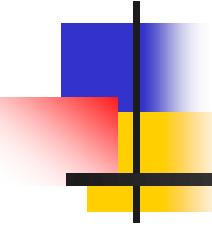


Module 1

Introduction to XML

For Aptech Center Use Only

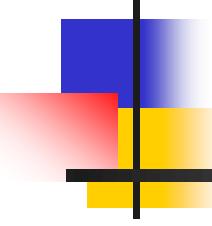


Module Overview

In this module, you will learn about:

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

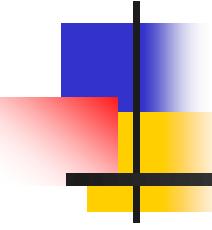
For Aptech Center USE Only



Lesson 1 – Introduction to XML

In this first lesson, **Introduction to XML**, you will learn to:

- Outline the features of markup languages and list their drawbacks.
- Define and describe XML.
- State the benefits and scope of XML.

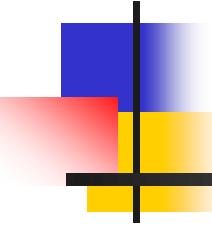


Features of Markup Languages 1-3

- Generalized Markup Language (GML) helps the documents to be edited, formatted, and searched by different programs using its content-based tags.
- Standard Generalized Markup Language (SGML) is a coding scheme for developing specialized markup languages.
- Hyper Text Markup Language (HTML) is used for sharing information by using hyperlinked text documents.

Features of Markup Languages 2-3





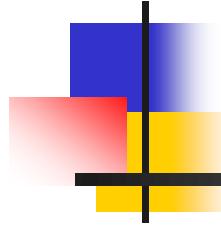
Features of Markup Languages 3-3

Features

- GML describes the document in terms of its format, structure and other properties.
- SGML ensures that system can represent the data in its own way.
- HTML used ASCII text, which allows the user to use any text editor.

Drawbacks

- GML and SGML were not suited for data interchange over the web.
- HTML instructions is used to display content rather than content they encompass.



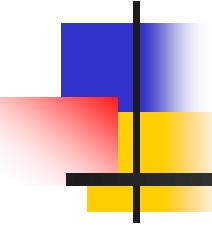
Evolution of XML 1-2

- XML is a W3C recommendation.
- It is a set of rules for defining semantic tags that break a document into parts.
- XML was developed over HTML.

Evolution of XML 2-2

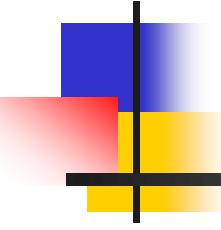
HTML	XML
HTML was designed to display data.	XML was designed to carry data.
HTML displays data and focuses on how data looks.	XML describes data and focuses on what data is.
HTML displays information.	XML describes information.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <Watch>
  <name>Titan</name>
  <price>$50</price>
  <description>A brown diamond studded watch</description>
</Watch>
```



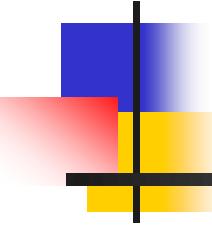
Features of XML

- XML stands for Extensible Markup Language
- XML is a markup language much like HTML
- XML was designed to describe data
- XML tags are not predefined
- XML uses a Document Type Definition (DTD) or an XML Schema to describe data



XML Markup 1-2

- XML markup defines the physical and logical layout of the document.
- XML markup divides a document into separate information containers called elements.
- A document consists of one outermost element, called `root` element that contains all the other elements, plus some optional administrative information at the top, known as XML declaration.



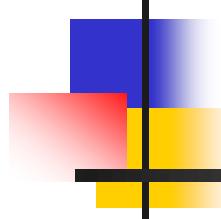
XML Markup 2-2

Code Snippet

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <FlowerPlanet>
  <Name>Rose</Name>
  <Price>$1</Price>
  <Description>Red in color</Description>
  <Number>700</Number>
</FlowerPlanet>
```

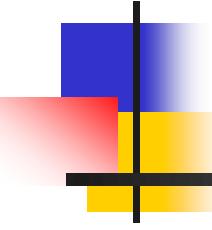
where,

<Name>, <Price>, <Description> and <Number> tags are elements
<FlowerPlanet> and </FlowerPlanet> are the root elements



Benefits of XML

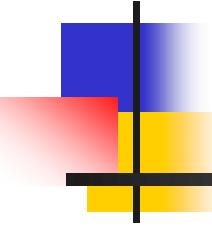
- Data independence - separates the content from its presentation
- Easier to parse - absence of formatting instructions makes it easy to parse
- Reducing Server Load - semantic and structural information enables it to be manipulated by any application, can now be performed by clients
- Easier to create – can easily create with the most primitive text processing tools
- Web Site Content – can be transformed into a number of other formats
- Remote Procedure Calls – protocol that allows objects on one computer to call objects on another computer
- E-Commerce - can be used as an exchange format



Lesson 2 – Exploring XML

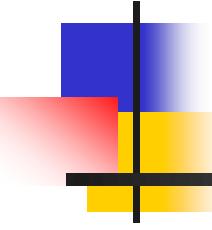
In this second lesson, **Exploring XML**, you will learn to:

- Describe the structure of an XML document.
- Explain the lifecycle of an XML document.
- State the functions of editors for XML and list the popularly used editors.
- State the functions of parsers for XML and list names of commonly used parsers.
- State the functions of browsers for XML and list the commonly used browsers.



XML Document Structure 1-4

- The two sections of an XML document are:
 - Document Prolog
 - Root Element
- An XML document consists of a set of unambiguously named "entities".
- Every document starts with a "root" or document entity. All other entities are optional.
- A single entity name can represent a large amount of text. The alias name is used each time some text is referenced and the processor expands the contents of the alias.



