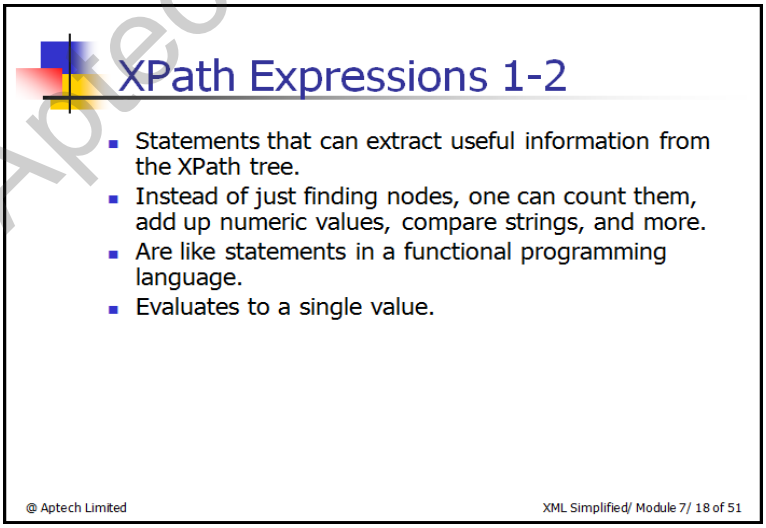
Let us understand the objectives of lesson 2.

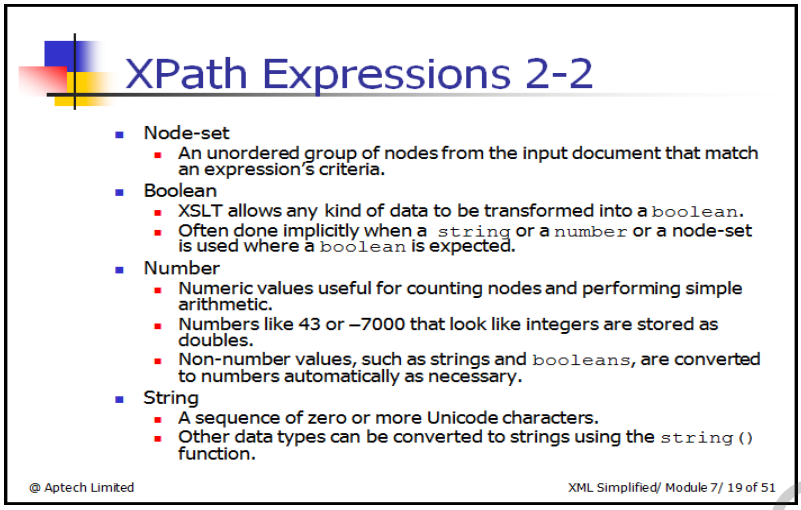


Using slide 17, explain the topics covered in the lesson 2.

Slides 18 and 19

Let us understand the XPath expressions.





Using slides 18 and 19, explain the XPath expression.

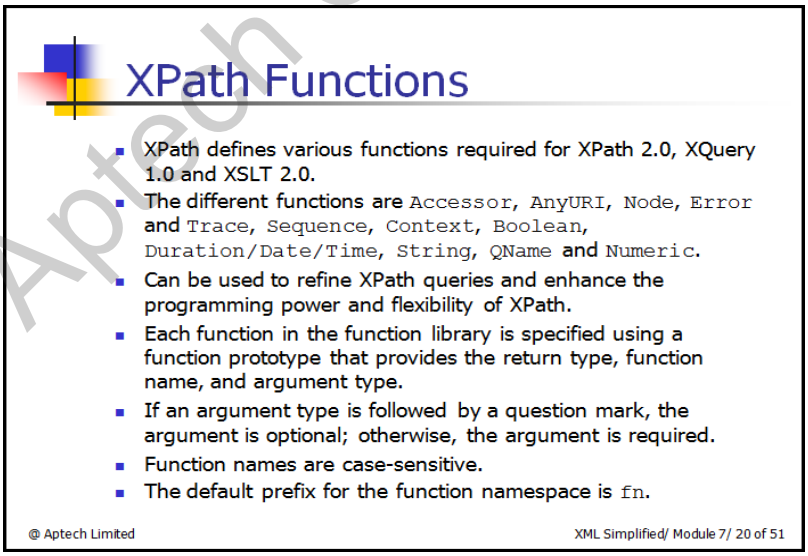
XPath expressions are statements that can extract useful information from the XPath tree. Instead of just finding nodes, one can count them, add up numeric values, compare strings, and more. They are much like statements in a functional programming language. Every XPath expression evaluates to a single value.

Explain the four types of expressions in XPath.

The XPath expression syntax includes literal forms for strings and numbers as well as operators and functions for manipulating all four XPath data types.

Slide 20

Let us understand the XPath functions.



Using slide 20, explain the XPath functions.

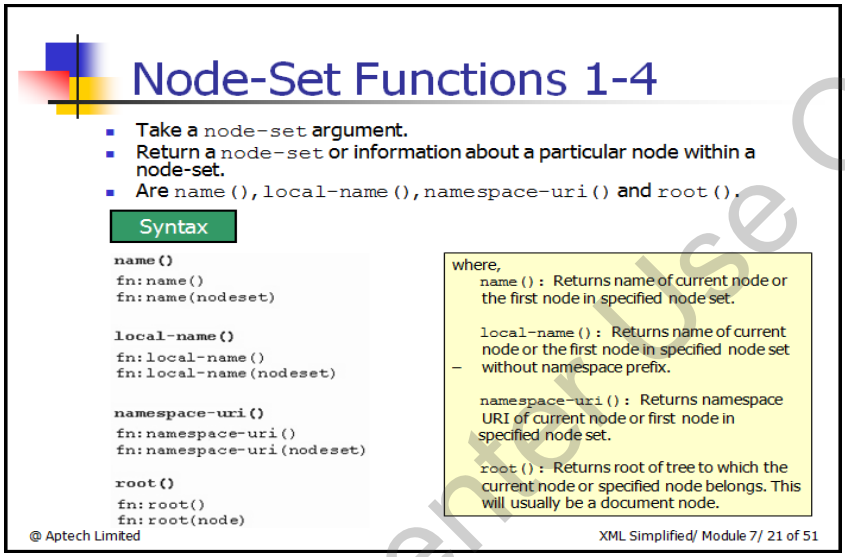
XPath defines various functions required for XPath 2.0, XQuery 1.0, and XSLT 2.0. The different functions are Accessor, AnyURI, Node, Error and Trace, Sequence, Context, Boolean, Duration/Date/ Time, String, QName, and Numeric.

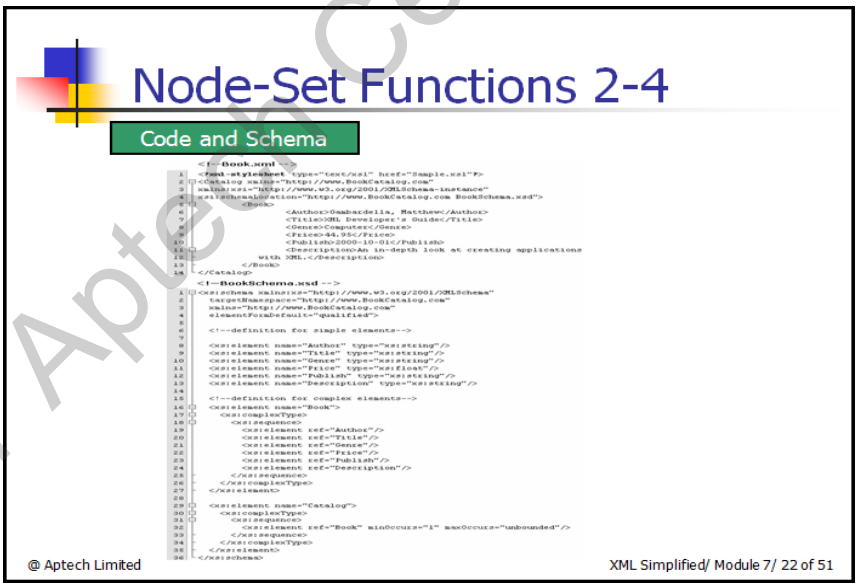
XML Path Language (XPath) functions can be used to refine XPath queries and enhance the programming power and flexibility of XPath. Each function in the function library is specified using a function prototype that provides the return type, function name, and argument type. If an argument type is followed by a question mark, the argument is optional; otherwise, the argument is required. Function names are case-sensitive.

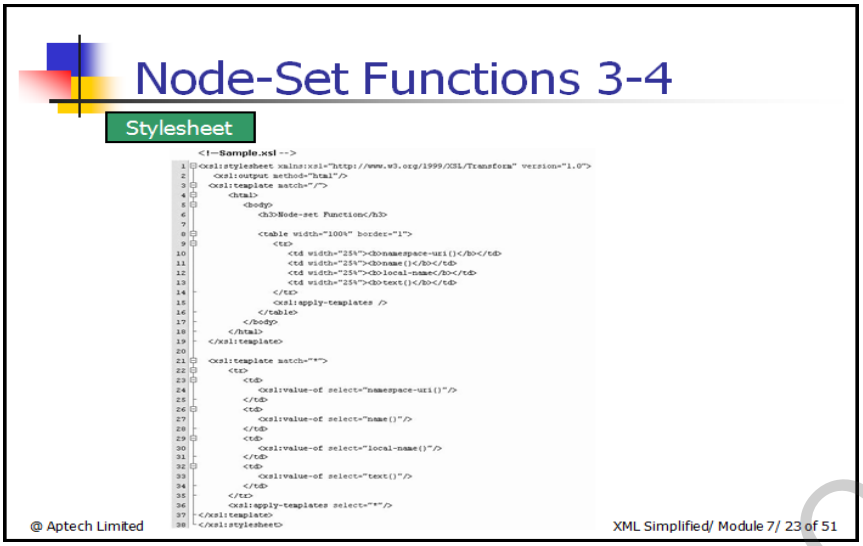
The default prefix for the function namespace is `fn`.

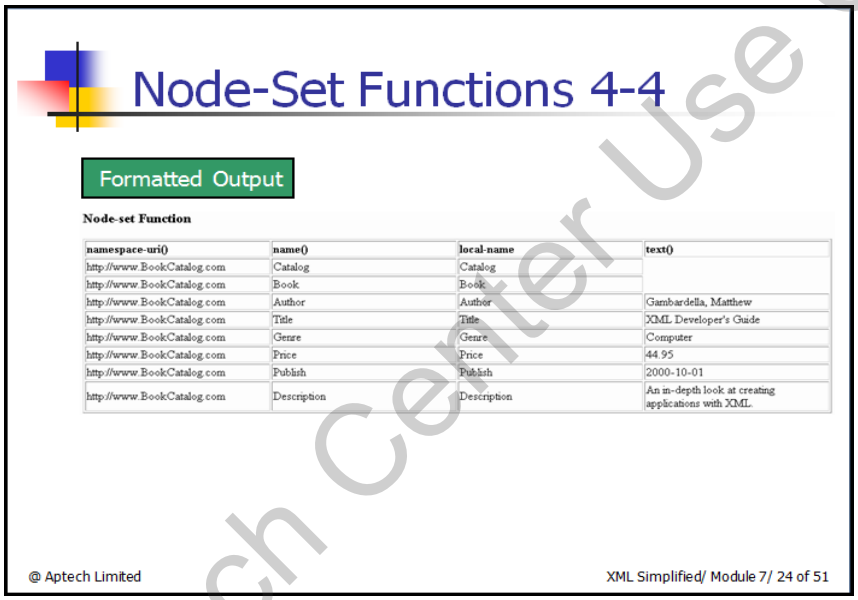
Slides 21 to 24

Let us understand the node-set functions.









Using slides 21 to 24, explain node-set functions.

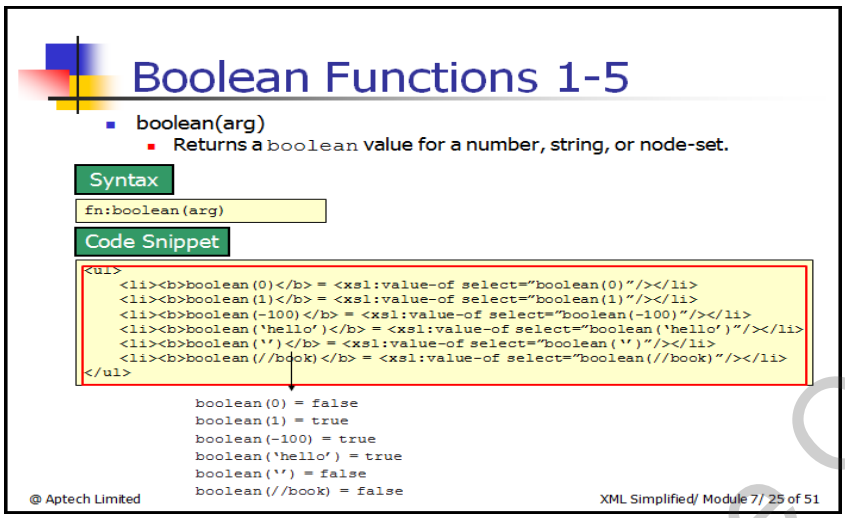
Node-set functions take a node-set argument. These return a node-set, or information about a particular node within a node-set. The different node-set functions are `name()`, `local-name()`, `namespace-uri()`, and `root()`.

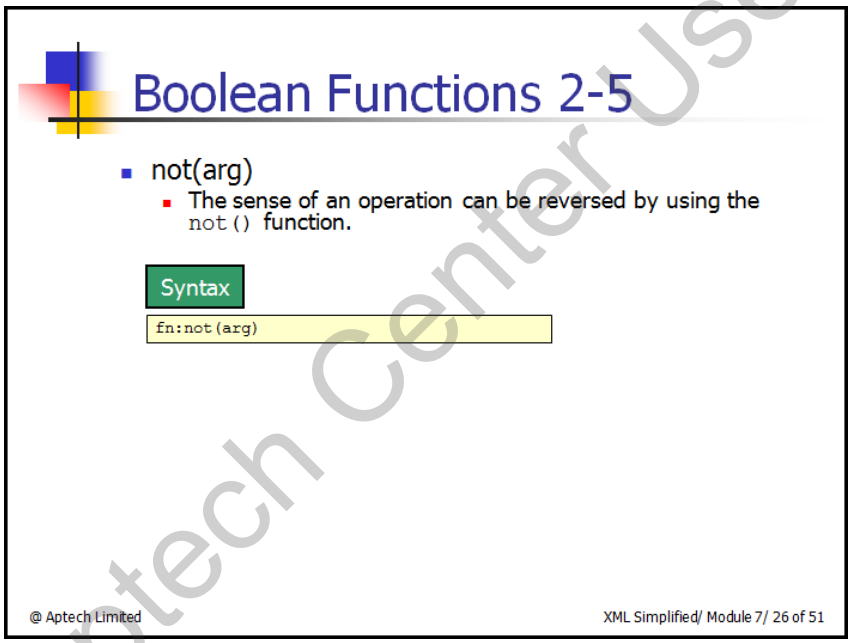
Figure on slide 21 shows the syntax for node-set functions. Explain the syntax. Figure on slide 22 shows the code and schema. Explain the code snippet.

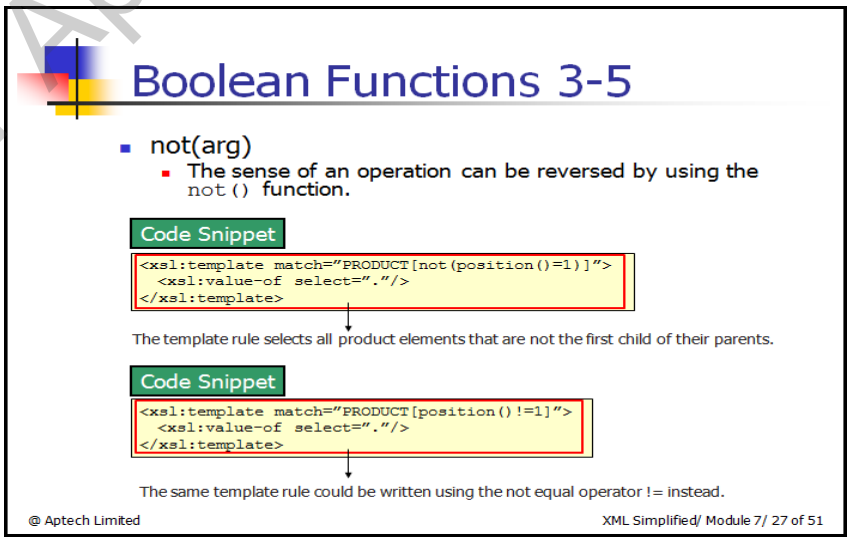
Explain the code snippet on slide 23 that depicts the style sheet.

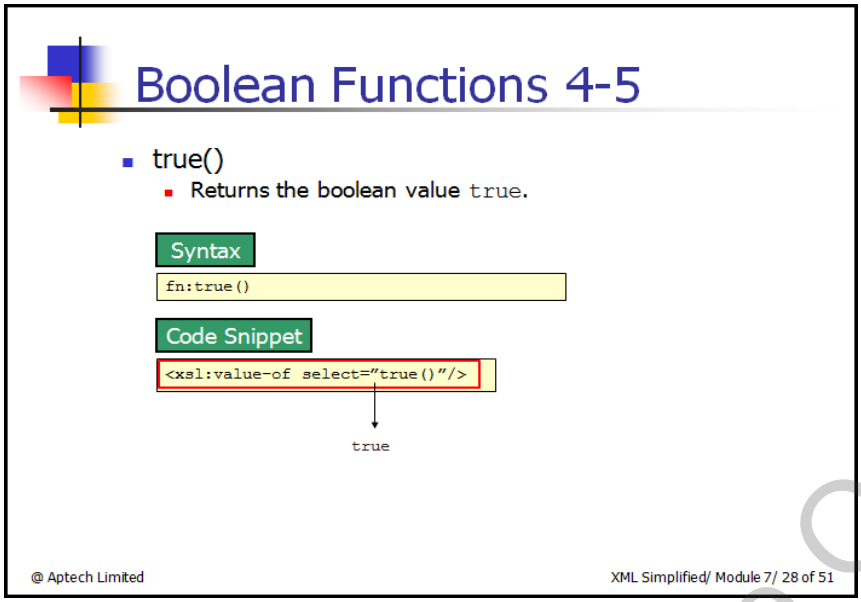
Slides 25 to 29

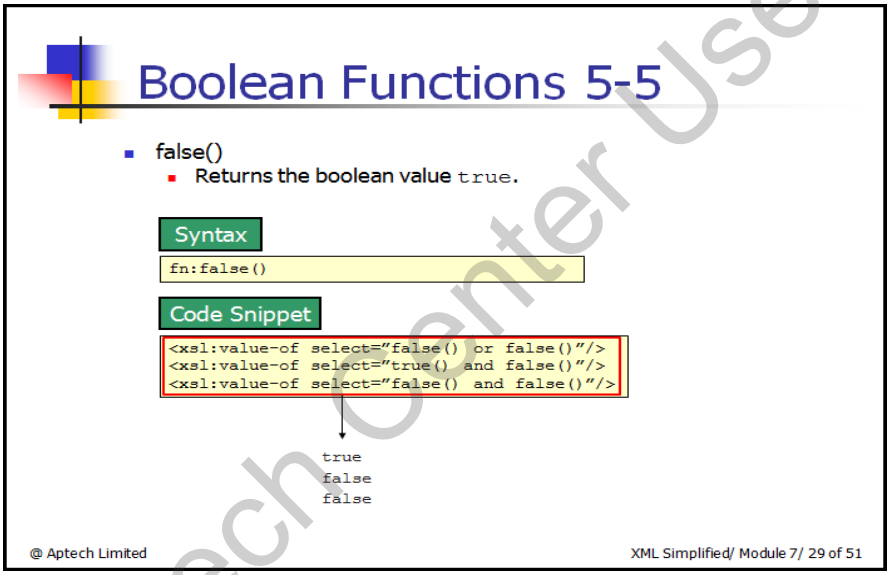
Let us understand the boolean functions.











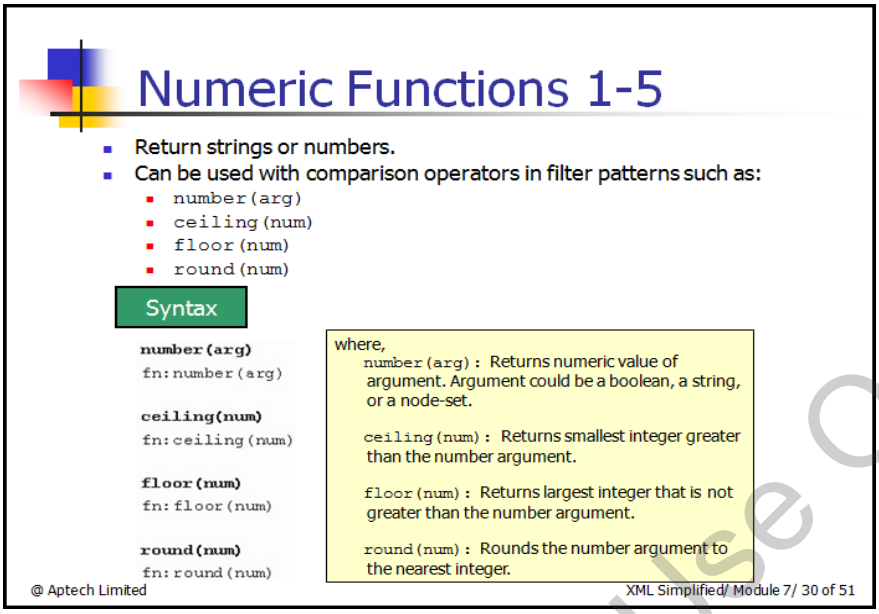
Using slides 25 to 29, explain the boolean functions.

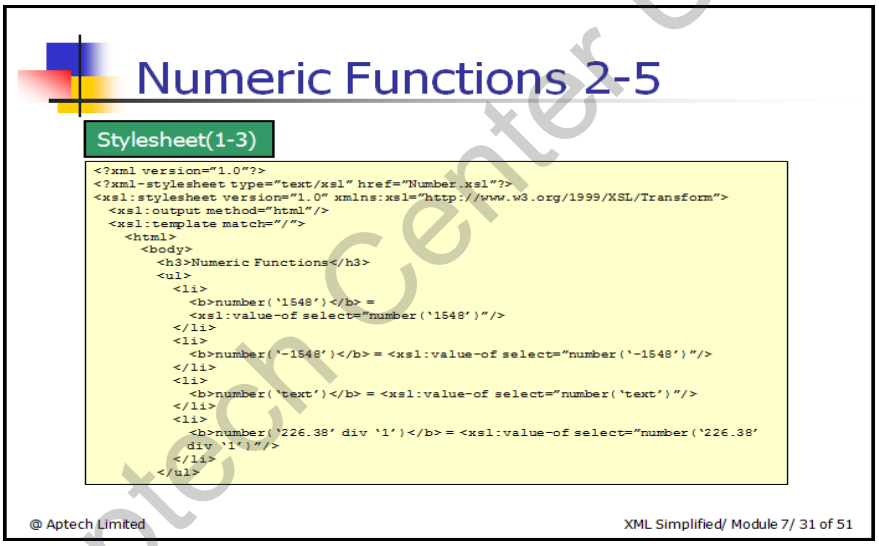
The XPath syntax supports boolean functions that return true or false, and can be used with comparison operators in filter patterns.

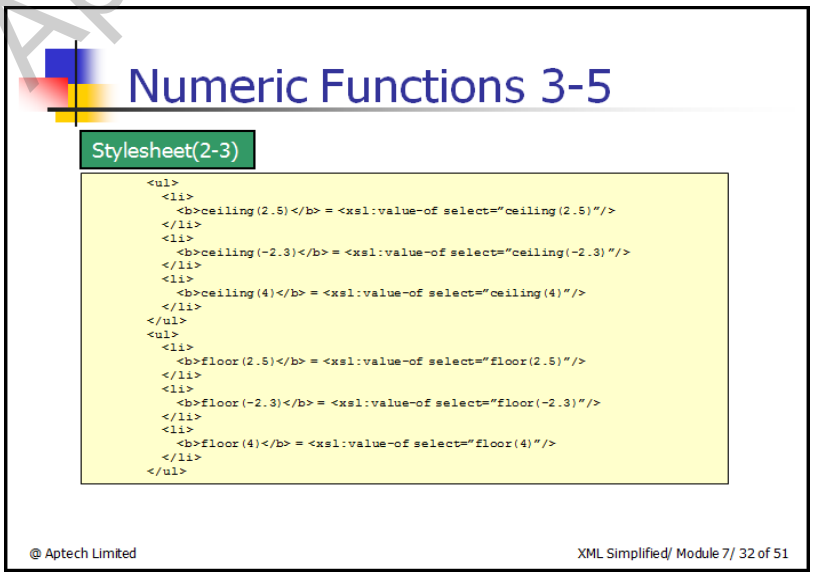
Explain the different boolean functions on the slides.

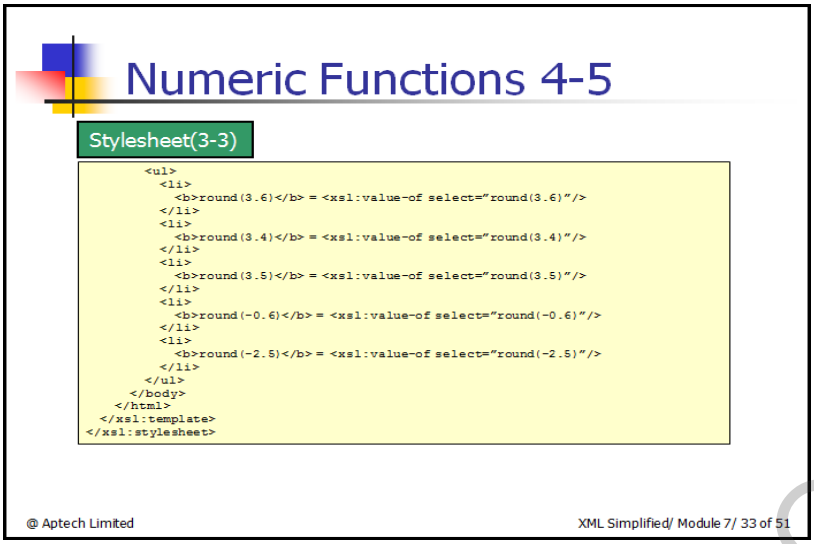
Slides 30 to 34

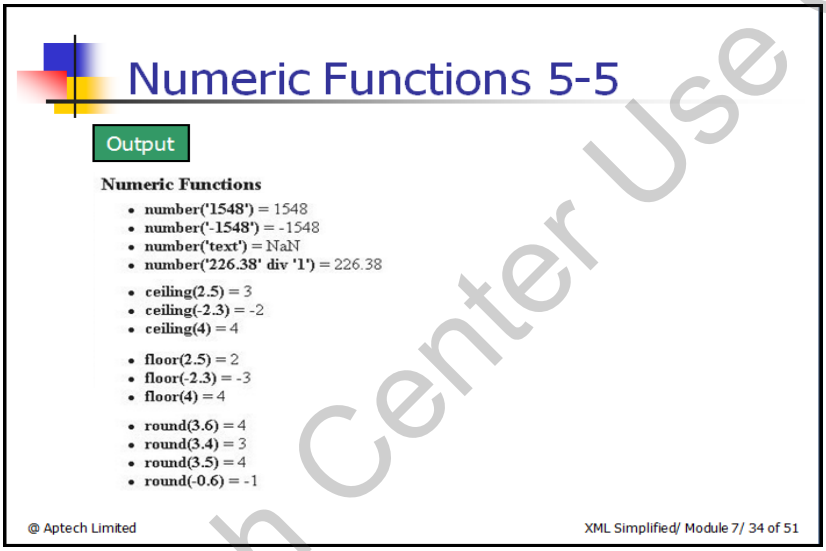
Let us understand the numeric functions.











Using slides 30 to 34, explain the numeric functions.

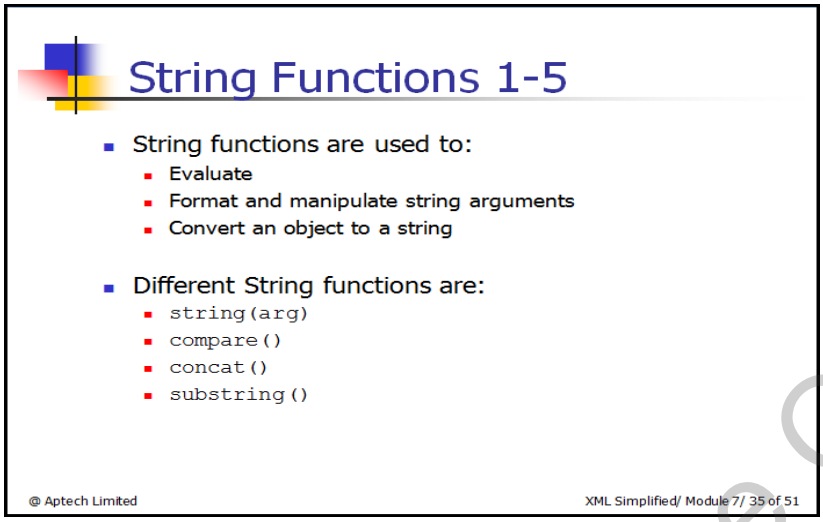
XPath syntax supports number functions that return strings or numbers and can be used with comparison operators in filter patterns. The different numeric functions are `number(arg)`, `ceiling(num)`, `floor(num)`, and `round(num)`.

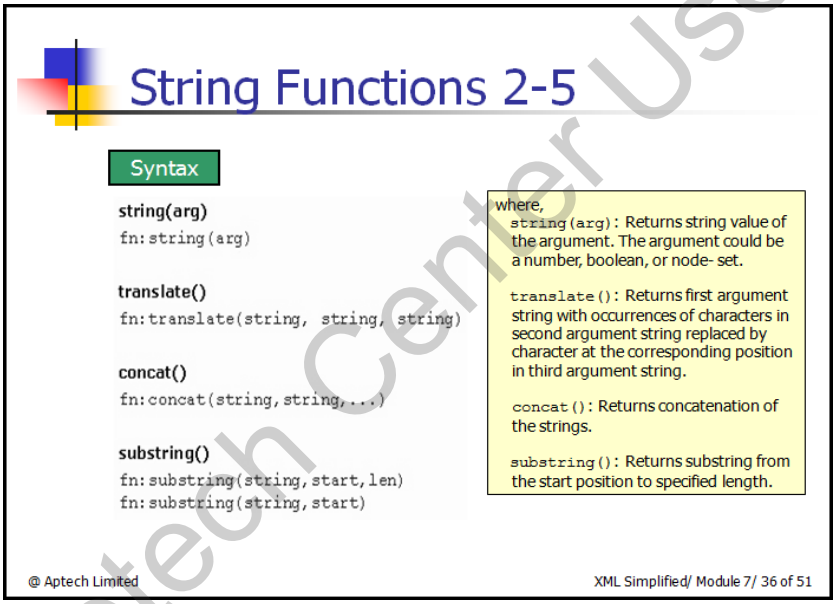
Explain the figure on slide 30 shows the syntax for numeric functions.

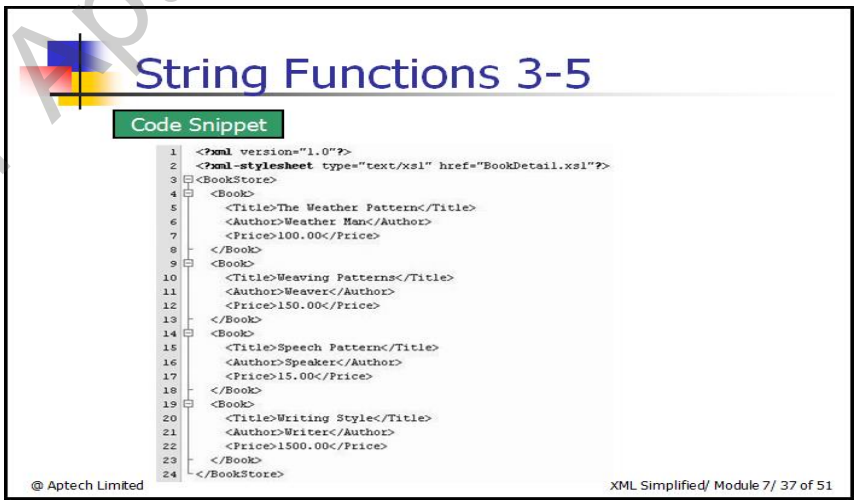
Explain the code snippet on slides 31 to 33 depicts the style sheet for numeric functions. Explain the code snippet.

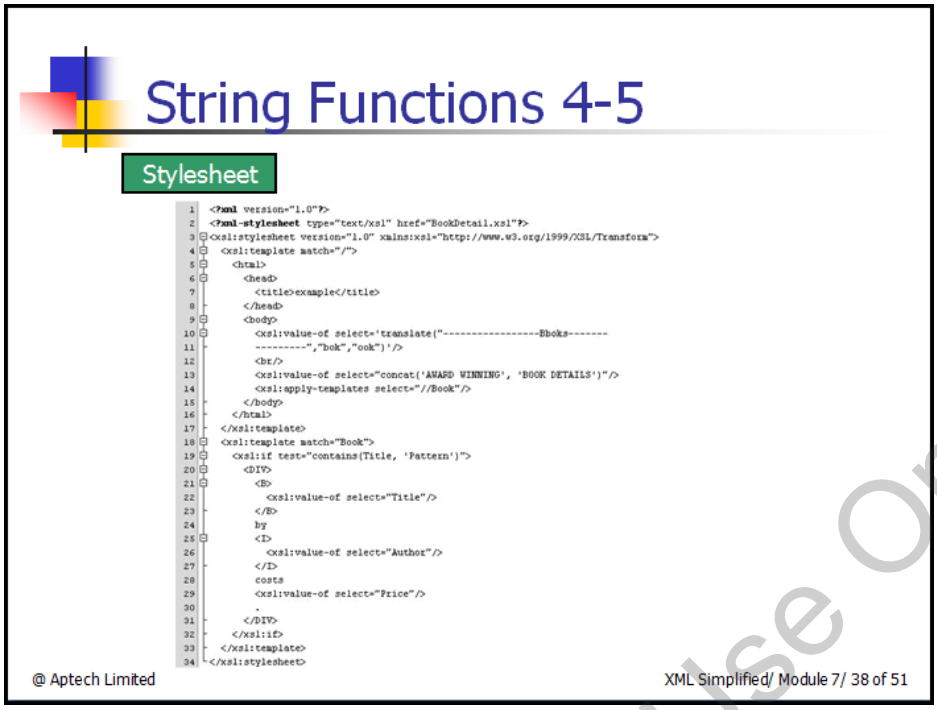
Slides 35 to 39

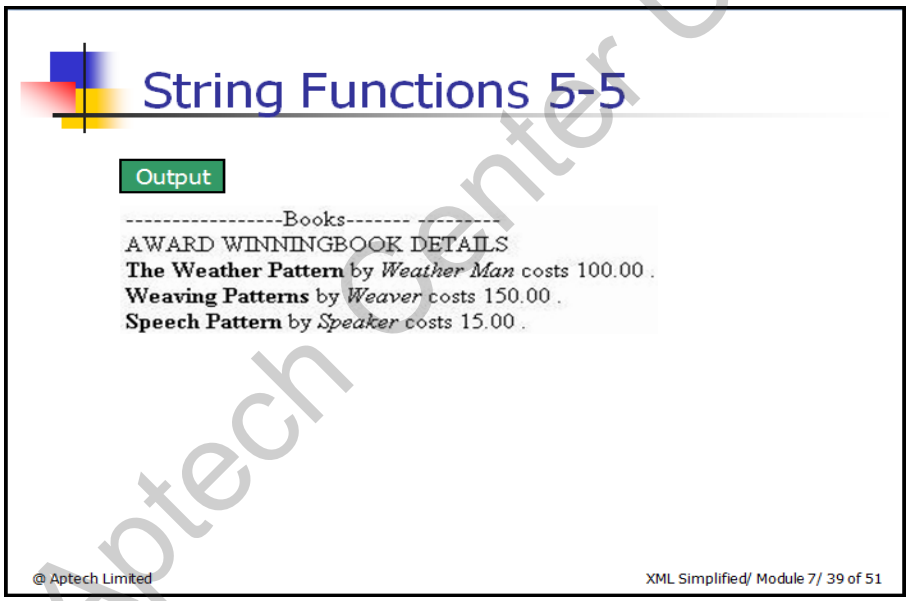
Let us understand the string functions.











Using slides 35 to 39, explain the string functions.

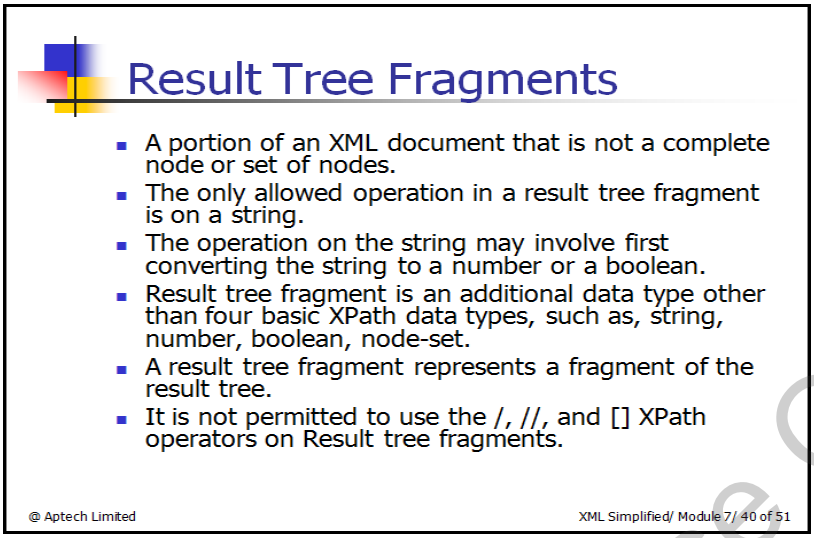
String functions are used to evaluate, format, and manipulate string arguments, or to convert an object to a string.

Explain the different string functions using slide 35.

Explain the figure on slide 36 shows the syntax for string functions. Then, explain the code snippet provided on slides 37 and 38.

Slide 40

Let us understand the result tree fragments.



Using slide 40, explain the result tree fragments.

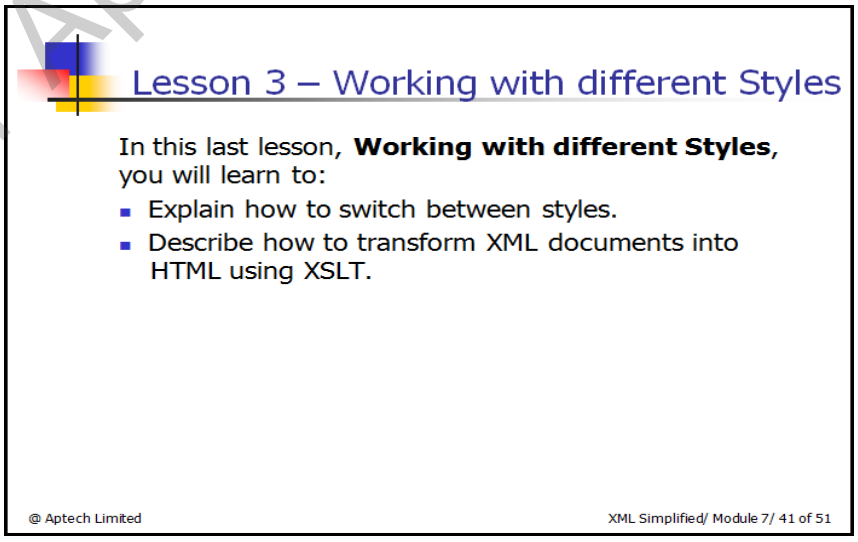
A result tree fragment is a portion of an XML document that is not a complete node or set of nodes.