XML Document Structure 2-4

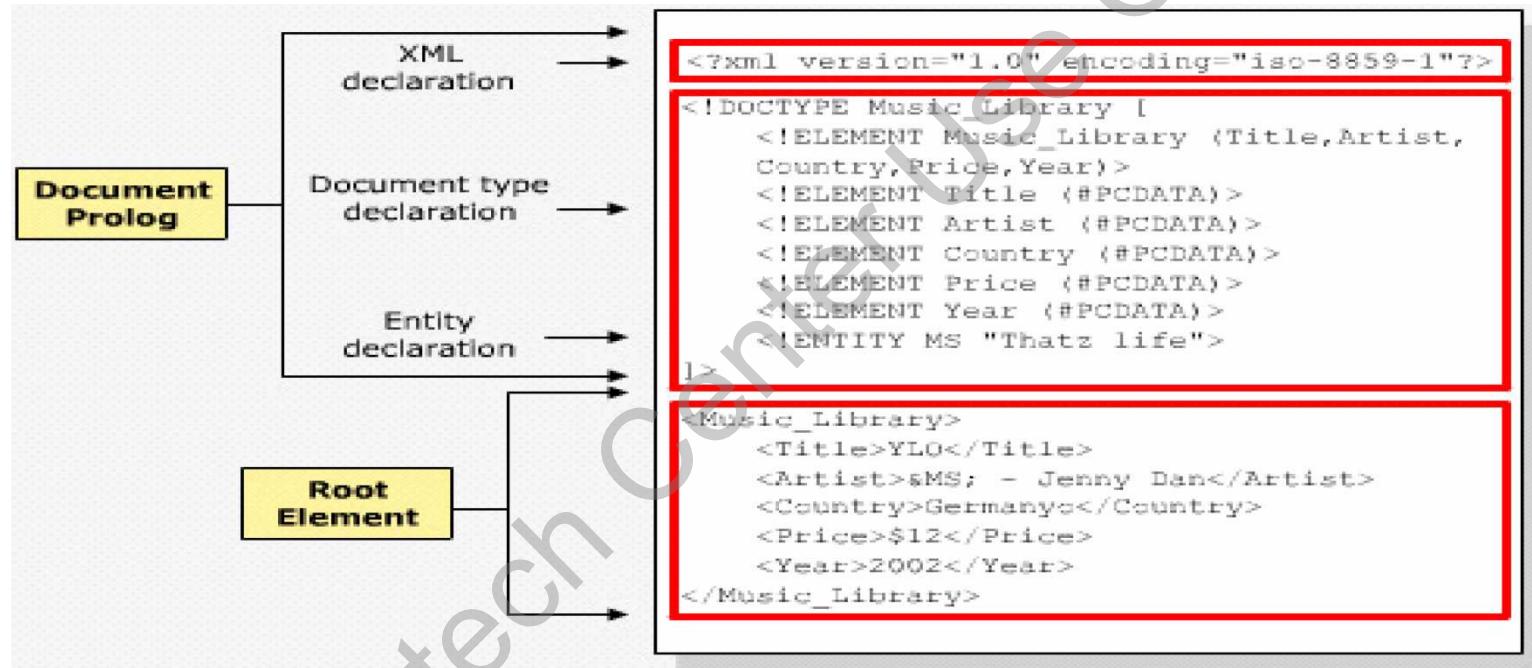
Document Prolog

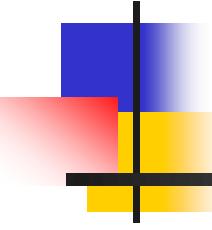
- Document prolog contains metadata and consists of two parts - XML Declaration and Document Type Declaration.
- XML Declaration specifies the version of XML being used.
- Document Type Declaration defines entities' or attributes' values and checks grammar and vocabulary of markup.

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

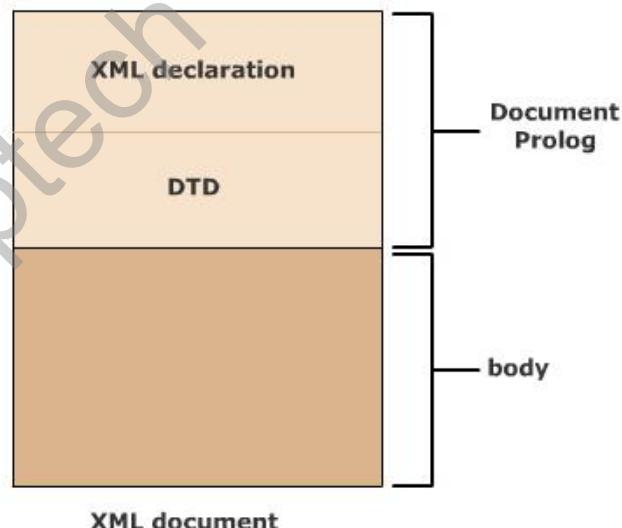
```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Music_Library [
    <!ELEMENT Music_Library (Title,Artist,Country,Price,Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Artist (#PCDATA)>
    <!ELEMENT Country (#PCDATA)>
    <!ELEMENT Price (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
    <!ENTITY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

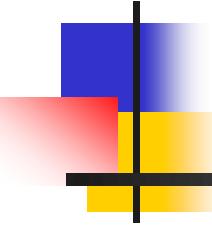
where,

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

Logical Structure 1-2

- The logical structure 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.
- XML Declaration and Document Type Definition are its components.





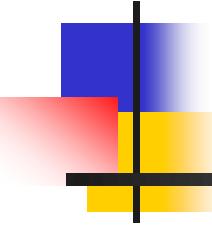
Logical Structure 2-2

Code Snippet

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

Code Snippet

```
<!DOCTYPE Music_Library SYSTEM  
"Mlibrary.dtd">
```



XML Document Life Cycle

1. XML document creation
2. Scanning
3. Parsing
4. Access
5. Conversion into Application program
6. Modification
7. Serialization



XML Author



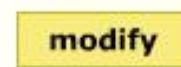
Editor



access



Document



modify



Parser



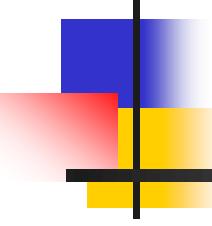
serialize



Browser



End-user



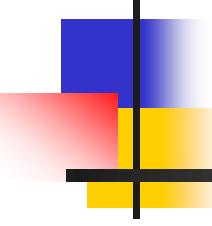
Editors

Main Functions:

- Add opening and closing tags to the code
- Check for validity of XML
- Verify XML against a DTD/Schema
- Perform series of transforms over a document
- Color the XML syntax
- Display the line numbers
- Present the content and hide the code
- Complete the word

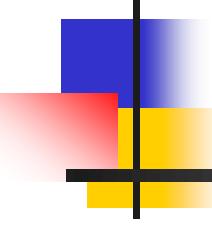
Some popularly used editors are:

- XMLwriter
- XML Spy
- XML Pro
- XMLmind
- XMetal



Parsers 1-2

- **An XML parser:**
 - Reads the document
 - Verifies it for its well-formedness
 - After verification, converts it into a tree of elements
- **Commonly used parsers:**
 - Crimson, Xerces, Oracle XML Parser, JAXP, MSXML
- **Type of parsers:**
 - Non Validating parser
 - Validating parser



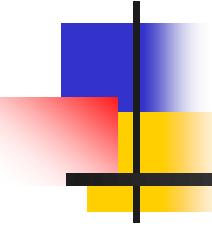
Parsers 2-2

Non Validating parser

- It checks the well-formedness of the document.
- Reads the document and checks for its conformity with XML standards.

Validating parser

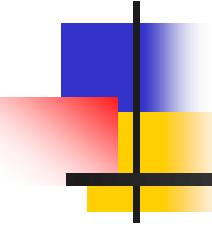
- It checks the validity of the document using DTD.



Browsers

- Web browsers can format XML data and display it to the user.
- Other programs like database, Musical Instrument Digital Interface (MIDI) program or a spreadsheet program may present XML data accordingly.

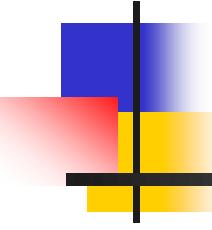
- Commonly used web browsers:
 - Netscape, Mozilla, Internet Explorer, Firefox, Opera



Lesson 3 – Working with XML

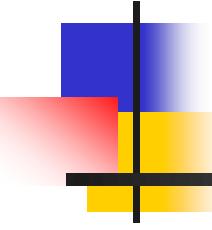
In this third lesson, **Working with XML**, you will learn to:

- Explain the steps towards building an XML document.
- Define what is meant by well-formed XML.



Creating an XML document

- An XML document has three main components:
 - Tags(markup) and text(content)
 - DTD or Schema
 - Formatting or display specifications
- The steps to build an XML document are as follows:
 1. Create an XML document in an editor
 2. Save the XML document
 3. Load the XML document in a browser



Exploring the XML document 1-3

- Various building blocks of an XML document:
 - XML Version Declaration
 - Document Type Definition
 - Document instance in which the content is defined by the mark up

Exploring the XML document 2-3