The only allowed operation in a result tree fragment is on a string. The operation on the string may involve first converting the string to a number or a boolean. Result tree fragment is an additional data type other than four basic XPath data types (such as string, number, boolean, node-set).

A result tree fragment represents a fragment of the result tree. In particular, it is not permitted to use the `/`, `//`, and `[]` XPath operators on result tree fragments.

Slide 41

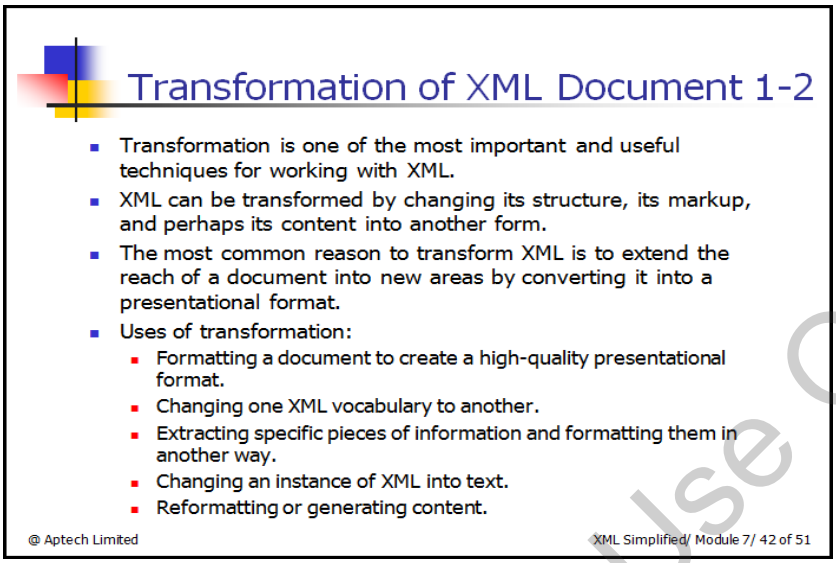
Let us understand the objectives of lesson 3.

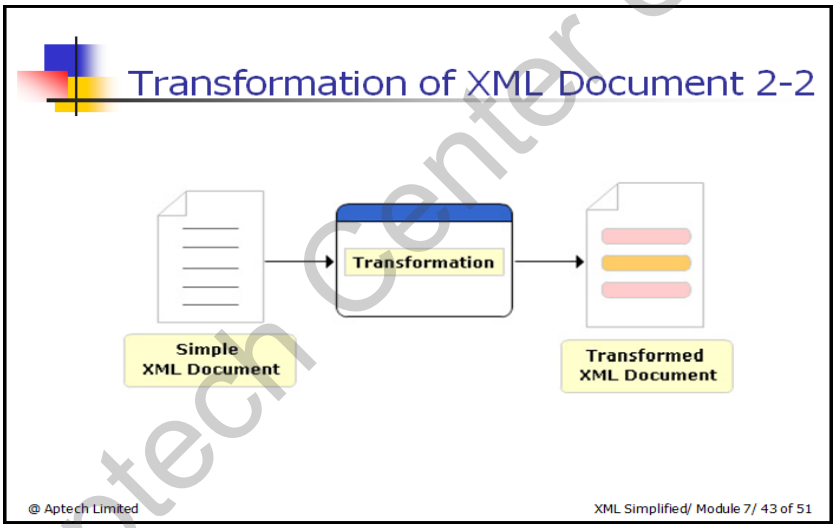


Using slide 41, explain the topics covered in lesson 3.

Slides 42 and 43

Let us understand how to transform a XML document.





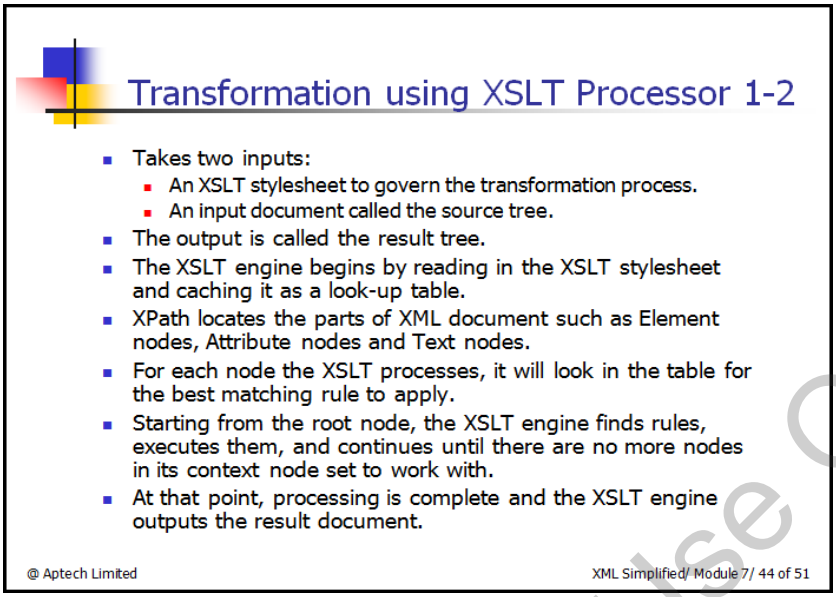
Using slides 42 and 43, explain how to transform a XML document.

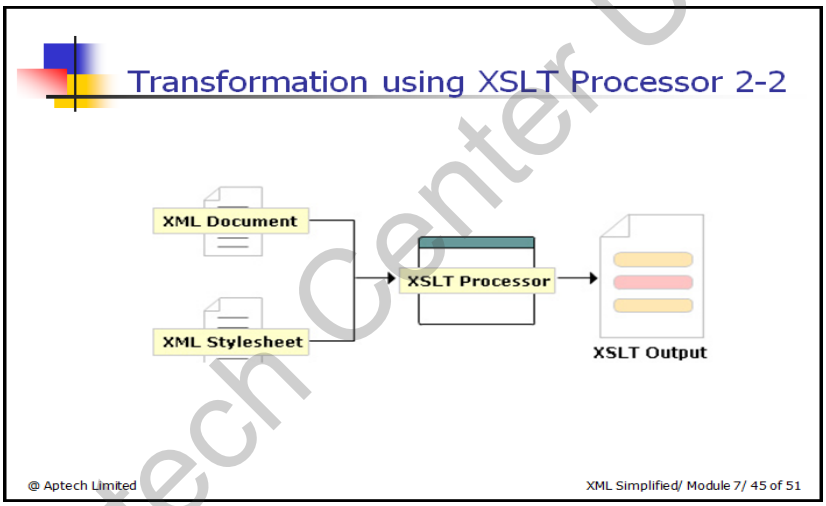
Transformation is one of the most important and useful techniques for working with XML. XML can be transformed by changing its structure, its markup, and perhaps its content into another form. The most common reason to transform XML is to extend the reach of a document into new areas by converting it into a presentational format.

Explain the figure on slide 43 demonstrating the transformation of XML document.

Slides 44 and 45

Let us understand transforming using XSLT processor.





Using slides 44 and 45, explain transformation using XSLT processor.

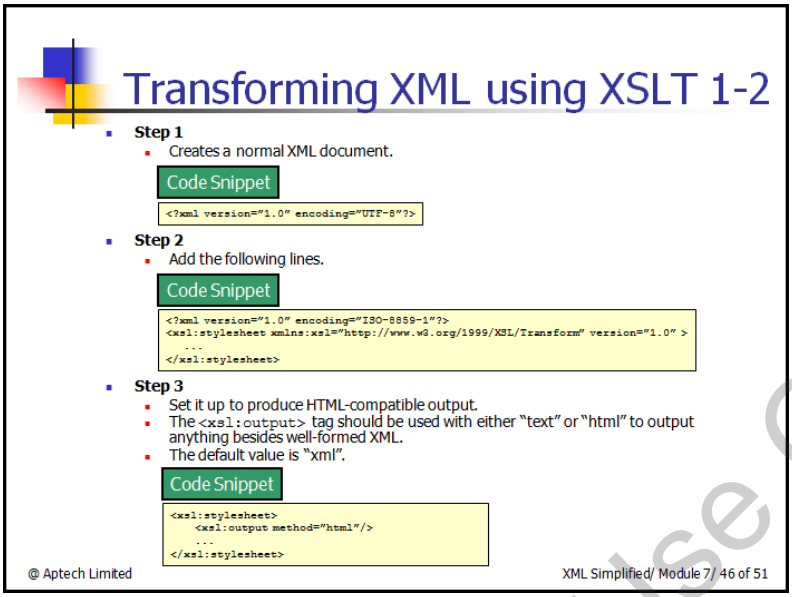
An XSLT processor takes two things as input: an XSLT style sheet to govern the transformation process and an input document called the source tree. The output is called the result tree.

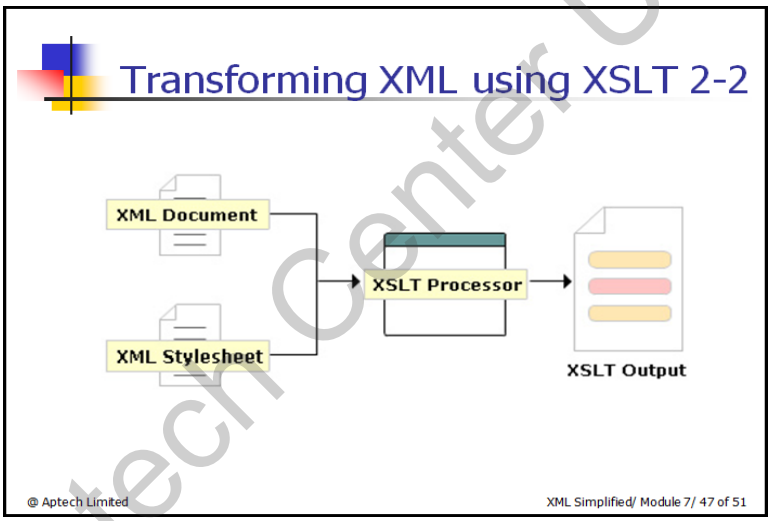
The XSLT engine begins by reading in the XSLT style sheet and caching it as a look-up table. XPath locates the parts of XML document such as Element nodes, Attribute nodes, and Text nodes. Thus, for each node the XSLT processes, it will look in the table for the best matching rule to apply. Starting from the root node, the XSLT engine finds rules, executes them, and continues until there are no more nodes in its context node set to work with. At that point, processing is complete and the XSLT engine outputs the result document.

Explain the figure on slide 45 depicting the XSLT transformation.

Slides 46 and 47

Let us understand transforming XML using XSLT.





Using slides 46 and 47, explain transforming XML using XSLT. The transformation of XML document can be done in various steps that are:

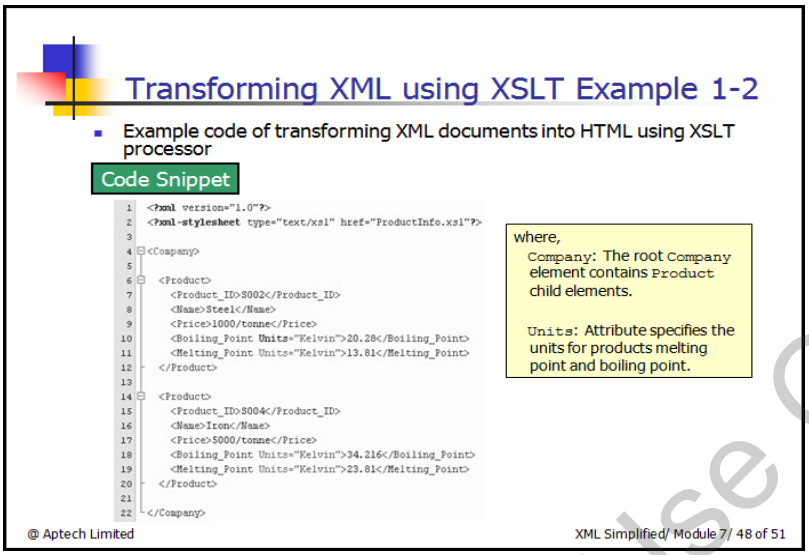
Step 1: Create a normal XML document.

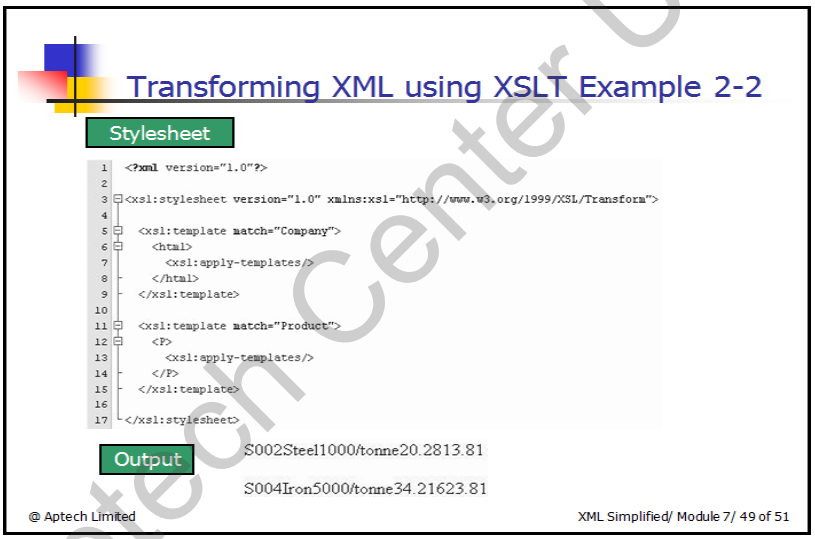
Step 2: Create an XSL style sheet.

Step 3: Set it up to produce HTML-compatible output.

Slides 48 and 49

Let us understand transforming XML using XSLT example.





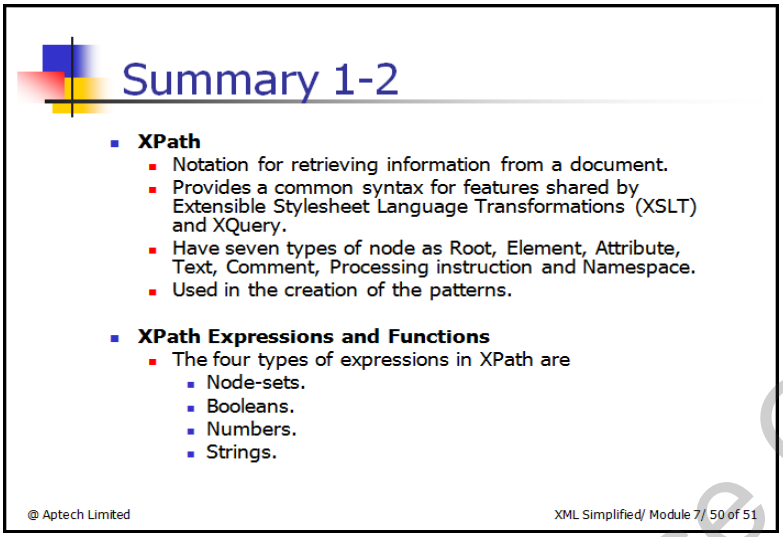
Using slides 48 and 49, explain transforming XML using XSLT example.

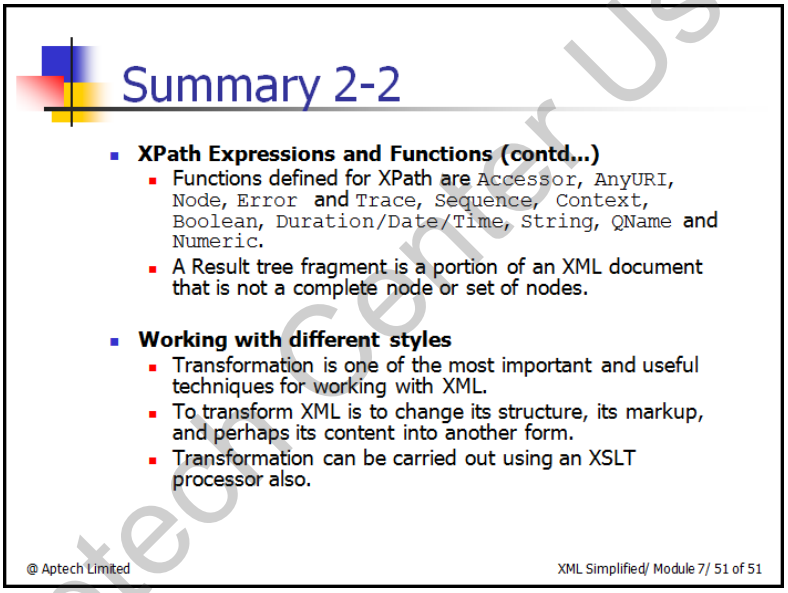
An example code of transforming XML documents into HTML using XSLT processor has been provided to explain the process.

Then, explain the code on slides 48 and 49 that depicts the style sheet for `ProductionInfo.xsl` file.

Slides 50 and 51

Let us summarize the session.





Using slides 50 and 51, summarize the session. End the session, with a brief summary of what has been taught in the session.

7.3 Post Class Activities for Faculty

You can brief the objectives of what has been taught in the course. You can solve the queries related to other sessions taught in the course.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 8: Introduction to JSON

8.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

8.1.1 Objectives

By the end of this session, learners will be able to:

- Define and describe JavaScript Object Notation (JSON)
- Explain the uses of JSON
- List and describe the features of JSON
- Compare JSON with XML
- Describe the advantages of JSON
- Explain syntax and rules
- Explain literal notation in JavaScript and conversion to JSON
- Explain how to create a simple JSON file

8.1.2 Teaching Skills

To teach this session, you should be well versed with JSON along with its history, features, uses, and benefits. You should be familiar with the differences between JSON and XML, JSON syntax and rules, and JavaScript literal notation. You should also be aware of how JSON files are created.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

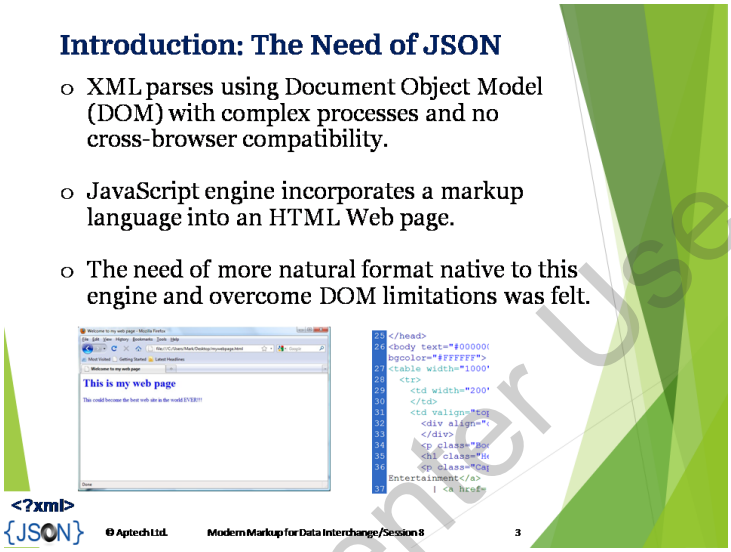
Overview of the Session:

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

8.2 In-Class Explanations

Slide 3

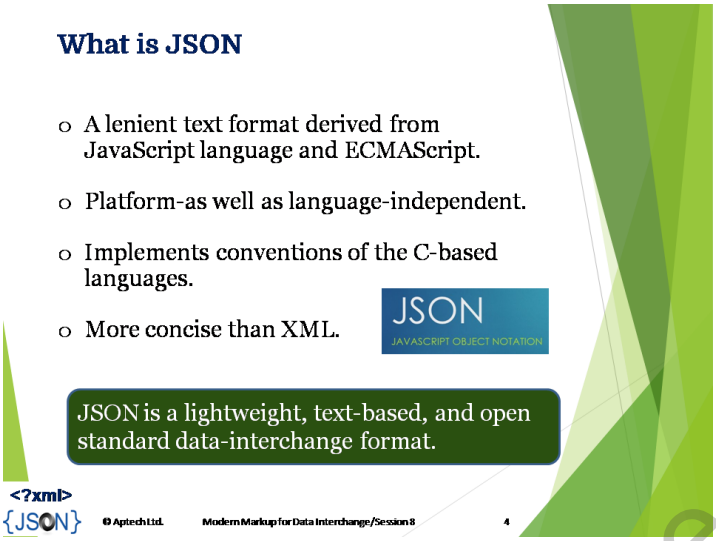
Let us understand the need of JSON.



Using slide 3, explain the need of JSON to students. Tell them that XML parses text by using Document Object Model (DOM). This model calls several methods and that DOM works differently in each browser. As a result, parsing becomes complex and time consuming. Moreover, the risk of cross-browser compatibility failure also exists. Due to these facts, it became essential to have a new component to incorporate a markup language into an HTML Web page easily in all mainstream browsers. This component was the JavaScript engine that required a more discerning and natural format. Thus, JSON was introduced.

Slide 4

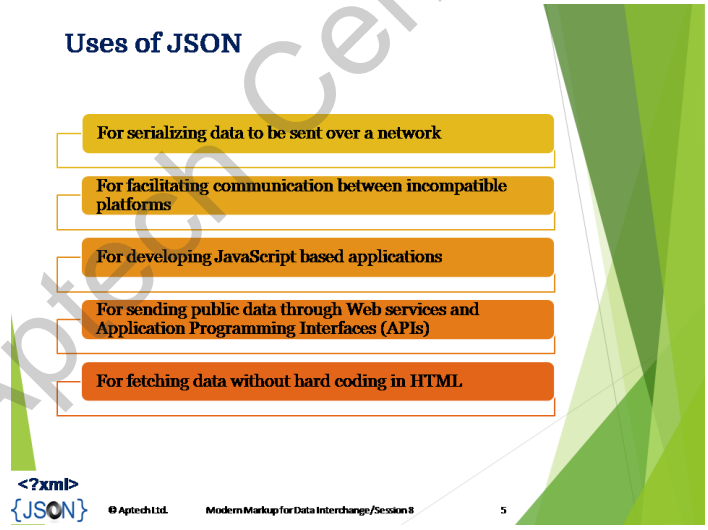
Let us understand JSON.



Using slide 4, briefly explain JSON to students. Focus on the definition. Tell them that JSON implements conventions that are familiar to the programmers of C languages, such as C#, Java, Perl, and JavaScript.

Slide 5

Let us explore the uses of JSON.



Using slide 5, briefly explain the uses of JSON. Tell them that serialization converts data structures to a format that is suitable for transferring over a network or for storing in a memory or file. Give them an example of incompatible platforms. For instance, JSON allows communicating between a Java application on UNIX and a Visual Basic application on Windows. Give them a real life scenario for the last use, such as the Web site eBay.com wherein a template defines how the site appears to visitors, rather than hard coding each product. The desired data is then retrieved from the JSON file on server.

Additional Information:

For more information, visit the following links:
http://tutorials.pluralsight.com/software-engineering-best-practices/everything-you-need-to-know-about-json?branch=vCillusion
http://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/

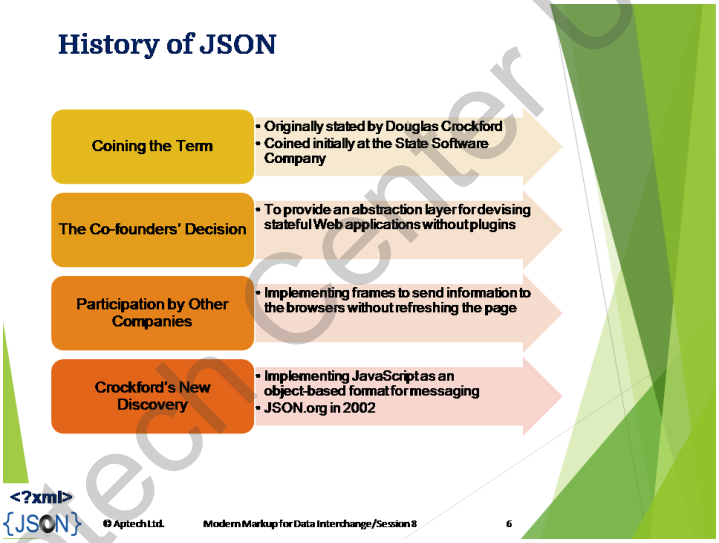
In-Class Question:

After you finish explaining the uses of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Why JSON is used to facilitate communication between incompatible platforms?

Answer: JSON as a data-interchange format is used to facilitate communication between incompatible platforms because it works on almost any platform and with most programming languages.

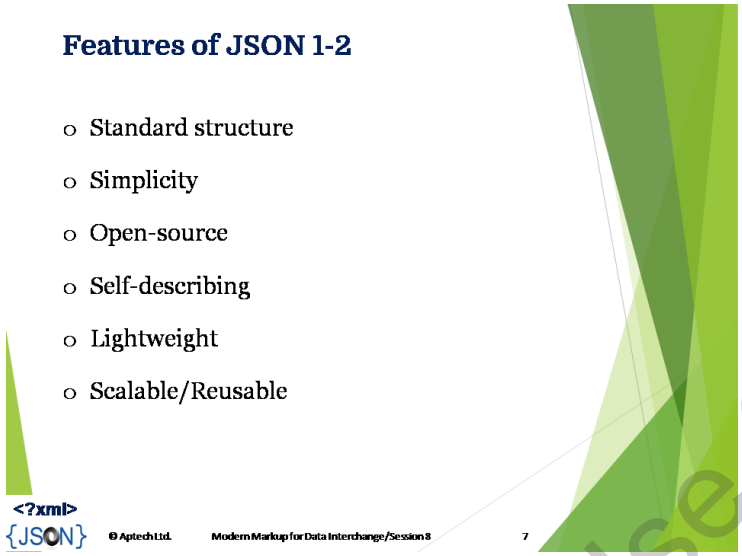
Slide 6
Let us see the history of JSON.

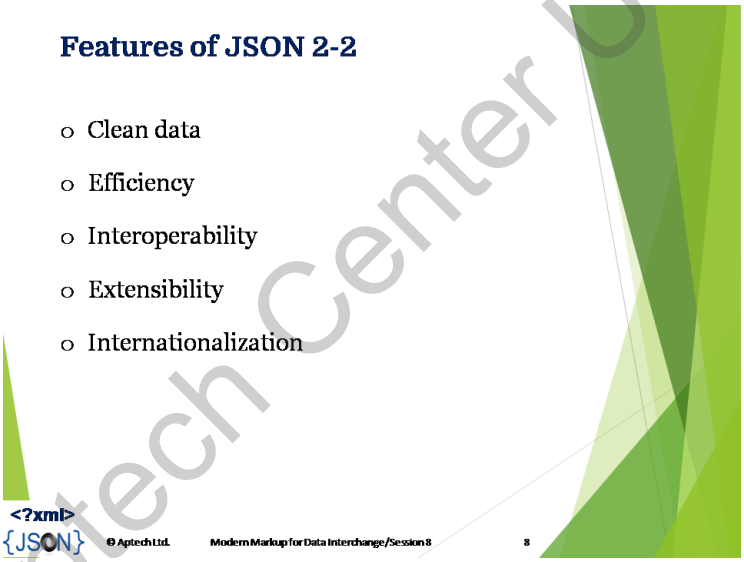


Using slide 6, explain to students how JSON evolved. Tell that the co-founders wanted a system that can take the advantage of the browser potential to give a layer of abstraction for designing stateful Web applications. These applications need to retain a lasting connection with a server by having two Hypertext Transfer Protocol (HTTP) connections open. For such a system, Crockford found that JavaScript can be implemented as an object-based format for messaging, which paved the way for JSON.

Slides 7 and 8

Let us now understand the features of JSON.





Using slides 7 and 8, explain each feature of JSON to students.

- **Standard Structure**: Makes it easier to code in JSON.
- **Simplicity**: Makes it easier to understand the code as well as parse.
- **Open**: Is publicly available for adding other elements when required.
- **Self-describing**: Allows describing one's own field names and their values.
- **Lightweight**: Takes less storage space than XML format as well as is easier to obtain and load such data without reloading the page.
- **Scalable/Reusable**: Makes it easier to change the programming language on the server due to same data structure across all such languages.
- **Clean Data**: Makes JSON less susceptible to errors due to standard format as well as enables offering robust cross-browser solutions.
- **Efficiency**: Facilitates transferring only the changed details to client on every request, as the previous output is cloned on client side.

- **Interoperability**: Allows two or more applications coded in a different programming language to communicate with each other smoothly.
- **Extensibility**: Involves storing only standard data such as numbers and text, unlike XML storing any data type. While this is a pro of XML, it is not required for transferring JSON data, which simply denotes simplicity.
- **Internationalization**: Implements the Unicode norm due to which generic tools can easily manipulate JSON structures.

Additional Information:

For more information, visit the following links:
http://www.json.org/xml.html
http://www.revillweb.com/articles/why-use-json/

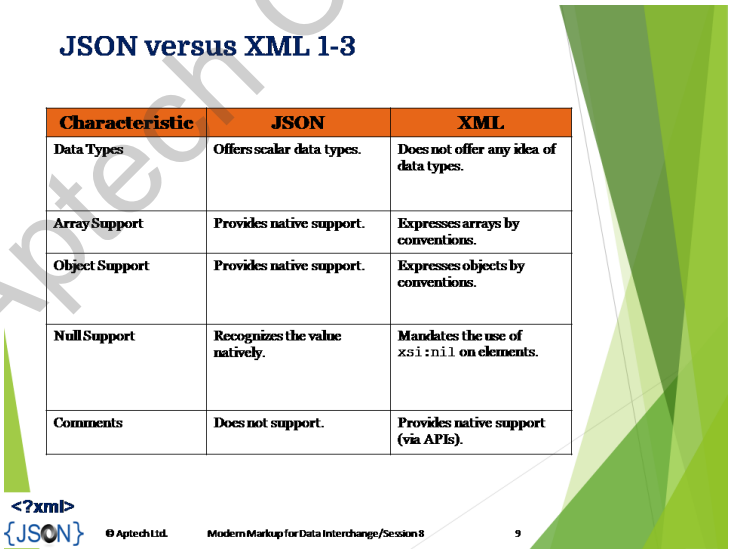
In-Class Question:

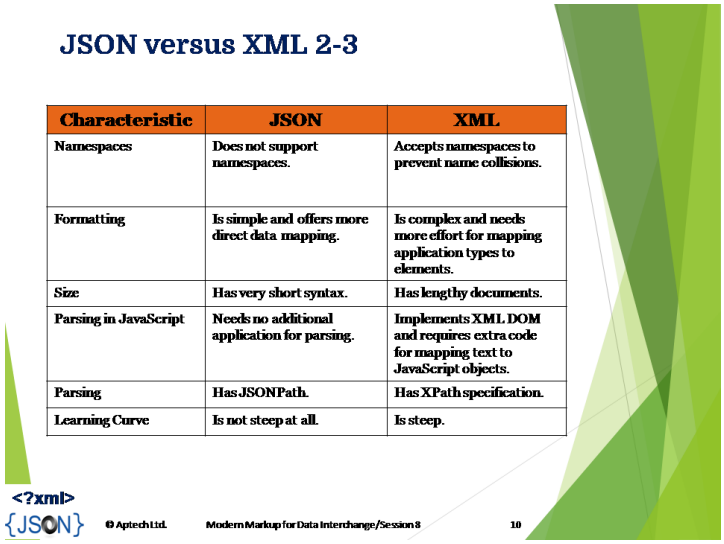
After you finish explaining the features of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

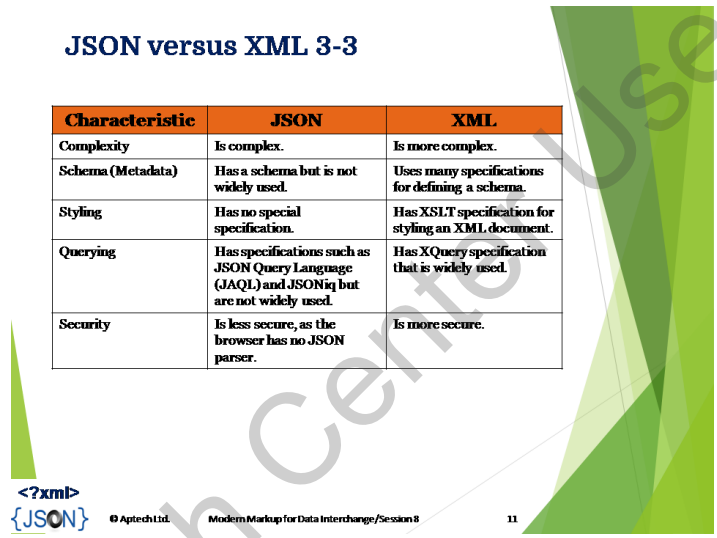
Why JSON is an open-source format?

Answer: JSON is an open-source format as a developer can easily add other elements when needed to customize it as per specific requirements.

Slides 9 to 11
Let us distinguish between JSON and XML.







Using slides 9 to 11, explain to students the differences between JSON and XML in terms of different characteristics.

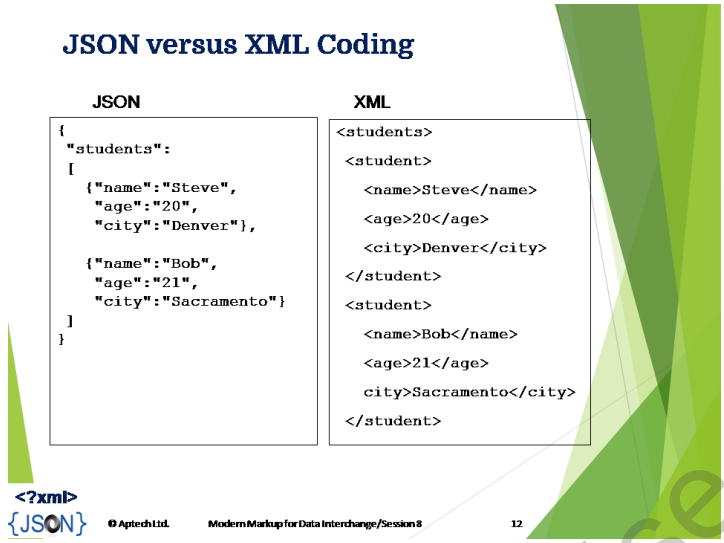
Additional Information:

For more information, visit the following links:

http://codebucket.co.in/which-one-is-better-xml-or-json/
http://www.cs.tufts.edu/comp/150IDS/final_papers/tstras01.1/FinalReport/FinalReport.html

Slide 12

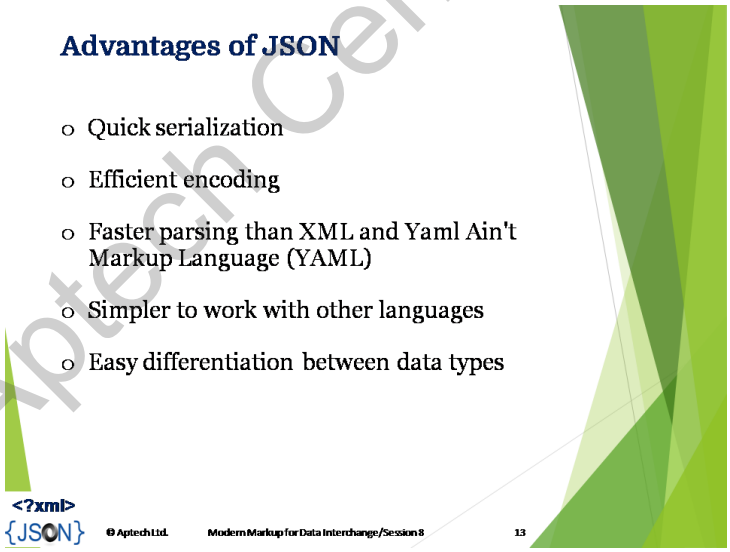
Let us view the difference between JSON code and XML code.



Using slide 12, focus on both the snippets and show students how JSON is lighter and more compact than XML.

Slide 13

Let us explore the advantages of JSON.



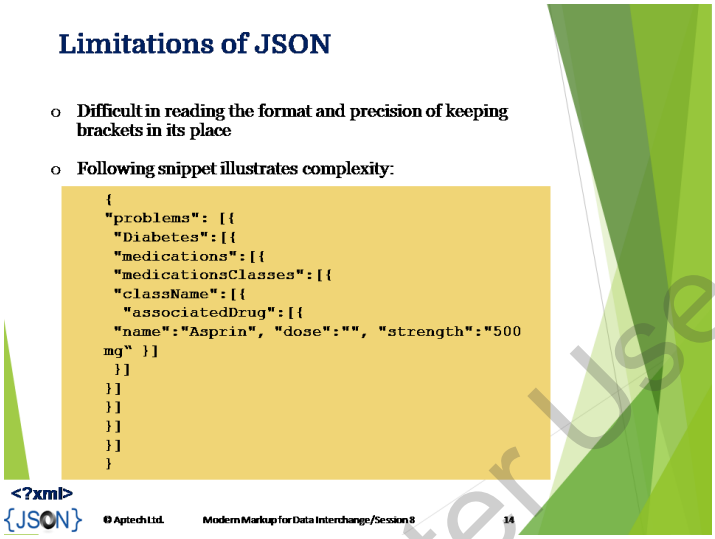
Using slide 13, explain the advantages of using JSON. Tell them that JSON is used for serializing and de-serializing data without involving lots of padding that is prevalent in XML. Further, the JSON structure is quite compact. As a result, the size of JSON documents is smaller than XML documents.

The encoding mechanism is efficient enough to add minimum characters needed to represent the structure and values. Even the parser for YAML that serializes complex data sets without a DTD consumes more time than JSON to give an output of larger streams of serialized data. Further, JSON differentiates between the string '2' and the number '2'.

Additional Information:

For more information, visit the following link:
http://www.json2yaml.com/yaml-vs-json

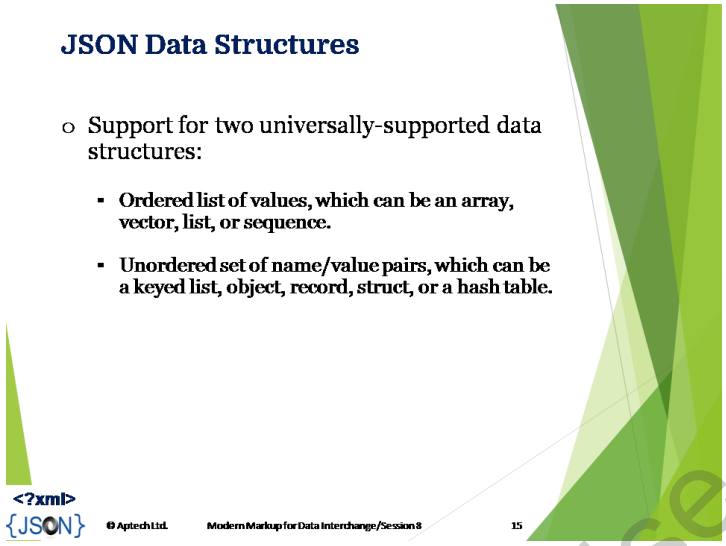
Slide 14
Let us now understand the limitations of JSON.



Using slide 14, explain limitations of JSON to students. Focus on the code and say that the most significant limitation is difficulty in reading the format when there are so many brackets and commas involved.

Slide 15

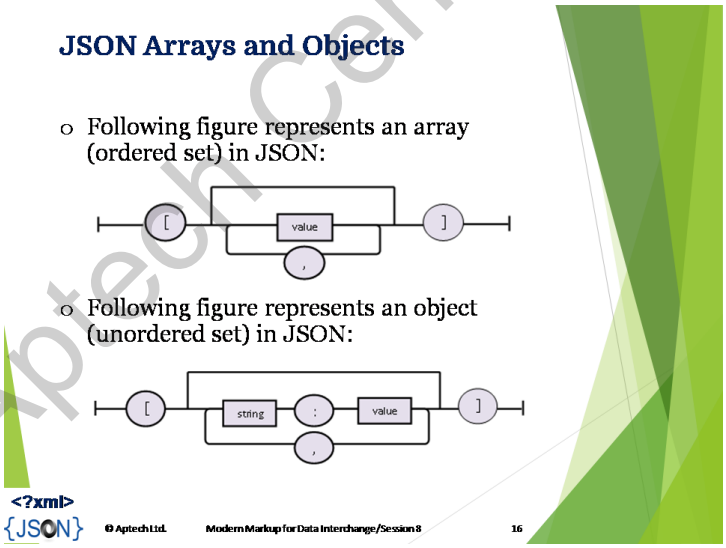
Let us explore the types of data structures supported by JSON.