```
1  <?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
2
3  <!DOCTYPE student [
4      <!ELEMENT student (name,dob,bloodgroup,rollnumber)>
5      <!ELEMENT name (#PCDATA)>
6      <!ELEMENT dob (#PCDATA)>
7      <!ELEMENT bloodgroup (#PCDATA)>
8      <!ELEMENT rollnumber (#PCDATA)>
9  ]>
10
11 <student>
12     <name>James</name>
13     <dob>26th September, 1983</dob>
14     <bloodgroup>A positive</bloodgroup>
15     <rollnumber>17</rollnumber>
16 </student>
```

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

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

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

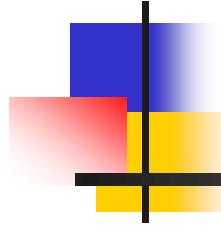
Exploring the XML document 3-3

```
1  <?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
2
3  <!DOCTYPE student [
4      <!ELEMENT student (name,dob,bloodgroup,rollnumber)>
5      <!ELEMENT name (#PCDATA)>
6      <!ELEMENT dob (#PCDATA)>
7      <!ELEMENT bloodgroup (#PCDATA)>
8      <!ELEMENT rollnumber (#PCDATA)>
9  ]>
10
11 <student>
12     <name>James</name>
13     <dob>26th September, 1983</dob>
14     <bloodgroup>A positive</bloodgroup>
15     <rollnumber>17</rollnumber>
16 </student>
```

Indicates the presence of external markup declarations. "Yes" indicates that there are no external markup declarations and "no" indicate that external markup declarations might exist.

Declares and defines the elements used in the document class.

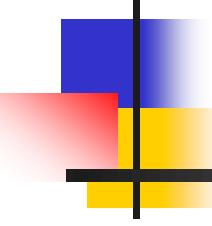
Defines the content of the XML document.



Meaning in Markup

- Structure
- Semantic
- Style

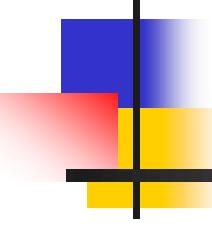
For Aptech Center Use Only



Well-formed XML document 1-3

- XML tags should be valid
- Length of the tags depend on the XML processors
- XML attributes should be valid
- XML documents should be verified

- Minimum of one element is required
- XML tags are case sensitive
- Every start tag should end with end tag
- XML tags should be nested properly



Well-formed XML document 2-3

Code Snippet

```
<Book>
<Name>Good XML</Name>
<Cost>$20</Cost>
</Book>
```

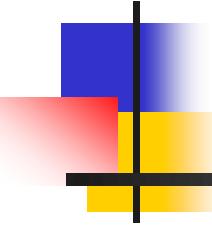
Well-formed XML document 3-3

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Library>
  <Name>
    4ULibrary
    #1, A Mansion, Izaho
    $10000
  </Name>
</Library>
```

Non Well- Formed Document

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Library>
  <Name>4ULibrary</Name>
  <Address> #1, A Mansion, Izaho </Address>
  <Auction>$10000</Auction>
</Library>
```

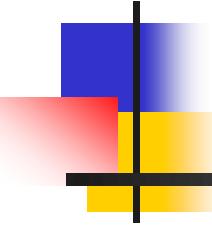
Well-Formed Document



Lesson 4 – XML Syntax

In this last lesson, **XML Syntax**, you will learn to:

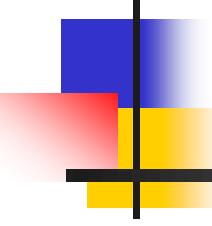
- State and describe the use of comments and processing instructions in XML.
- Classify character data that is written between tags.
- Describe entities, DOCTYPE declarations and attributes.



Comments 1-3

- Used for the people to give information about the code
- Usually made for the content

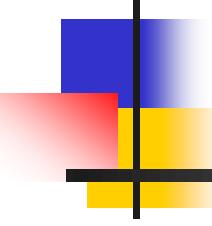
```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <Name NickName="John">
  <First>John</First>
  <!-- John is yet to pay the term fees -->
  <Last>Brown</Last>
  <Semester>Final</Semester>
</Name>
```



Comments 2-3

- Rules:

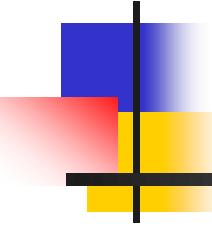
- Comments should not include “-” or “”
- Comments should not be placed within a tag or entity declaration
- Comments should not be placed before the XML declaration
- Comments can be used to comment the tag sets
- Comment should not be nested



Comments 3-3

Code Snippet

```
<Name NickName='John'>
    <First>John</First>
    <!--John is yet to pay the term fees-->
    <Last>Brown</Last> <Semester>Final</Semester>
</Name>
```



Processing Instructions 1-2

- The main objective is to present some special instructions to the application.
- All the processing instructions should begin with an identifier called target.

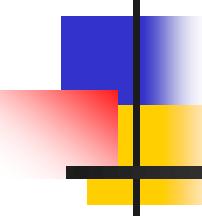
Syntax

```
<?PITarget <instruction>?>
```

where,

PITarget is the name of the application that should receive the processing instructions.

<instruction> is the instruction for the application



Processing Instructions 2-2

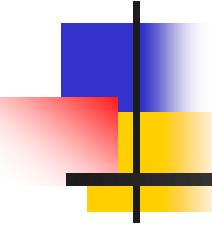
Code Snippet

```
<Name NickName='John'>
    <First>John</First>
    <!--John is yet to pay the term fees-->
    <Last>Brown</Last>
    <?feesprocessor SELECT fees FROM
STUDENTFEES?>
    <Semester>Final</Semester>
</Name>
```

where,

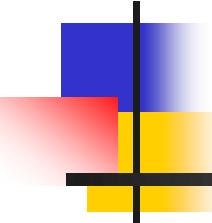
`feesprocessor` is the name of the application that receives the processing instruction

`SELECT fees FROM STUDENTFEES` is the instruction.



Classification of character data

- Character data describes the document's actual content with the white space.
- The character data can be classified into:
 - Character Data (CDATA)
 - Parsed Character Data (PCDATA)



PCDATA 1-2

- The data that is parsed by the parser is called as PCDATA.
- The main objective is to present some special instructions to the application.

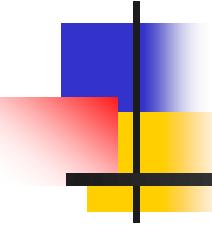
Output X

The XML page cannot be displayed

Cannot view XML input using XSL style sheet. Please correct the error and then click the [Refresh](#) button, or try again later.

Whitespace is not allowed at this location. Error processing resource 'file:///D:/XML By Example/Data.xml'. Line 5, Positi...

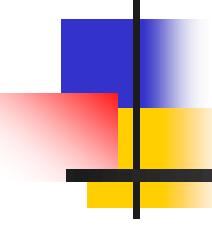
```
<Semester>Final> 10 & <20</Semester>
-----^
```



PCDATA 2-2

Code Snippet