Using slide 15, explain the two data structures supported by JSON to students. Tell that both of them are supported by almost all programming languages.

Slide 16

Let us now understand JSON arrays and objects.



Using slide 16, explain to students that an ordered set is also called an array, while an unordered set is called an object. Focus on the diagrams to tell them how they are written.

In-Class Question:

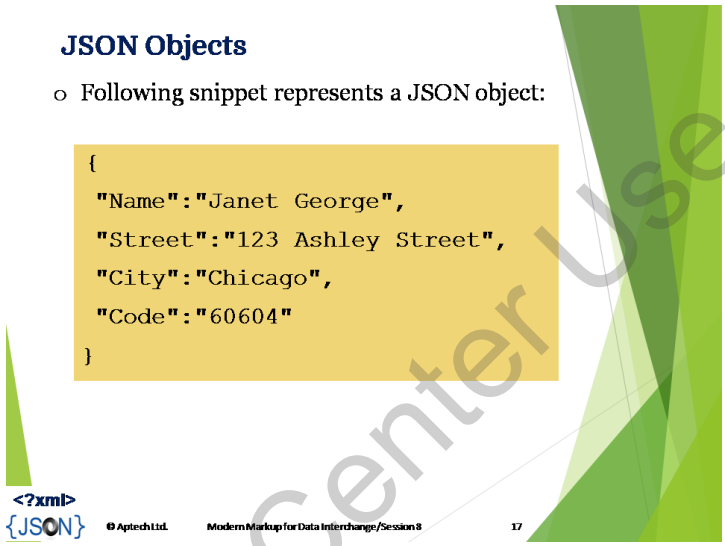
After you finish explaining JSON arrays and objects, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.


Give an example of an array and object data structures.

Answer: A sequenced set holding colors of rainbow is an array, while a set holding details of a book such as name, ISBN no, and price is an object.

Slide 17

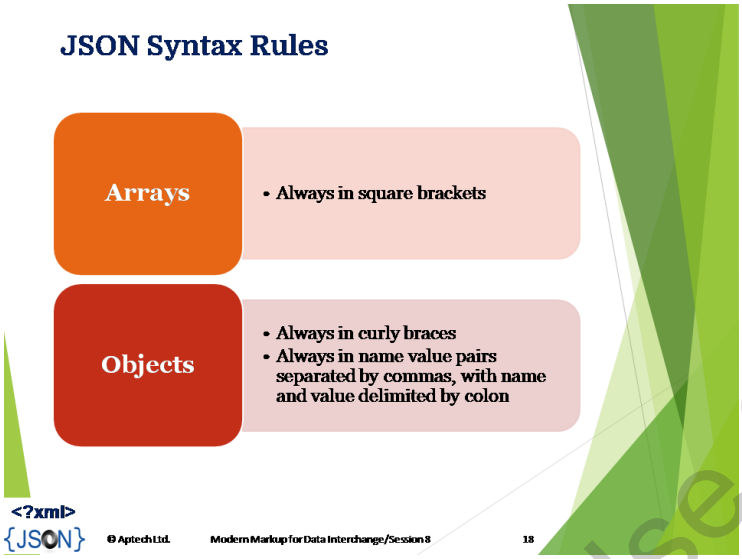
Let us see how to define a JSON object.



Using slide 17, explain how to write a JSON object by focusing on the snippet. Tell them that in an object, a name/value pair is a member. The member name should always be a string in double quotes.

Slide 18

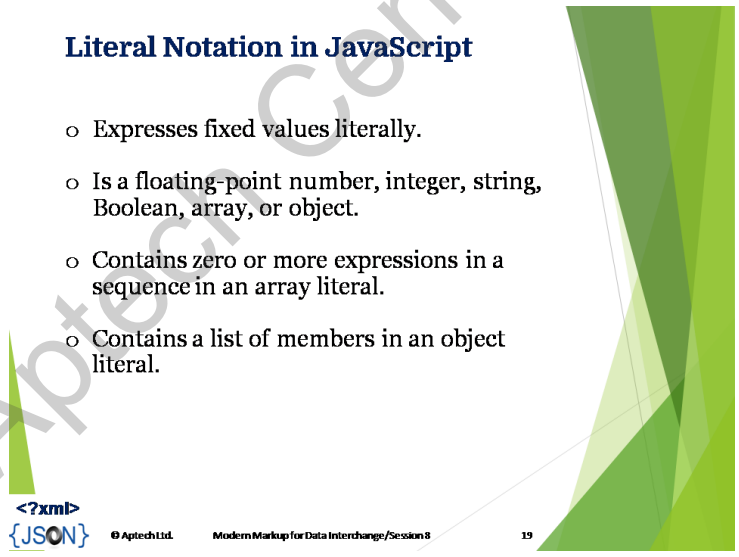
Let us now understand the syntax rules for arrays and objects.



Using slide 18, explain the syntax rules for arrays and objects to students.

Slide 19

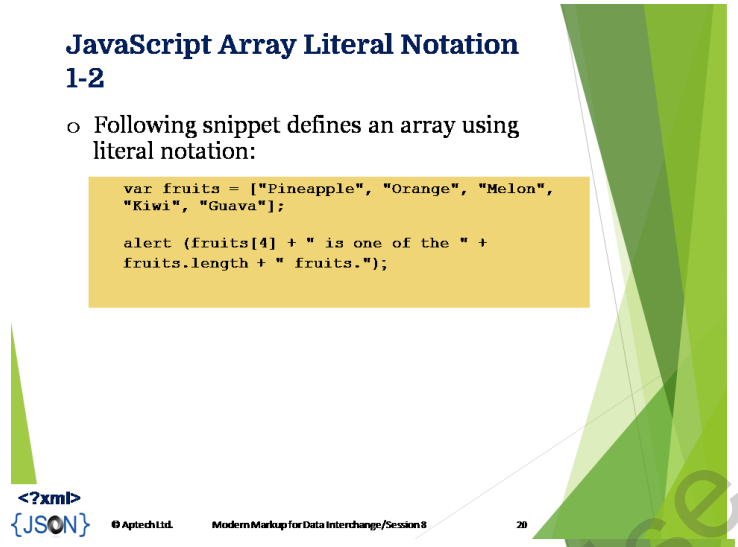
Let us now understand literal notation in JavaScript.

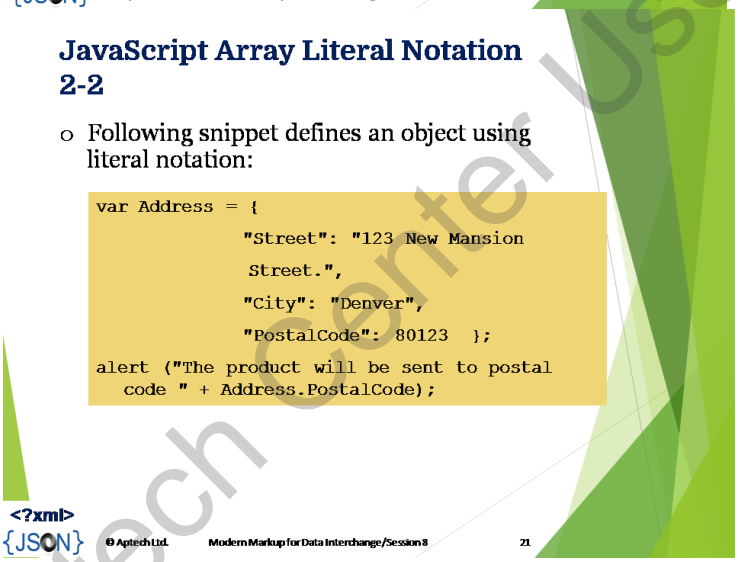


Using slide 19, explain JavaScript literal notation to students. Tell them that in a programming language, literals are fixed values such as a string value “passed” and a constant integer value of 7. The literals are used in most languages for stating an expression. This expression can be an input parameter to a function, a value assigned to a variable, or a part of a condition in a control statement.

Slides 20 and 21

Let us now explore literal notation for arrays and objects in JavaScript.





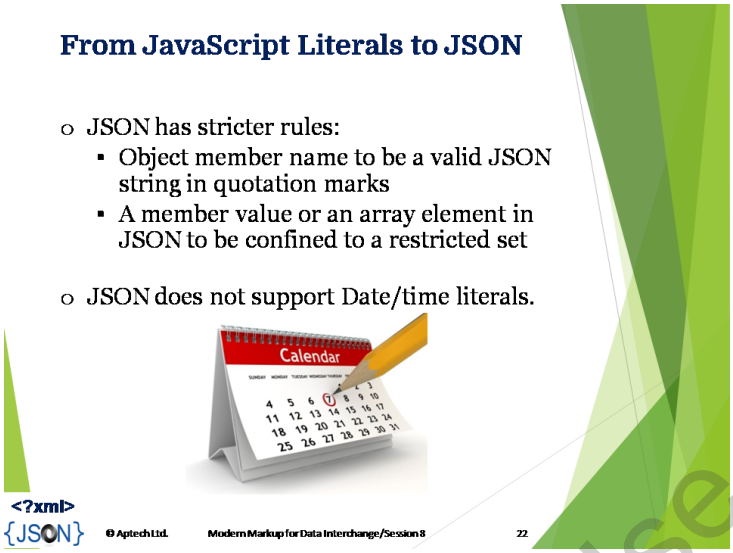
Using slides 20 and 21, explain how literal notation is used to define an array and object in JavaScript.

Additional Information:

For more information, visit the following link:
http://www.crockford.com/javascript/survey.html

Slide 22

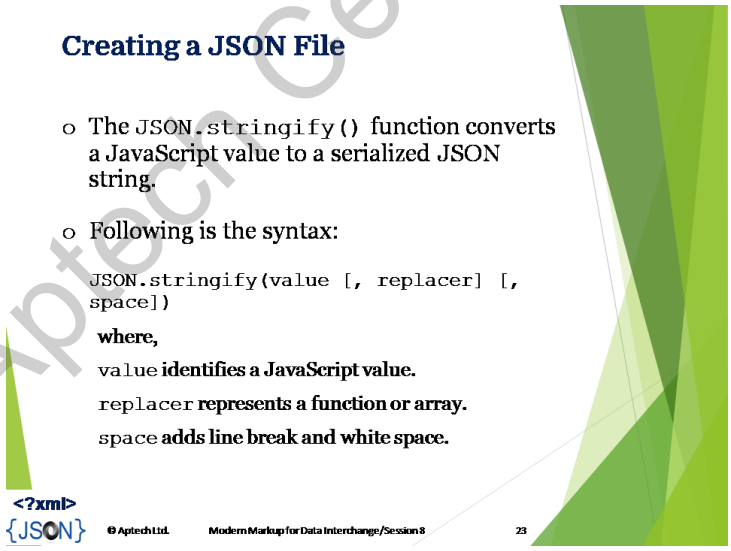
Let us now explore literals in JSON.



Using slide 22, explain about the stricter rules that JSON follows for literals in arrays and objects.

Slide 23

Let us now see how to create a simple JSON file.

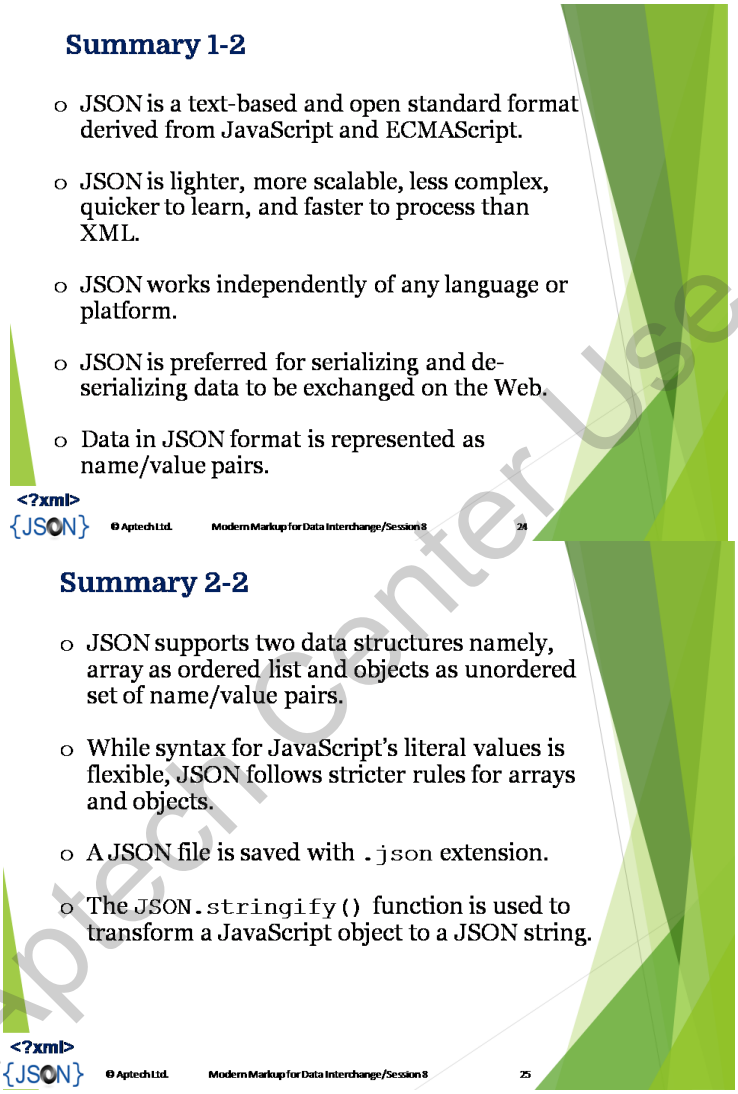


Using slide 23, explain the `JSON.stringify()` function to students. Focus on the syntax.

Additional Information:

For more information, visit the following link:
https://msdn.microsoft.com/library/cc836459(v=vs.94).aspx

Slides 24 and 25
Let us summarize the session.



Using slides 24 and 25, summarize the session. Explain each of the following points in brief. Tell them that:

- JSON is a text-based and open standard format derived from JavaScript and ECMAScript.
- JSON is lighter, more scalable, less complex, quicker to learn, and faster to process than XML.
- JSON works independently of any language or platform.
- JSON is preferred for serializing and de-serializing data to be exchanged on the Web.

- Data in JSON format is represented as name/value pairs.
- JSON supports two data structures namely, array as ordered list and objects as unordered set of name/value pairs.
- While syntax for JavaScript's literal values is flexible, JSON follows stricter rules for arrays and objects.
- A JSON file is saved with .json extension.
- The `JSON.stringify()` function is used to transform a JavaScript object to a JSON string.

8.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 9: Work with JSON Data

9.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

9.1.1 Objectives

By the end of this session, learners will be able to:

- Identify data types supported by JSON
- Explain how to represent complex data with JSON
- Explain how to execute serialization and de-serialization of JSON with JavaScript
- Describe tools and editors that can be used to work with JSON
- Describe the syntax and schema of a JSON document

9.1.2 Teaching Skills

To teach this session, you should be well versed with the different data types used in JSON and their syntax. You should be familiar with various data structures supported by JSON, JSON schema and its creation, and various JSON tools and editors.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

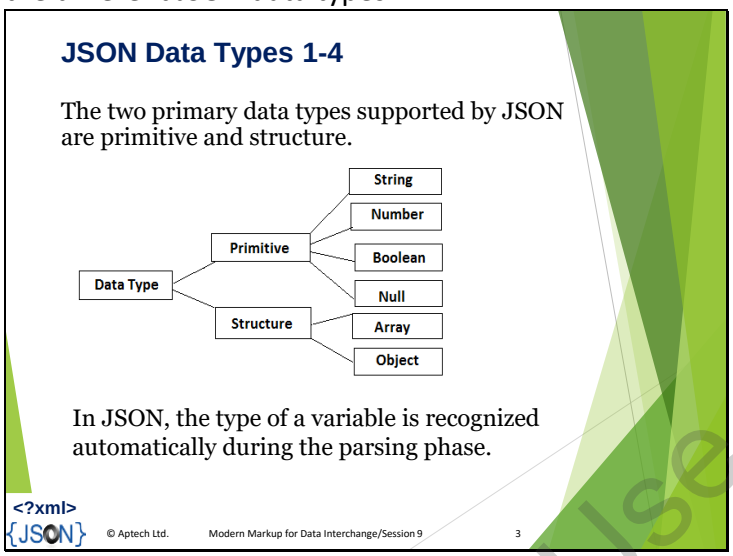
Overview of the Session:

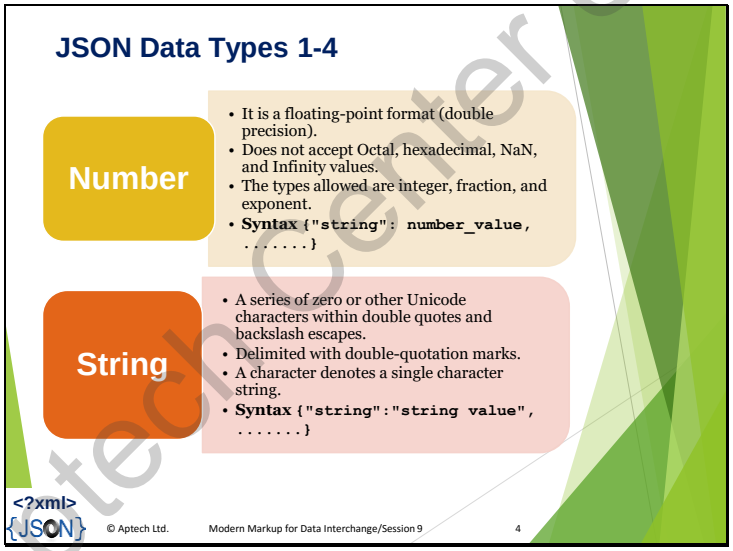
Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

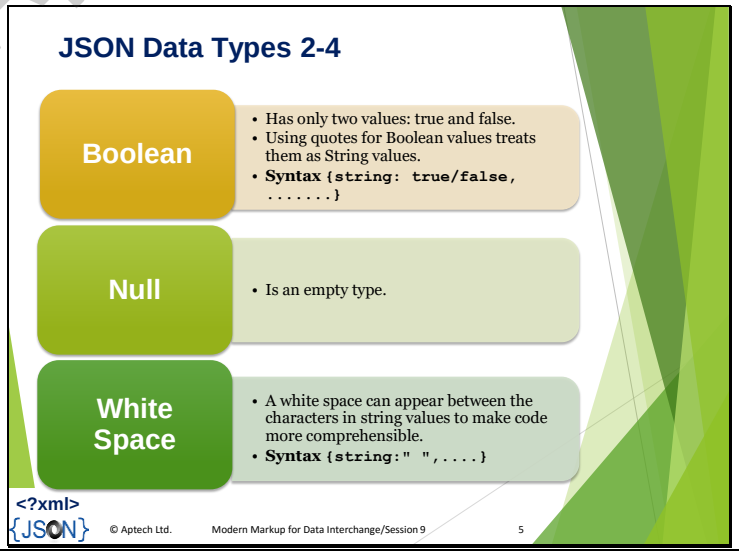
9.2 In-Class Explanations

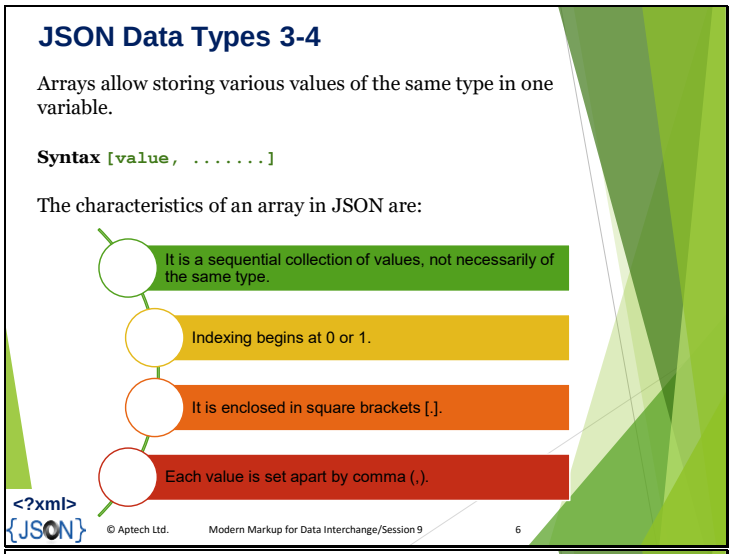
Slides 3 to 7

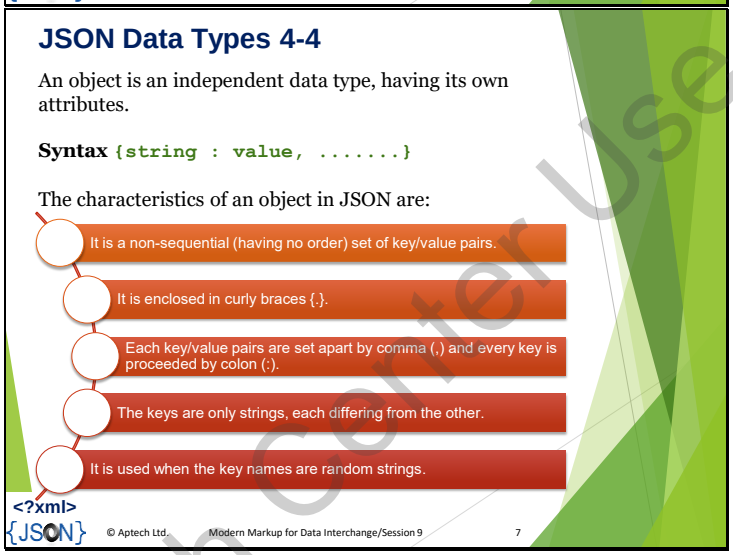
Let us understand the different JSON data types.