```
<Name nickname='John'>
    <First>John</First>
    <!--John is yet to pay the term fees-->
    <Last>Brown</Last>
    <Semester>Final> 10 &lt;20</Semester>
</Name>
```

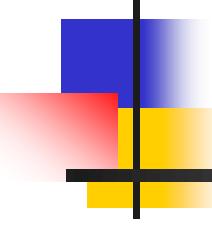


CDATA

- CDATA part begins with a “<! [CDATA [“ and ends with “]] >”.
- CDATA sections cannot be nested.

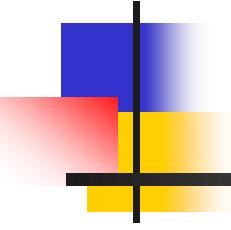
Code Snippet

```
<?xml version="1.0" standalone="yes"?>
<Svg>
    <Desc>Three shapes</Desc>
    <Para>But the formula was <! [CDATA[if (&x1 +
&x2) ]]> which
resulted in 7.</Para>
</Svg>>
```



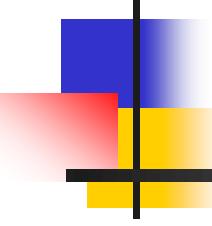
Entities 1-3

- XML document is made up of large amount of information called as entities.
- Every entity consists a name and a value.
- Entity reference consists of an ampersand (&), the entity name, and a semicolon (;).



Entities 2-3

Predefined Entity	Description	Output
<	produces the left angle bracket	<
>	produces the right angle bracket	>
&	produces the ampersand	&
'	produces a single quote character	'
"	produces a double quote character	"

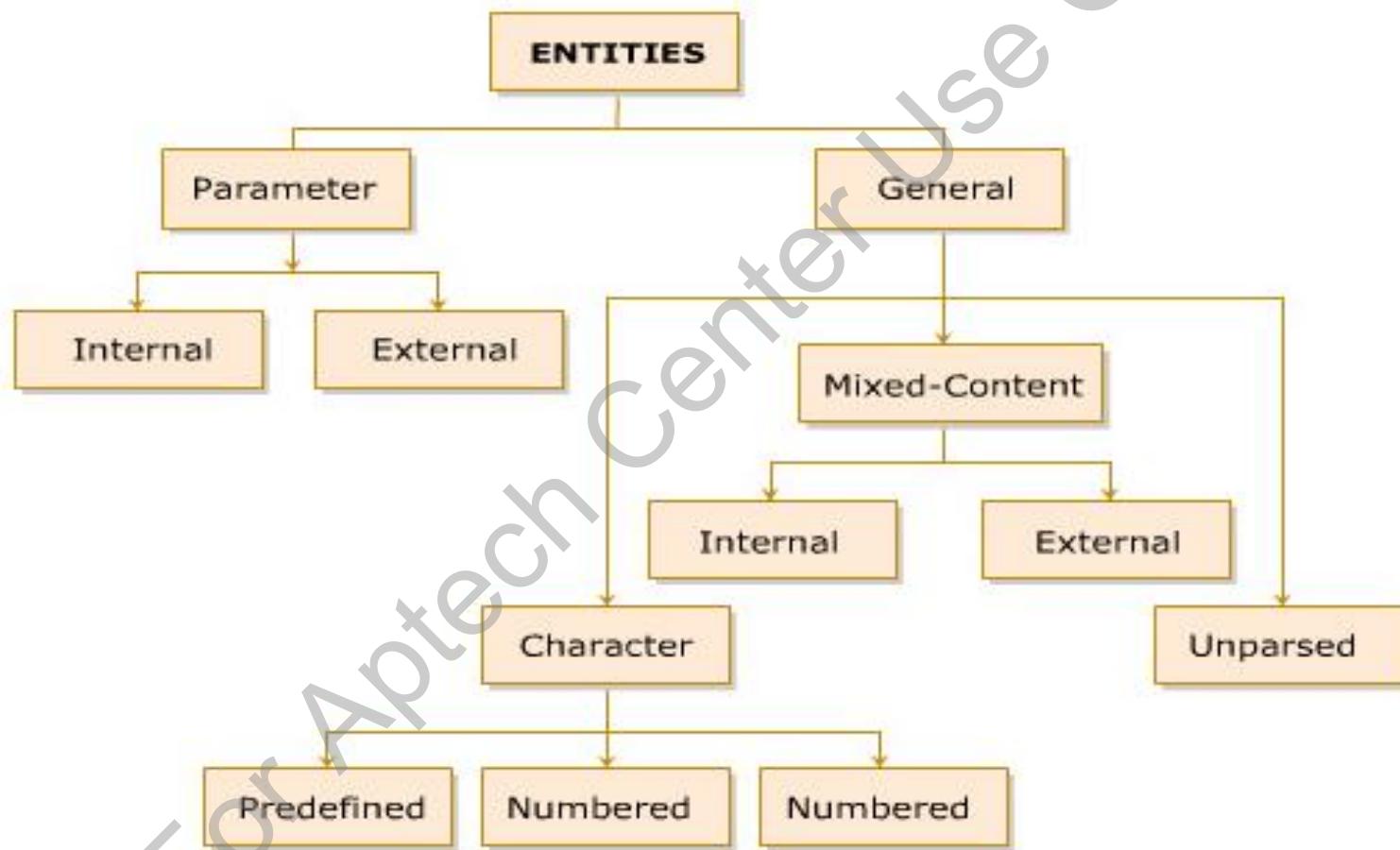


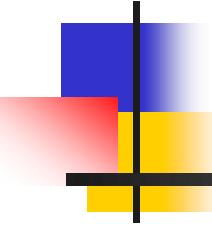
Entities 3-3

Code Snippet

```
<?xml version="1.0"?>
<!DOCTYPE Letter [
  <!ENTITY address "15 Downing St Floor 1">
  <!ENTITY city "New York">
]>
<Letter>
  <To>&quot;Tom Smith&quot;</To>
    <Address>&address;</Address>
    <City>&city;</City>
  <Body>
    Hi! How are you?
    The sum is &gt; $1000
  </Body>
  <From>ARNOLD</From>
</Letter>
```

Entity Categories 1-3





Entity Categories 2-3

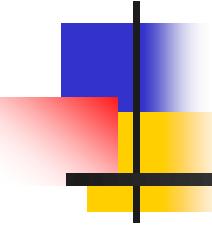
■ General Entity

- These entities are used within the document content.

Code Snippet

```
<<!ENTITY % ADDRESS "text that is to be  
represented by an entity">
```

A well-formed parameter entity will look like a general entity, except that it will include the "%" specifier.



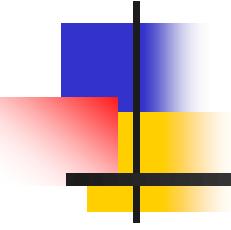
Entity Categories 3-3

■ Parameter Entity

- These entities are used only in the DTD.

Code Snippet