Using slides 3 to 7, explain the classification of JSON data types. For each data type, list the characteristics and explain the syntax.

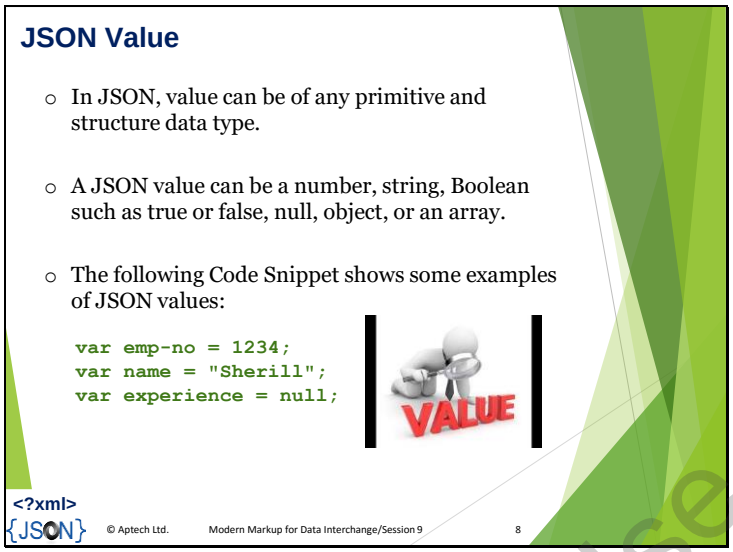
Additional Information:

For more information, visit the following links:

https://www.postgresql.org/docs/9.4/static/datatype-json.html
http://www.tutorialspoint.com/json/json_data_types.htm

Slide 8

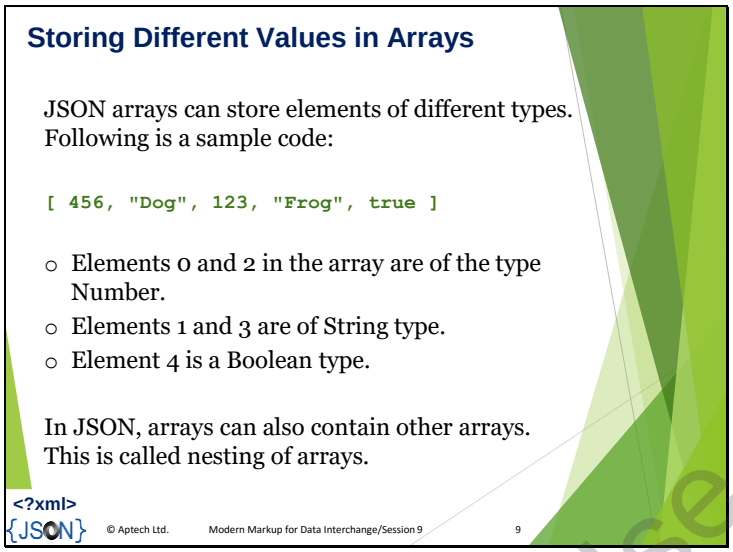
Let us understand about JSON value.



Using slide 8, explain how the data types are used. Explain some examples given on the slide.

Slide 9

Let us now understand how to store different values in arrays.



Using slide 9, explain to the students that JSON not only allows storing different values in arrays and objects but also enables you to nest them. To make it easily understandable for the students, you can explain the concept using the example provided on the slide.

Using the example given in Figure 9.2 in the Learner Guide, explain that in JSON, arrays can also contain other arrays and this is called nesting of arrays.

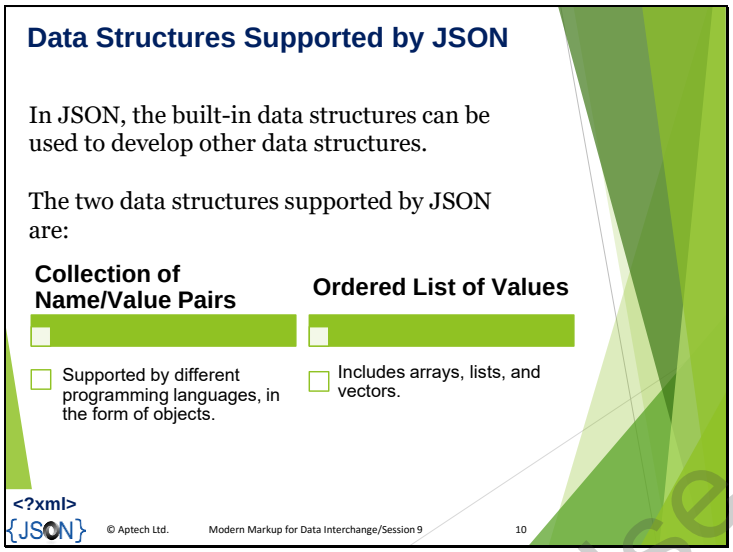
Using the following example, explain the concept of nesting array in a JSON Object. Explain that the code shows a JSON blog object storing user details along with an array of tags. The array contains the tag details of blog contents in an array. These tag details help in searching the content on a blog.

```
{
 "blogID":12345,
 "firstName":"James",
 "lastname":"Gunn",

"blogContent":"This is my first JSON object and it looks simpl
e",
 "tags":[
 "first",
 "JSON",
 "looks",
 "simple"
 ]
}
```

Slide 10

Let us understand about the data structures supported by JSON.



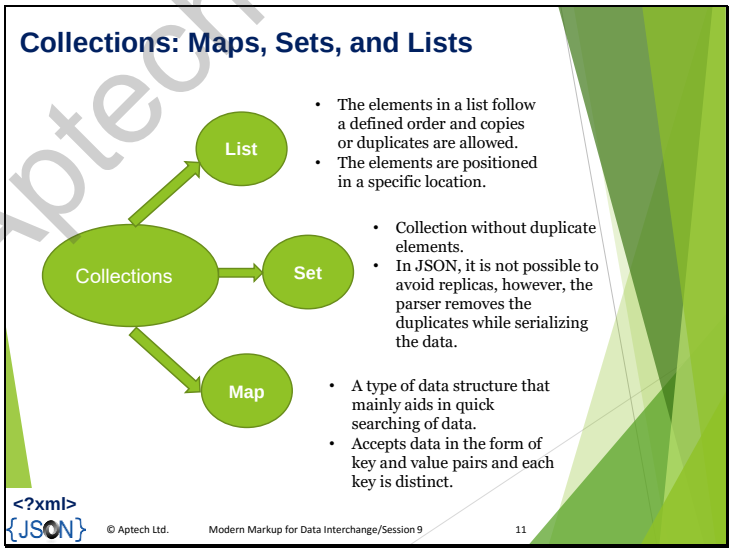
Using slide 10, tell the students that data structures are an integral part of all the programming languages. Explain the two data structures supported by JSON.

Additional Information:

For more information, visit the following link:
https://adobe.github.io/Spry/samples/data_region/JSONDataSetSample.html

Slide 11

Let us understand about collections.

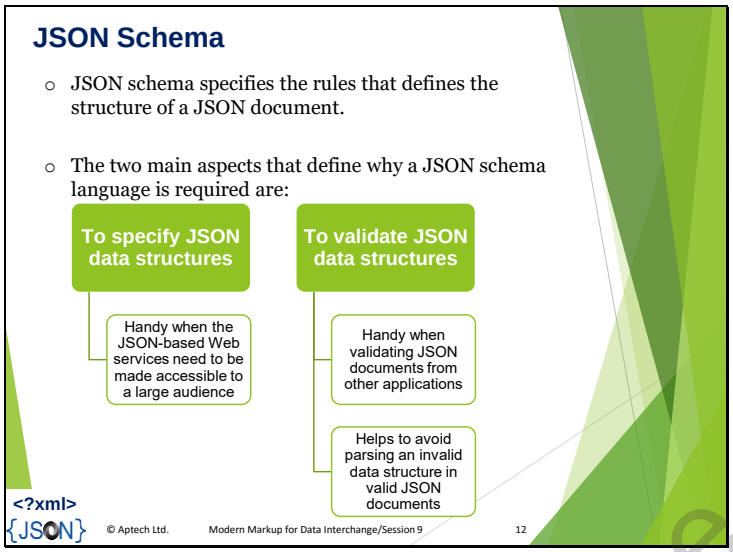


Using slide 11, explain that sets and lists correspond to a set of objects, whereas maps relate a value to an object. Provide a description of each of these data structures and explain their features.

Inform the students that it is the parser that takes up the main role of serializing and de-serializing JSON data. The JSON parser maps the core data structure with the internal JSON structure.

Slide 12

Let us now understand about JSON Schema.



Using slide 12, explain JSON Schema to the students and the two main aspects that define why a JSON schema language is required.

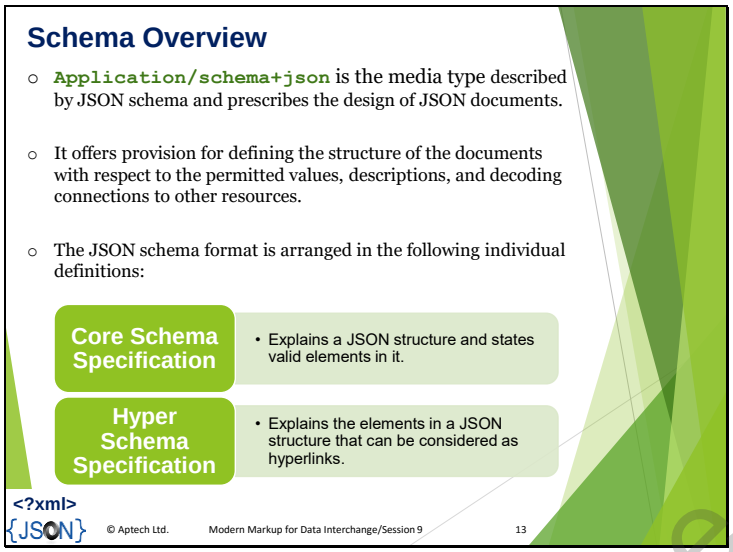
As additional knowledge, you can tell the students that JSON schema is based on the XML Schema Definition (XSD) and is just an Internet draft; the latest version 4 expired on August 4th, 2013. Though it has expired, application developers still use the concept for generating JSON document, which is a contract between different applications or producers and consumers of Web services.

Additional information:

For more information, visit the following link:
http://json-schema.org/documentation.html

Slide 13

Let us understand more about the Schema.

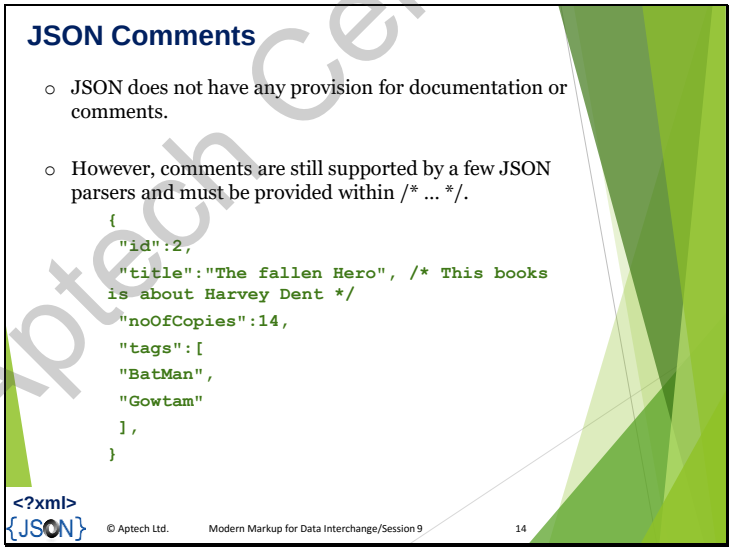


Using slide 13, provide details about the schema and how the JSON schema format is arranged.

At this stage, you can convey the concept of a schema using a Library Management System (LMS) as an example.

Slide 14

Let us understand about comments.

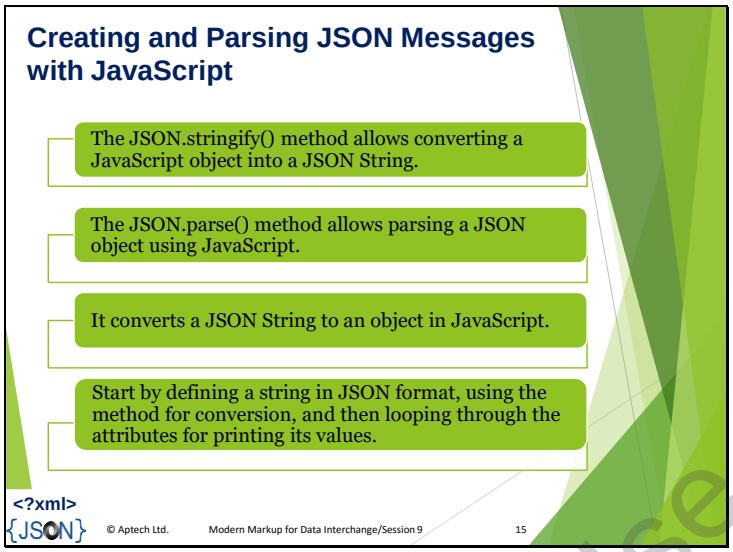


Using slide 14, introduce comments to the students. Explain that comments are not supported, as its developer Douglas Crockford found users utilizing them to store parsing directives.

However, you can still use a workaround to give comments within a JSON document, although you need to delete all before a parser processes it.

Slide 15

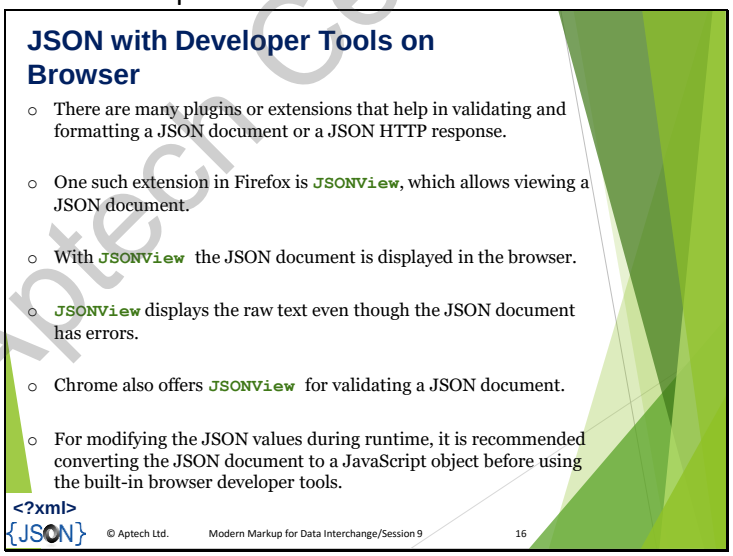
Let us now understand how to create and parse JSON messages with JavaScript.



Using slide 15, explain how to create and parse JSON messages with JavaScript. Use relevant code snippets to demonstrate how a JavaScript object is converted to a JSON string using the `JSON.stringify()` method and how a JSON object is converted to a JavaScript object in an HTML file using the `JSON.parse()` method.

Slide 16

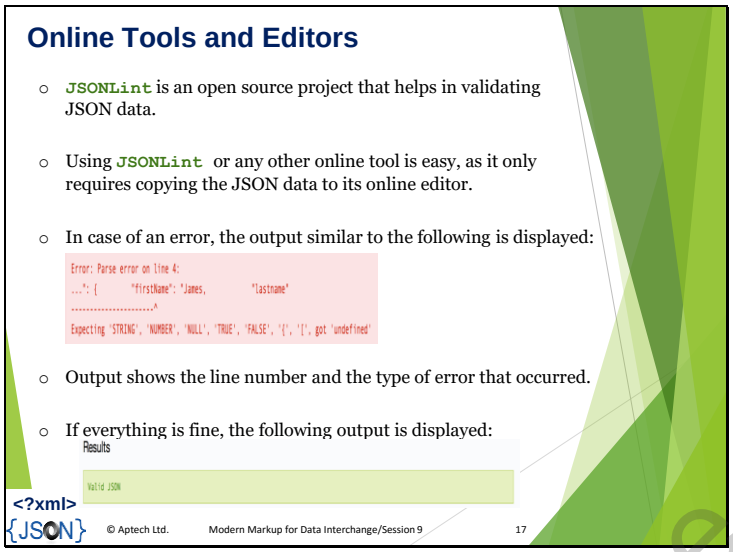
Let us understand about developer tools on browser.



Using slide 16, give details about the various plugins or extensions that help in validating and formatting a JSON document or a JSON HTTP response. You can show how JSONView in Firefox and Chrome allow to view a JSON document.

Slide 17

Let us understand about online tools and editors.



Using slide 17, explain that there are many online tools and editors for validating JSON data and the most popular online validator is JSONLint. You can demonstrate how this works by copying the JSON data to any online editor or asking the students to test this for themselves whenever they have access to a browser.