```
<!DOCTYPE MusicCollection [  
    <!ENTITY R "Rock">  
    <!ENTITY S "Soft">  
    <!ENTITY RA "Rap">  
    <!ENTITY HH "Hiphop">  
    <!ENTITY F "Folk">  
]>
```



DOCTYPE declarations 1-3

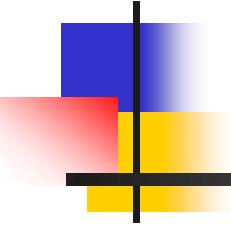
- **DTD File:**

Syntax

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

where,

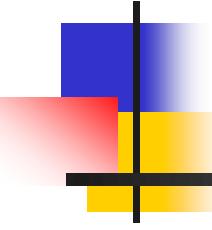
name_of_root_element is the name of the root element
SYSTEM is the url where the DTD is located
[Internal DTD subset] are declarations in the document



DOCTYPE declarations 2-3

Code Snippet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE program SYSTEM "HelloJimmy.dtd">
<Program>
<Comments>
This is a simple Java Program. It will display
the message "Hello Jimmy, How are you?" on
execution.
</Comments>
<Code> public static void main(String[] args) {
System.out.println("Hello Jimmy, How are you?");
// Display the string.
}
}
</Code>
</Program>
```



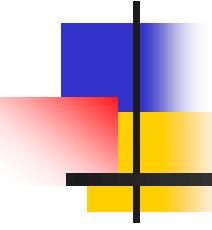
DOCTYPE declarations 3-3

- Document Type Definition defines the elements in the document

```
<!ELEMENT Program (comments, code)>
<!ELEMENT Comments (#PCDATA)>
<!ELEMENT Code (#PCDATA)>
```

- Output:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE program (View Source for full doctype...)>
- <Program>
  <Comments>This is a simple Java Program. It will display the message "Hello
  Jimmy,How are you?" on execution.</Comments>
  <Code>public static void main(String[] args) { System.out.println("Hello
  Jimmy,How are you?"); // Display the string. } </Code>
</Program>
```



Attributes

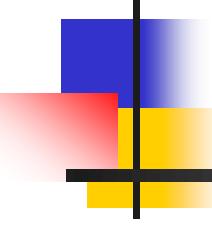
- Attributes are case sensitive and must start with a letter or underscore.

Syntax

```
<elementName attName1="attValue2"  
attName2="attValue2" ...>
```

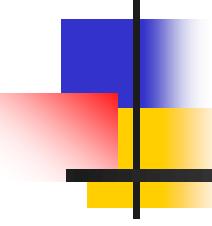
Code Snippet

```
<?xml version="1.0" ?>  
  <Player Sex="male">  
    <FirstName>Tom</FirstName>  
    <LastName>Federer</LastName>  
  </Player>
```



Summary 1-2

- **Introduction to XML**
 - XML was developed to overcome the drawbacks of earlier markup languages.
 - XML consists of set of rules that describe the content to be displayed in the document.
 - XML markup contains the content in the information containers called as elements.
- **Exploring XML**
 - XML is divided into two parts namely, document prolog and root element.
 - An XML editor creates the XML document and the parser validates the document.



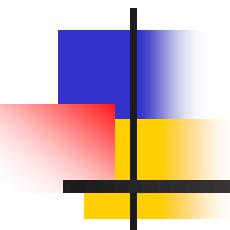
Summary 2-2

- **Working with XML**

- XML document is divided into XML Version Declaration, DTD and the document instance in which the markup defines the content.
- The XML markup is again categorized into structural, semantic and stylistic.
- The output of the XML document is displayed in the browser if it is well formed.

- **XML Syntax**

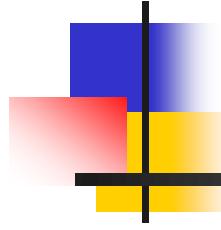
- Comments are used in the document to give information about the line or block of code.
- The content in XML document is divided into markup and character data.
- The entities in XML are divided into general entities and parameter entities.
- A DTD can be declared either internally or externally.



Module 2

Namespaces

For Aptech Center Use Only

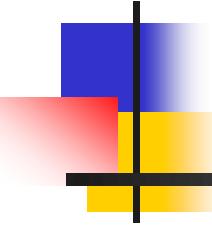


Module Overview

In this module, you will learn about:

- XML Namespaces
- Working with Namespaces Syntax

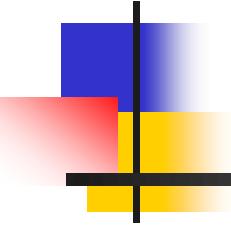
For Aptech Center Use Only



Lesson 1 – XML Namespaces

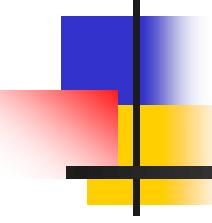
In this first lesson, **XML Namespaces**, you will learn to:

- Identify the need for a namespace.
- Define and describe namespaces in XML.



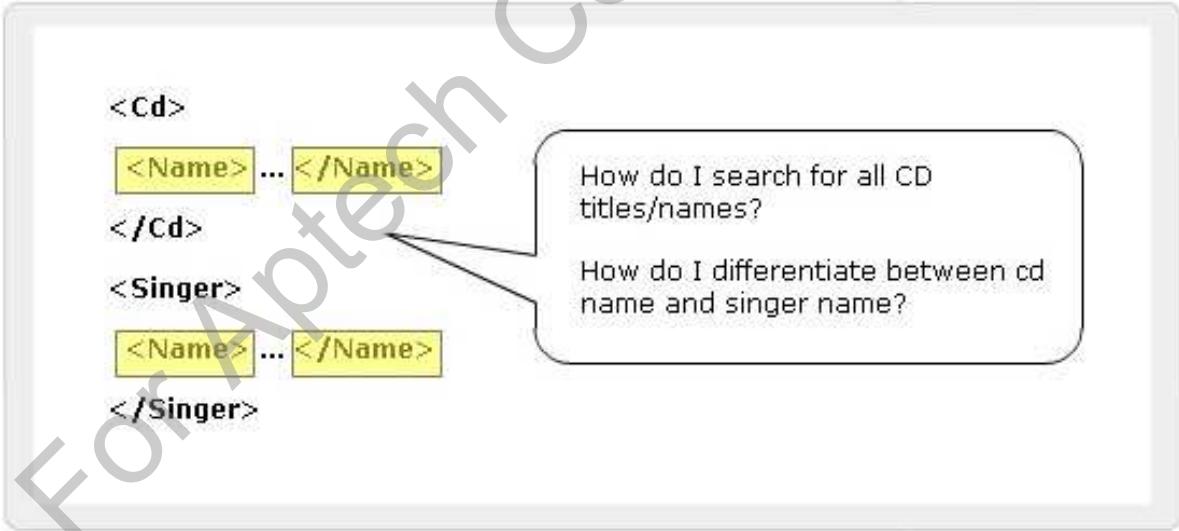
Duplicate Element Names

- It allows developers to create their own elements and attributes for their own projects.
- Developer has to ensure the uniqueness of the element names and attributes in a document.



Consequences of Duplicate Element Names

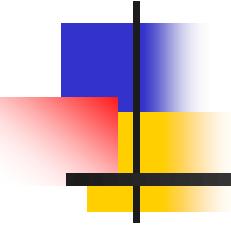
- Name conflicts are inevitable from different developers.
- It is difficult for the browser to distinguish a conflicting element.