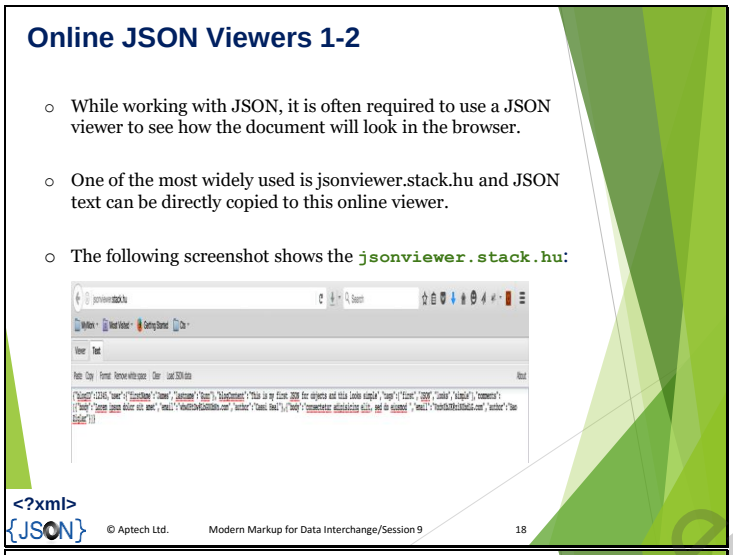
Additional Information:

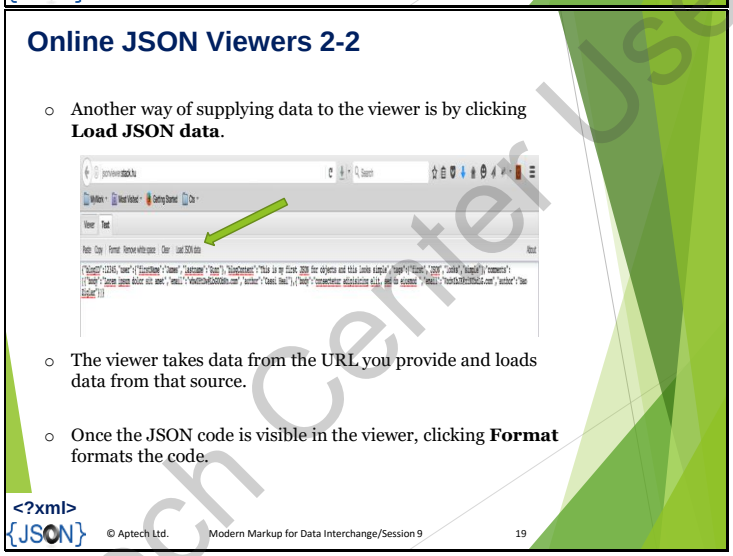
For more information, visit the following link:

http://jsonlint.com/

Slides 18 and 19

Let us understand about online JSON viewers.

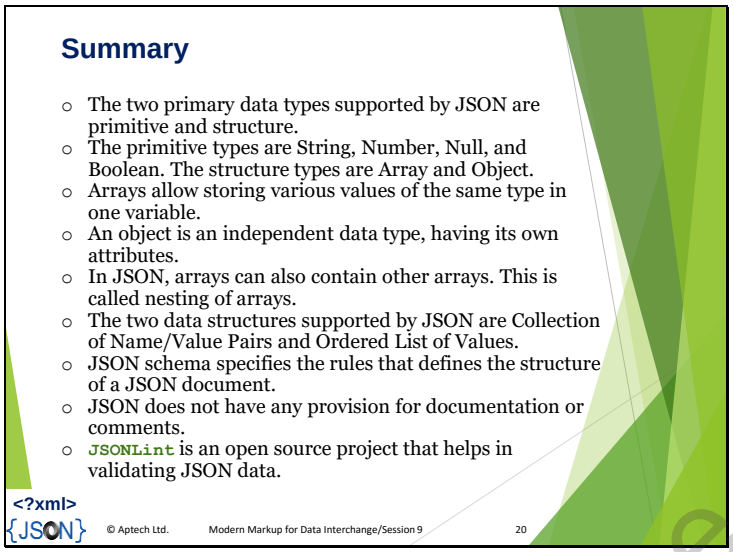




Using slides 18 and 19, explain about the jsonviewer.stack.hu tool which helps to view how the document looks in the browser. As it is an online tool, you can practically show the students how it works.

Slide 20

Let us summarize the session.



Using slide 20, summarize the session. Explain each of the following points in brief. Tell them that:

- The two primary data types supported by JSON are primitive and structure.
- The primitive types are String, Number, Null, and Boolean. The structure types are Array and Object.
- Arrays allow storing various values of the same type in one variable.
- An object is an independent data type, having its own attributes.
- In JSON, arrays can also contain other arrays. This is called nesting of arrays.
- The two data structures supported by JSON are Collection of Name/Value Pairs and Ordered List of Values.
- JSON schema specifies the rules that defines the structure of a JSON document.
- JSON does not have any provision for documentation or comments.
- **JSONLint** is an open source project that helps in validating JSON data.

9.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session.

Tips: You can also check the **Articles/Blogs/Expert Videos** uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session.

Session 10: JSON in Real World

10.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two that will be a key point to relate the current session objectives.

10.1.1 Objectives

By the end of this session, learners will be able to:

- Describe the support of different browser and programming languages for JSON
- Describe JSON content types
- Explain how to use JSON for Web and Data Storage
- Compare JSON with relational databases
- Identify security and data portability issues with respect to JSON

10.1.2 Teaching Skills

To teach this session, you should be well versed with support for JSON by different browsers and programming languages, especially C# and Java. You should be familiar with the JSON content types, using JSON for Web and data storage, and differences between JSON and relational as well as NoSQL databases. You should also be aware of security and portability issues in JSON.

You should teach the concepts in the theory class using the images provided. For teaching in the class, you are expected to use slides and LCD projectors.

Tips:
It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities

Follow the order given here during In-Class activities.

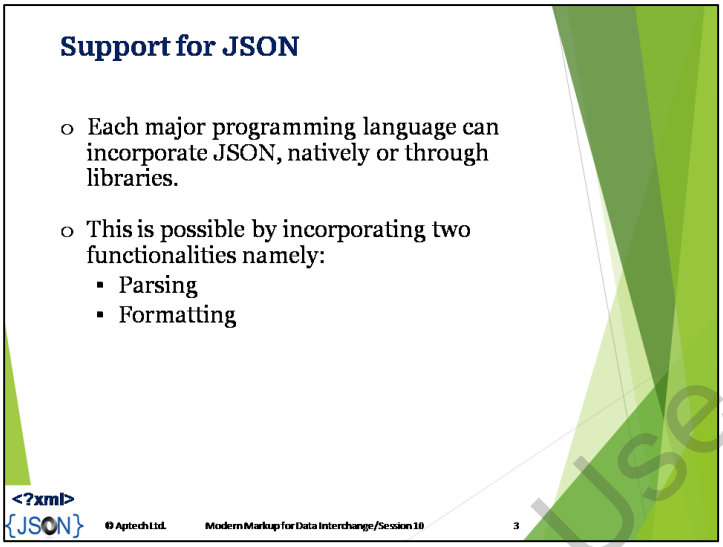
Overview of the Session:

Give the students an overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

10.2 In-Class Explanations

Slide 3

Let us understand the support for JSON in different programming languages.



Using slide 3, explain how JSON is supported in different programming languages through parsing and formatting functionalities. Tell students that any major programming language can incorporate JSON either through libraries or native support. However, none of them can bypass JavaScript for parsing JSON, which converts it into an object literal.

Each major language has powerful libraries for parsing and formatting data in JSON format. Parsing converts JSON data into the data structure supported by the corresponding language. This structure can be an array, hash, or dictionary. Formatting converts array, hash, or dictionary to JSON text.

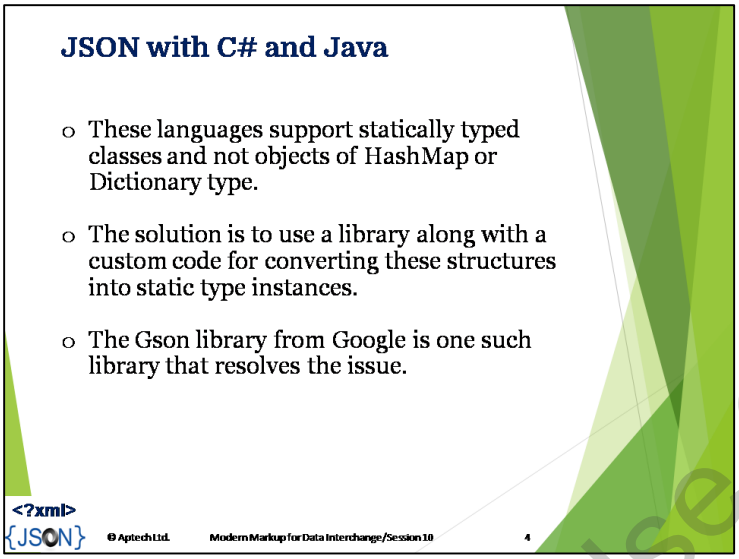
Additional Information:

For more information, visit the following link:

https://www.quora.com/With-which-programming-language-does-JSON-pair-best

Slide 4

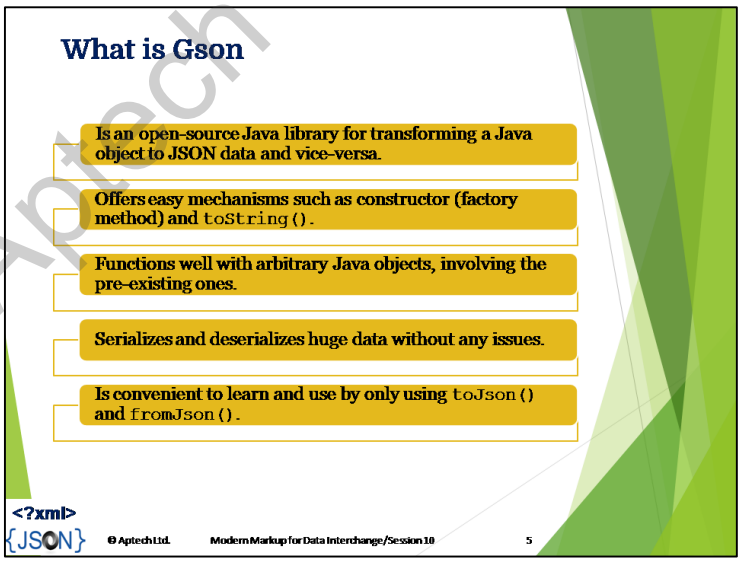
Let us understand the use of JSON with C# and Java.



Using slide 4, tell students that a few developers do not feel it feasible to use JSON in C# or Java. This is because the languages support statically typed classes instead of Dictionary or HashMap objects. Thus, developers would need a library for converting these structures to static instances. One such library is Gson from Google, which transforms JSON data to Java objects.

Slide 5

Let us now understand Gson.

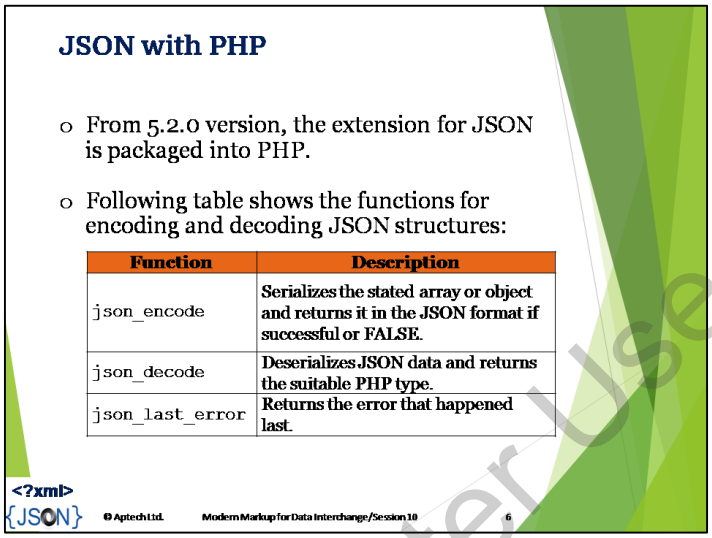


Using slide 5, briefly explain Gson to students. Finally, tell them that the goals of Gson are to convert already existing objects to and from JSON, allow custom representations for objects, and giving readable and compact JSON data.

Additional Information:

For more information, visit the following link:
https://sites.google.com/site/gson/gson-user-guide

Slide 6
Let us see how JSON is used in PHP.



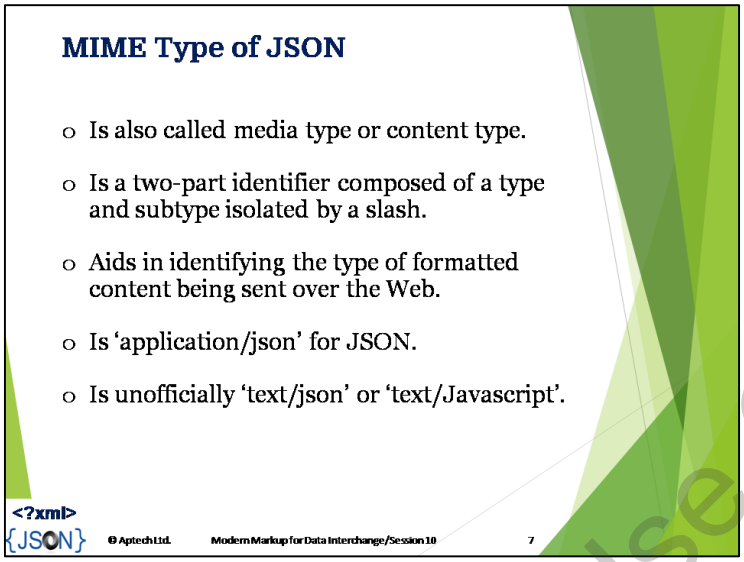
Using slide 6, explain the functions used in PHP for encoding and decoding JSON structures.

Additional Information:

For more information, visit the following links:
http://php.net/manual/en/book.json.php
http://www.phpbuilder.com/articles/application-architecture/object-oriented/processing-json-in-php.html

Slide 7

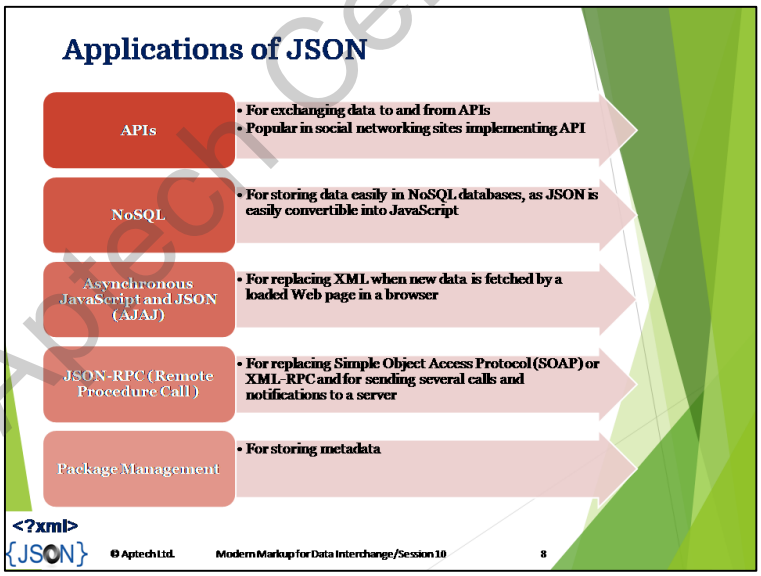
Let us now explore the MIME or content type of JSON.



Using slide 7, explain MIME type to students. Focus on its definition and its official as well as unofficial form for JSON.

Slide 8

Let us now explore the applications of JSON.



Using slide 8, explain various applications of JSON to students. Tell them about NoSQL databases in brief. Then, state that AJAJ is similar to AJAX but uses JSON and not XML. AJAJ, as a dynamic Web development methodology, allows a Web page to ask and display new data. Next, the SOAP protocol defines some commands and data types for allowing a system to send multiple calls and notifications to the server. Packages make it simpler to deploy the applications and integrate any changes, later. Many package management tools

implement the `package.json` file having metadata such as version, name, description, and license details.

Additional Information:

For more information, visit the following link:
http://www.w3resource.com/JSON/introduction.php

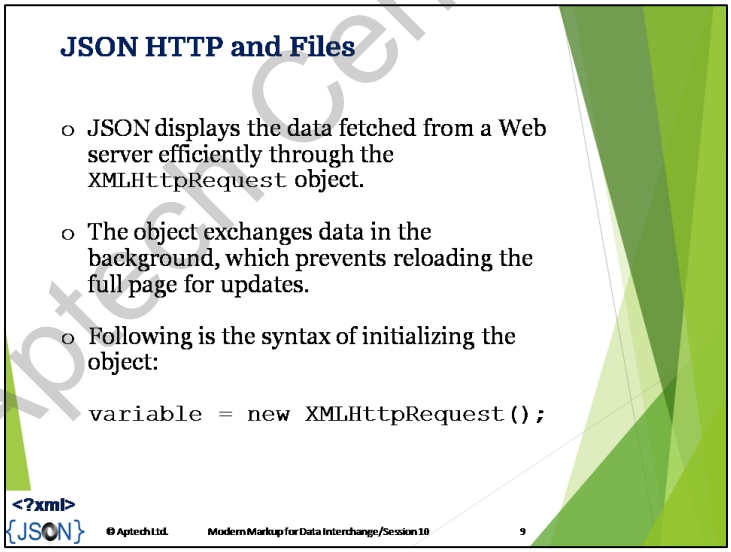
In-Class Question:

After you finish explaining the applications of JSON, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Why it is easy to work with JSON for storing data into a database?

Answer: It is easy to work with JSON for storing data into a database because a developer can convert a JSON structure to a compatible object structure in the browser. This converted structure is also easy to integrate in Server-side script for storing the data in the server's database.

Slide 9
Let us now understand how JSON is useful in displaying data fetched from a Web server on a Web page.



Using slide 9, explain how JSON uses the `XMLHttpRequest` object to display the data from a Web server. Focus on the syntax.

Additional Information:

For more information, visit the following links:
http://enterprisewebbook.com/ch2_ajax_json.html
http://www.tutorialspoint.com/ajax/what_is_xmlhttprequest.htm

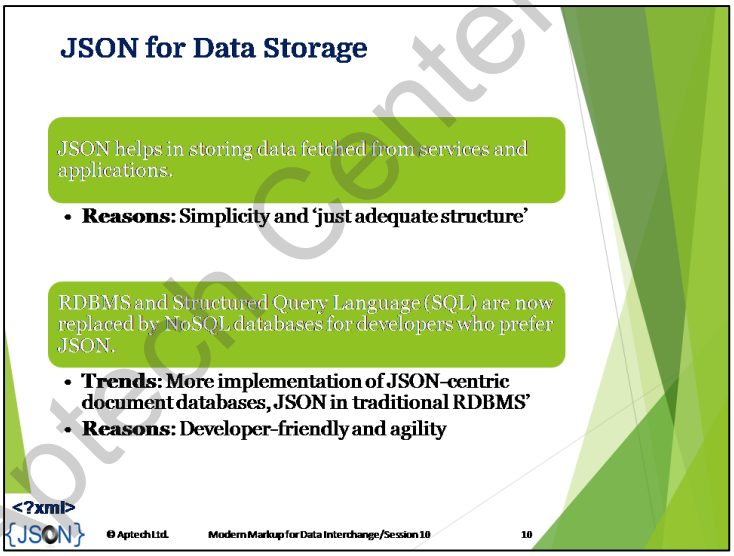
In-Class Question:

After you finish explaining how JSON is useful in displaying data fetched from a Web server, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

Why the `XMLHttpRequest` object is efficient?

Answer: The `XMLHttpRequest` object is efficient, as it exchanges data with the server in the background such that there is no need to reload the full page for updating just a few sections.

Slide 10
Let us understand why JSON is used to store data.

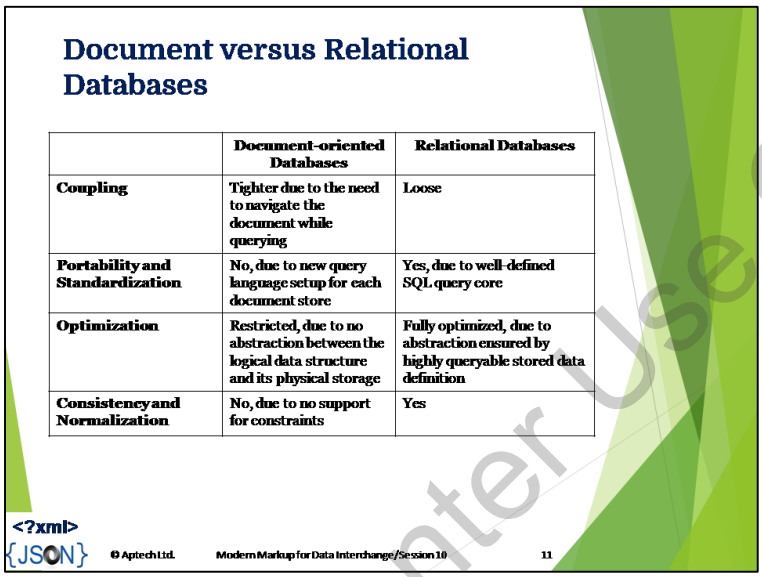


Using slide 10, explain why JSON is useful even in storing data. The long control of RDBMS and SQL has come to an end with NoSQL databases, which are catering to the needs of developers who prefer JSON. Database vendors have also started offering JSON-centric document databases. Many NoSQL database vendors prefer JSON over conventional relational schemes in Oracle, MySQL, MS SQL Server, and PostgreSQL. This is ideal for those who want JSON data interchange format for their applications. JSON records are easily extensible and legibly structured, making the format agile. Further, the absence of predefined schema enables easier upgrades.