```
<Cd>
  <Name> ... </Name>
</Cd>
<Singer>
  <Name> ... </Name>
</Singer>
```

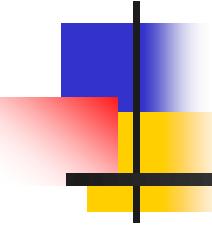
How do I search for all CD titles/names?

How do I differentiate between cd name and singer name?



Namespaces

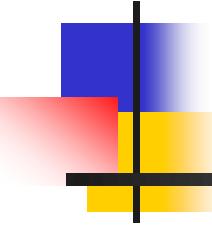
- Elements are distinguished by using namespaces.
- A namespace is a collection of names.
- Namespaces allow the browser to:
 - Combine documents from different sources
 - Identify the source of elements or attributes



Lesson 2 – Working with Namespaces Syntax

In this last lesson, **Working with Namespaces syntax**, you will learn to:

- Explain the syntax for XML namespaces.
- Discuss attributes and namespaces.
- Discuss how to use default namespaces.



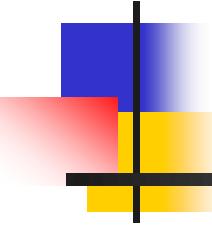
Prefixing element names

- Prefixes in element names provide a means to prevent name collisions.

Code Snippet

```
<CD:Title> Feel </CD:Title>  
and  
<Book:Title> Returning to Earth </Book:Title>.
```

In the above example, both CD and Book are namespace prefixes.

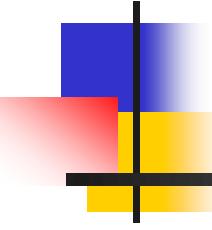


Problems Posed by Prefixes

- Duplication would still exist if prefixes are not unique
- To solve this problem, each namespace prefix is added to a Uniform Resource Identifier (URI)

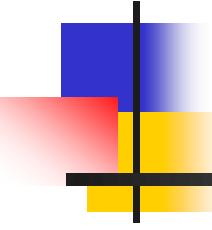
Code Snippet

```
<S:Student xmlns:S=" http://www.spectrafocus.com  
/student/">  
  <S:First>John</S:First>  
  <S:Last>Dewey</S:Last>  
  <S:Title>Student</S:Title>  
</S:Student>
```



Namespace Syntax 1-3

```
<namespacePrefix:  
elementName xmlns:  
namespacePrefix = 'URI'>
```



Namespace Syntax 2-3

NamespacePrefix

- Used as a reference to the namespace
- Prefixes must not begin with `xmlns` or `xml`

ElementName

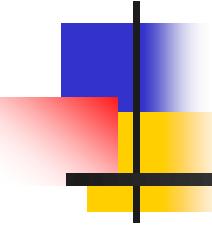
- Specifies the name of the element

xmlns

- `xmlns` stands for XML namespace

URI

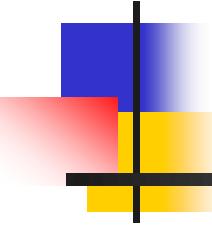
- URI is a string of characters which identifies an Internet Resource



Namespace Syntax 3-3

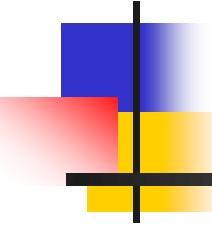
Code Snippet

```
<Auc:Books  
  xmlns:Auc="http://www.auction.com/books"  
  xmlns:B="http://www.books.com/HTML/1998/xml1">  
  ...  
  <Auc:BookReview>  
    <B:Table>  
      ...
```



Placing attributes in a Namespace 1-3

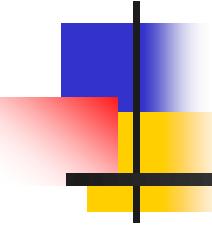
- Attributes belong to particular elements
- They are not a part of namespace, even if the element is within some namespace
- If an attribute has no prefix, it has no namespace
- An attribute without a prefix is in default namespace
- If an attribute name has a prefix, its name is in the namespace indicated by the prefix



Placing attributes in a Namespace 2-3

Syntax

```
<Catalog xmlns:Book =
"http://www.aptechworldwide.com">
<Book:Booklist>
<Book:Title Book:Type = "Fiction">Evening in
Paris</Book:Title>
<Book:Price>$123</Book:Price>
</Book:Booklist>
</Catalog>
```



Placing attributes in a Namespace 3-3

Code Snippet

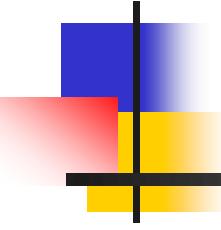
```
prefix:localname='value'  
or  
prefix:localname="value"
```

where,

`prefix` is used as a reference to the namespace. Prefixes must not begin with `xmlns` or `xml`

`localname` is the name of an attribute

`value` mentions a user defined value for an attribute



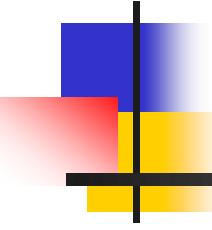
Default Namespaces 1-2

MathML Document

- XML-based markup language to represent complex mathematical expressions
- Comes in two types:
 - As a markup language for presenting the layout of mathematical expressions
 - As a markup language for presenting the mathematical content of the formula

Code Snippet

```
<MRow>
  <Mi>x</Mi>
  <Mo>+</Mo>
  <Mn>1</Mn>
</MRow>
```



Default Namespaces 2-2

Syntax

```
<elementName xmlns='URL'>
```

where,

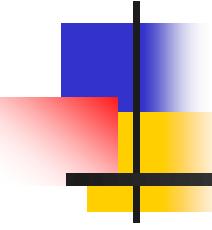
elementName specifies the name of the element belonging to the same namespace

URL specifies the namespace which is reference for a document or an HTML page on the Web

Code Snippet

```
<Catalog xmlns:Book = "http://www.aptechworldwide.com">
<Book:Booklist>
  <Book:Title Book:Type = "Fiction">Evening in
  Paris</Book:Title>
  <Book:Price>$123</Book:Price>
</Book:Booklist>
</Catalog>
```

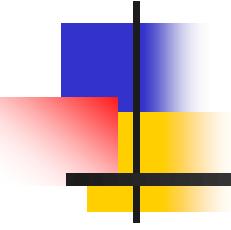
A default namespace using the `xmlns` attribute with a URI as its value



Override Default Namespaces 1-2

- Default namespace applies to the element on which it was defined and all descendants of that element
- New namespace definition overrides the previous one and becomes the default for that element

For Aptech Center USE ONLY

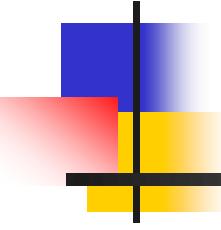


Override Default Namespaces 2-2

Code Snippet

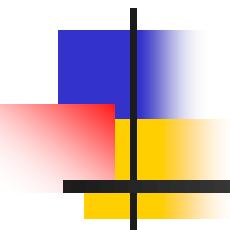
```
<Catalog xmlns = "http://www.aptechworldwide.com">
  <Book>
    <Title type = "Fiction">Evening in Paris</Title>
    <Price>$123</Price>
  </Book>
  <Book>
    <Title type = "Non-Fiction">Return to Earth</Title>
    <Price xmlns = "http://www.aptech.ac.in">$23</Price>
    <Title type = "Non-Fiction">Journey to the center of
the Moon</Title>
    <Price>$123</Price>
  </Book>
</Catalog>
```

This namespace of `price` element applies only to it and overrides the namespace in the `catalog` element



Summary

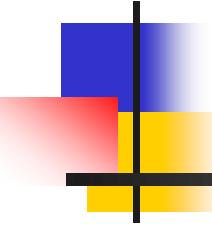
- **XML Namespaces**
 - Namespaces distinguish between elements and attributes with the same name from different XML applications.
 - It is a collection of names that can be used as element names or attribute names in XML document.
 - XML namespaces provide a globally unique name for a element or attribute to avoid name collisions.
- **Working with Namespaces syntax**
 - Namespaces are declared by an `xmlns` attribute whose value is the URI of the namespace.
 - If an attribute name has no prefix, it has no namespace.
 - A default namespace is used by an element and its child elements if the element does not have a namespace prefix.



Module 3

DTDs

For Aptech Center Use Only

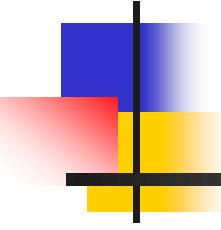


Module Overview

In this module, you will learn about:

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

For Aptech Center USE Only

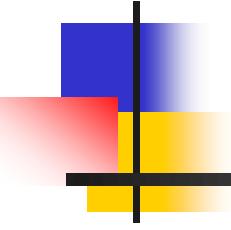


Lesson 1 – Document Type Definition

In this first lesson, **Document Type Definition**, you will learn to:

- Define what is meant by a DTD.
- Identify the need for a DTD.

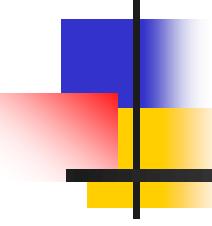
For Aptech Center USE ONLY



Definition of a DTD

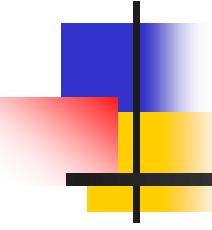
- It is a non XML document made up of element, attribute and entity declarations.
- It helps XML parsers to validate the XML document.

For Aptech Center Use Only



Need for a DTD

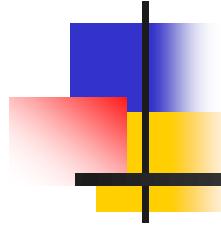
- XML allows a user to define his/her own tag.
- Standardization of elements and attributes was needed.
- A DTD can define all the possible combinations and sequences for elements.



Lesson 2 – Working with DTDs

In this second lesson, **Working with DTDs**, you will learn to:

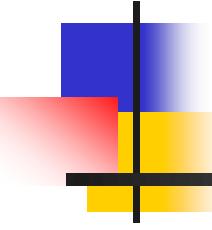
- Describe the structure of a DTD.
- Explain how to create a simple DTD.
- Describe what is meant by document type declarations.



Structure of DTD

- Element Declarations
- Attribute Declarations
- Entity Declarations

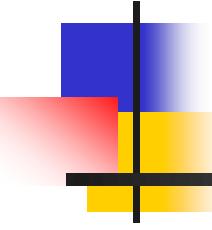
For Aptech Center Use Only



Creating Internal DTDs

1-3

- Declare all the possible elements
- Specify the permissible element children, if any
- Set the order in which elements must appear
- Declare all the possible element attributes
- Set the attribute data types and values
- Declare all the possible entities

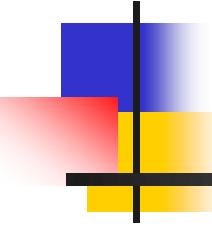


Creating Internal DTDs

2-3

Syntax

```
<!ELEMENT element-name (element-content)>
...
<!ATTLIST element-name attribute-name attribute-
type default-value>
...
<!ENTITY entity-name "entity-value">
...
```

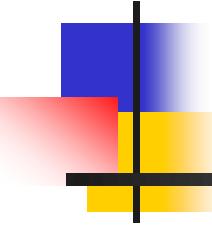


Creating Internal DTDs

3-3

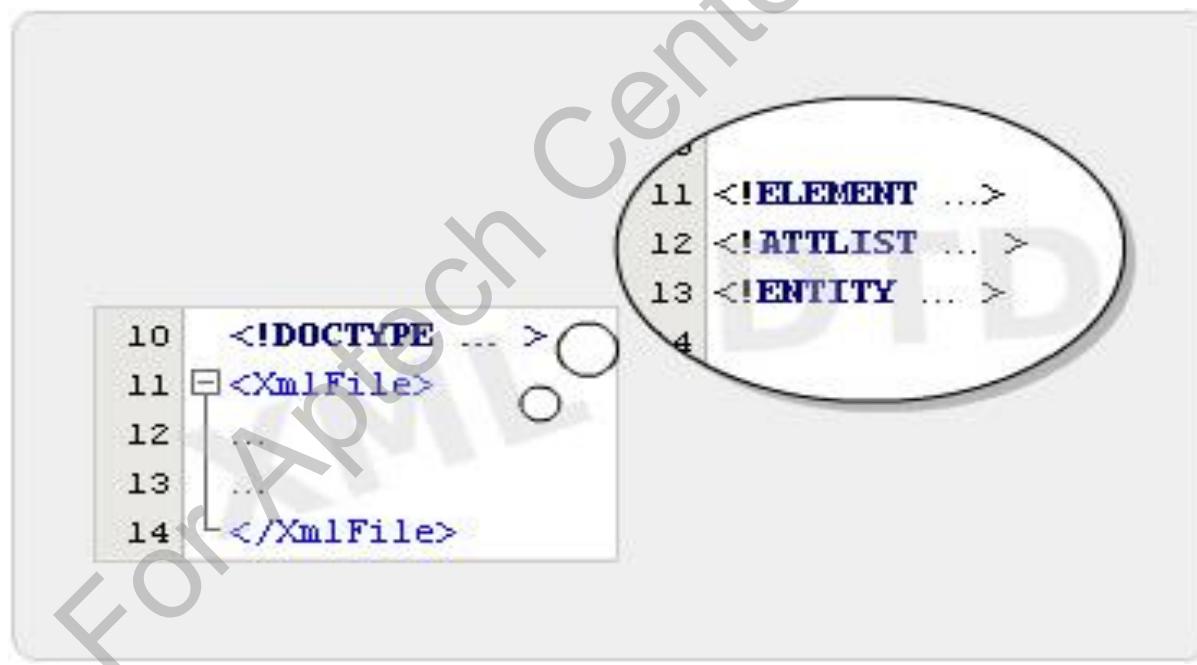
Code Snippet