Additional Information:

For more information, visit the following link:
http://www.infoworld.com/article/2608293/nosql/how-json-sparked-nosql----and-will-return-to-the-rdbms-fold.html

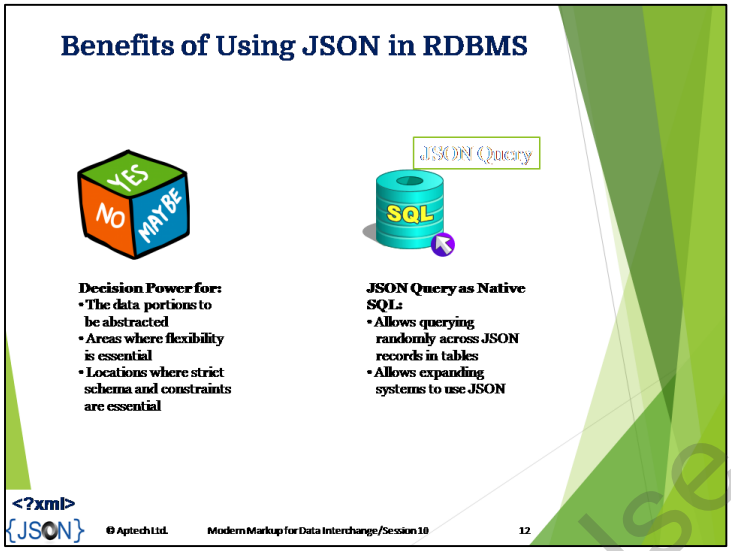
Slide 11
Let us now compare document and relational databases.



Using slide 11, explain the differences between document and relational databases.

Slide 12

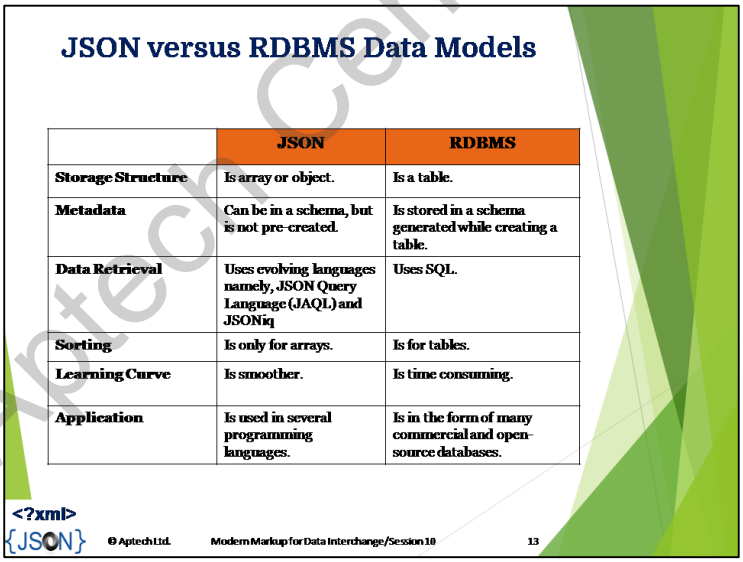
Let us explore the benefits of using JSON in an RDBMS.



Using slide 12, explain the benefits of using JSON in an RDBMS.

Slide 13

Let us now distinguish between JSON and RDBMS.



Using slide 13, explain the differences between JSON and RDBMS to students. Tell them that JSON has a much smooth learning curve, as the supported structures and data types are used in several programming languages. This makes it quick for a developer with basic programming background to understand JSON concepts and coding syntax.

In-Class Question:

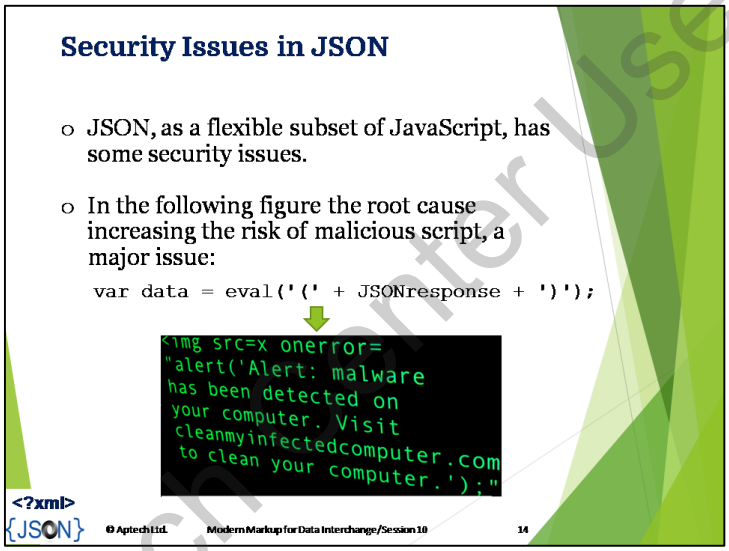
After you finish explaining the distinction between JSON and RDBMS, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

 Why is the learning curve for JSON easier than an RDBMS?

Answer: The learning curve for JSON is easier than an RDBMS because the data types and structure of JSON are supported and used in several programming languages. This makes it quicker for a developer to grasp JSON concepts. However, RDBMS is a distinct and vast field to understand.

Slide 14

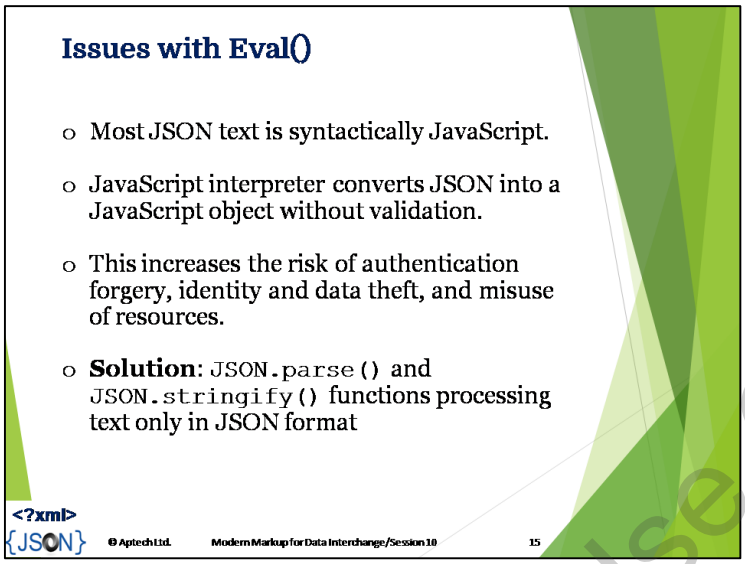
Let us now explore security issues with JSON.



Using slide 14, tell students that JSON code or data has some security issues. A major issue is the increasing risk of malicious script being injected inside the code. Focus on the figure given on the slide.

Slide 15

Let us understand the issue with the `eval()` function.



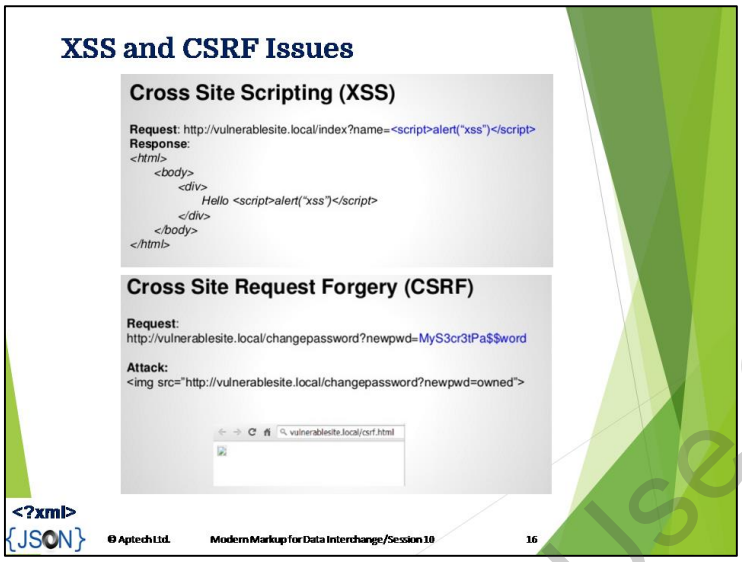
Using slide 15, explain the major security issue of malicious script is triggered by using the `eval()` function of JavaScript. Most JSON text is JavaScript in terms of syntax due to which the function parses JSON data. This means there is no JSON-based parser used and that directly the JavaScript interpreter executes the JSON code for generating JavaScript objects. This is actually risky because there is no validation happening before parsing. Such data is thus vulnerable to security issues. For instance, it can be exposed to malicious code injection attacks for authentication forgery and data theft. Finally, focus on the solutions. `JSON.parse()` processes text only in JSON format.

Additional Information:

For more information, visit the following links:
https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse
http://stackoverflow.com/questions/17785592/difference-between-json-stringify-and-json-parse

Slide 16

Let us now explore the security breaches to which JSON code or format can be easily exposed.



Using slide 16, explain students about two major security breaches namely, Cross-site Scripting (XSS) and Cross-site Request Forgery (CSRF). XSS involves inserting random HTML (along with JavaScript) into Web pages for usually bypassing access control mechanisms. In CSRF, unauthorized commands are sent by a user that seems to be authorized to the site.

Additional Information:

For more information, visit the following link:
http://flask.pocoo.org/docs/0.11/security/

In-Class Question:

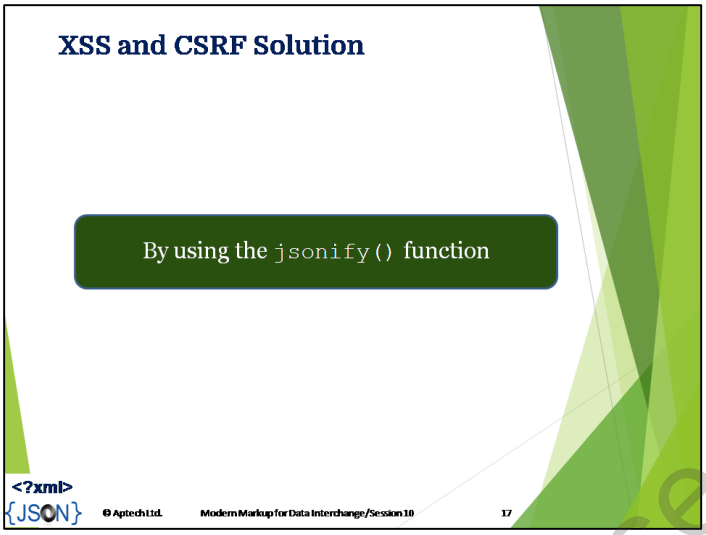
After you finish explaining the security breaches, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.

What is the difference between XSRF and XSS?

Answer: In XSS, the user's trust for a specific site is exploited. However, in CSRF, the site's trust for a user's browser is exploited.

Slide 17

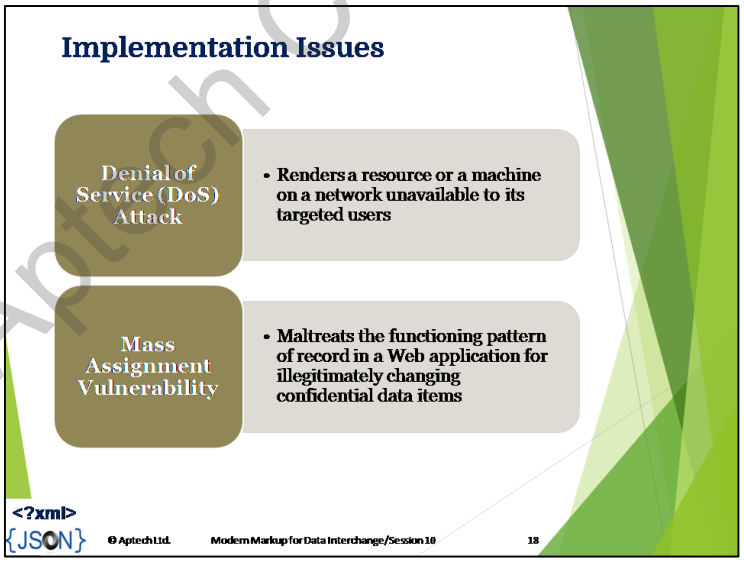
Let us now see the solution for the two security breaches.



Using slide 17, tell students that a strange section of JavaScript specification can keep these breaches away but only if the `jsonify()` function is used for generating JSON structures. Well, this means that the risk still exists if other methods are used for generating JSON structures.

Slide 18

Let us now see some implementation issues related to JSON.

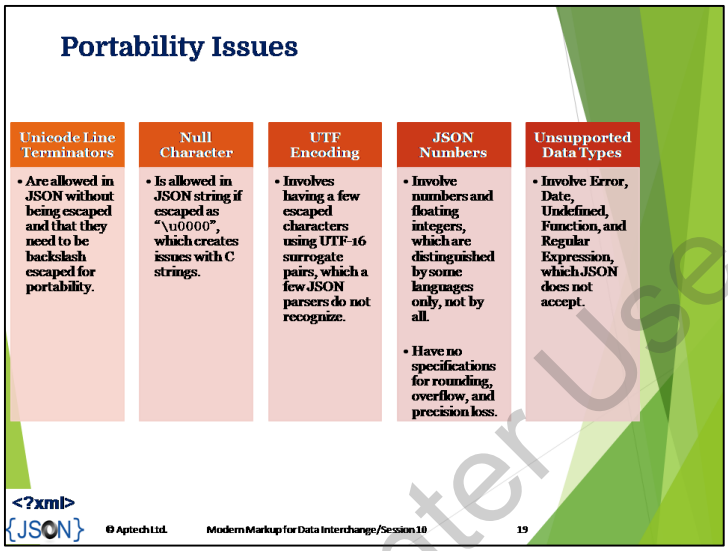


Using slide 18, explain the two implementation issues with JSON to students.

Additional Information:

For more information, visit the following links:
https://en.wikipedia.org/wiki/Denial-of-service_attack
https://en.wikipedia.org/wiki/Mass_assignment_vulnerability

Slide 19
Let us now see some portability issues related to JSON.



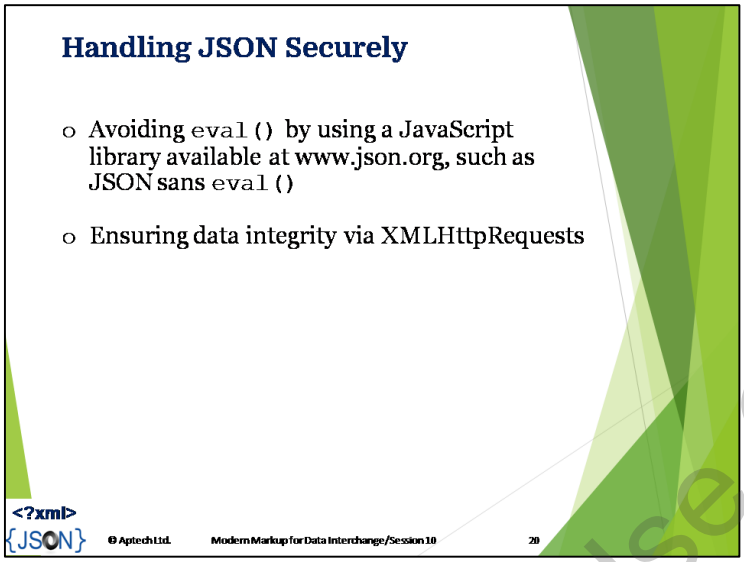
Using slide 19, explain the five portability issues with JSON to students.

Additional Information:

For more information, visit the following link:
https://www.reddit.com/r/Unity3D/comments/30cles/managing_a_large_number_of_small_levels_in_using/

Slide 20

Let us now see how to handle JSON securely.



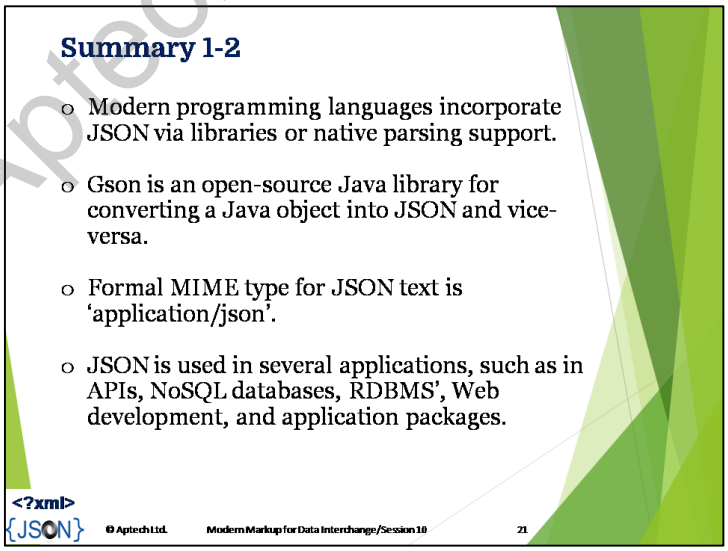
Using slide 20, explain the students how to handle JSON securely by avoiding the `eval()` function and using `XMLHttpRequests`.

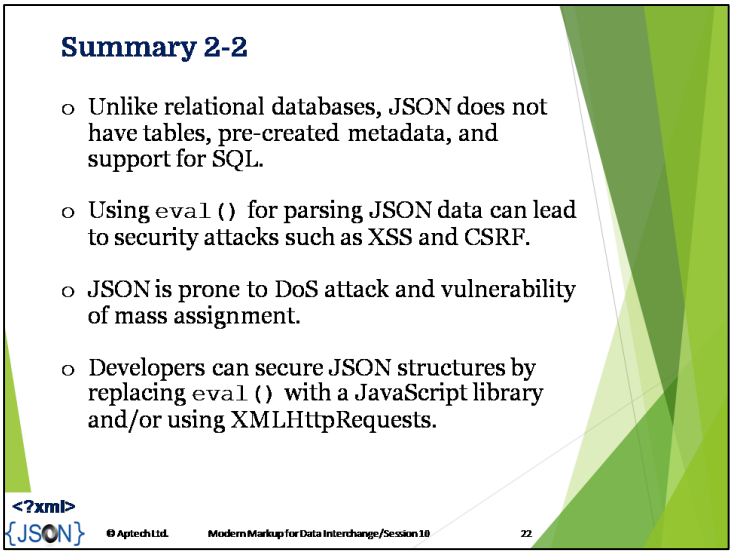
Additional Information:

For more information, visit the following link:
http://yuiblog.com/blog/2007/04/10/json-and-browser-security/

Slides 21 and 22

Let us summarize the session.





Using slides 21 and 22, summarize the session. Explain each of the following points in brief. Tell them that:

- Modern programming languages incorporate JSON via libraries or native parsing support.
- Gson is an open-source Java library for converting a Java object into JSON and vice-versa.
- Formal MIME type for JSON text is 'application/json'.
- JSON is used in several applications, such as in APIs, NoSQL databases, RDBMS', Web development, and application packages.
- Unlike relational databases, JSON does not have tables, pre-created metadata, and support for SQL.
- Using `eval()` for parsing JSON data can lead to security attacks such as XSS and CSRF.
- JSON is prone to DoS attack and vulnerability of mass assignment.
- Developers can secure JSON structures by replacing `eval()` with a JavaScript library and/or using `XMLHttpRequests`.

10.3 Post Class Activities for Faculty

Tips: You can check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the session.