```
<!ELEMENT Mobile (Company, Model, Price,  
Accessories)>  
<!ELEMENT Company (#PCDATA)>  
<!ELEMENT Model (#PCDATA)>  
<!ELEMENT Price (#PCDATA)>  
<!ELEMENT Accessories (#PCDATA)>  
<!ATTLIST Model Type CDATA "Camera">  
<!ENTITY HP "Head Phones">  
<!ENTITY CH "Charger">  
<!ENTITY SK "Starters Kit">
```



DOCTYPE Declarations 1-2

- It specifies the name of the DTD and either its content or location.
- It begins with `<!DOCTYPE` and ends with a `>`.



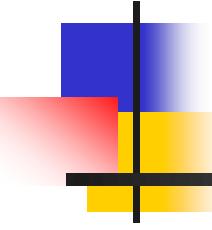
```
10  <!DOCTYPE ... >
11  <XmlFile>
12
13 ...
14 </XmlFile>
```

The code editor displays the following XML structure:

- Line 10: `<!DOCTYPE ... >`
- Line 11: A node labeled `<XmlFile>` is expanded, showing its children.
- Line 12: An ellipsis (...).
- Line 13: An ellipsis (...).
- Line 14: `</XmlFile>`

A callout bubble highlights the first three lines (the DOCTYPE declaration and the opening XML file tag) with the number 4. Another callout bubble highlights the three declarations (ELEMENT, ATTLIST, and ENTITY) with the number 11.

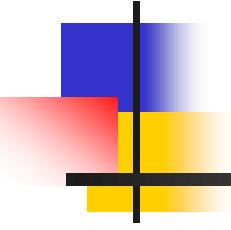
```
11 <!ELEMENT ... >
12 <!ATTLIST ... >
13 <!ENTITY ... >
```



DOCTYPE Declarations 2-2

Syntax

```
<!DOCTYPE name_of_root_element [ internal DTD  
subset ]>  
or  
<!DOCTYPE name_of_root_element SYSTEM "URL of the  
external DTD subset" >
```



Types of DTDs 1-2

- DTDs can be classified as Internal or External.
- **Internal DTDs**
 - It consists of the DTD name followed by the DTD enclosed in square brackets.

Internal DTD

14

15

16

```
<!DOCTYPE Mobile SYSTEM "mobile.dtd">
```

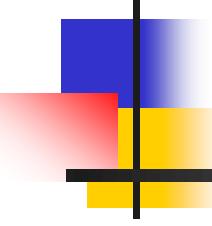
Types of DTDs 2-2

■ External DTDs

- It consists of the DTDs name followed by the SYSTEM keyword followed by the address

```
External DTD Reference

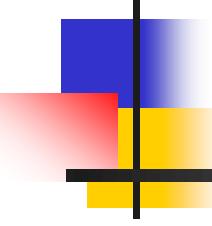
1<!DOCTYPE Mobile [
2    <!ELEMENT Mobile (Company, Model, Price, Accessories)>
3    <!ELEMENT Company (#PCDATA)>
4    <!ELEMENT Model (#PCDATA)>
5    <!ELEMENT Price (#PCDATA)>
6    <!ELEMENT Accessories (#PCDATA)>
7    <!ATTLIST Model Type #CDATA "Camera">
8    <!ENTITY HP "Head Phones">
9    <!ENTITY CH "Charger">
10   <!ENTITY SK "Starters Kit">
11  ]>
12 ...
13
14
15
16
17
18
19
```



Lesson 3 – Valid XML Documents

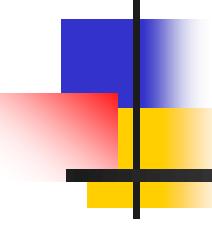
In this third lesson, **Valid XML Documents**, you will learn to:

- Define document validity.
- Describe in brief how to test for document validity.



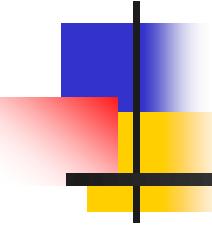
Well-Formed XML documents 1-2

- All elements must be enclosed by the root element
- All elements must have closing tags
- All tags should be case sensitive
- All elements must be properly nested
- All attribute values must always be quoted



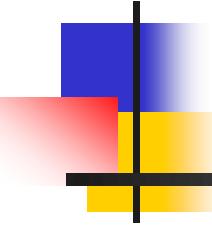
Well-Formed XML documents 2-2

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Mail>
  <To>anne@xyz.com</To>
  <From>bob@xyz.com</From>
  <Date>27th February 2007</Date>
  <Time>11:30 am</Time>
  <Cc />
  <Bcc />
  <Subject>Meeting at Main Conference Room at 4:30pm</Subject>
  <Message>Hi, Kindly request you to attend the cultural body
    general meeting in the main conference room at 4:30 pm.
    Please be present to learn about the new activities being
    planned for the employees for this year. Yours sincerely,
    Bob</Message>
  <Signature />
</Mail>
```



Valid XML documents 1-2

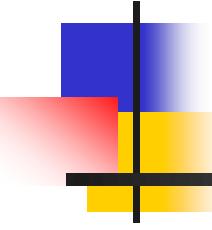
- A valid XML document is a well-formed XML document that adheres to its DTD.
- The validity of an XML document is determined by checking it against its DTD.



Valid XML documents 2-2

Code Snippet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Mail [
<!ELEMENT Mail (To, From, Date, Time, Cc, Bcc, Subject, Message, Signature)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT Cc (#PCDATA)>
<!ELEMENT Bcc (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT Signature (#PCDATA)>
]>
<Mail>
<To> anne@xyz.com </To>
<From> bob@xyz.com </From>
<Date> 27th February 2007 </Date>
<Time> 11:30 am </Time>
<Cc> </Cc>
<Bcc> </Bcc>
<Subject> Meeting at Main Conference Room at 4:30pm </Subject>
<Message> Hi, Kindly request you to attend the cultural body general meeting in the main conference room at 4:30 pm. Please be present to learn about the new activities being planned for the employees for this year. Yours sincerely, Bob
</Message>
<Signature> </Signature>
</Mail>
```



Testing XML for Validity

1-2

- It can be determined by using a validating parser such as Microsoft XML Core Services (MSXML) 6.0 Parser.
- MSXML enables the Internet Explorer (IE) browser to validate the code.
- The first image displays the validation result of a valid mobile.xml file.
- The second image displays the validation result after the removal of the signature element.

Testing XML for Validity

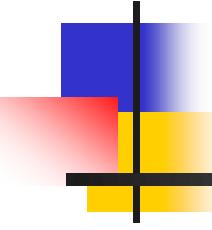
```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Mail (View Source for full doctype...)>
- <Mail>
  <To>anne@xyz.com</To>
  <From>bob@xyz.com</From>
  <Date>27th February 2007</Date>
  <Time>11:30 am</Time>
  <Cc />
  <Bcc />
  <Subject>Meeting at 4:30pm</Subject>
  <Message>Hi, Kindly request you to attend the cultural body general meeting in the main conference room at 4:30 pm. Please be present to learn about the new activities being planned for the employees for this year. Yours sincerely, Bob</Message>
  <Signature />
</Mail>
```



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE Mail (View Source for full doctype...)>
- <Mail>
```



```
<Message>Hi, Kindly request you to attend the cultural body general meeting in the main conference room at 4:30 pm. Please be present to learn about the new activities being planned for the employees for this year. Yours sincerely, Bob</Message>
<Signature />
</Mail>
```

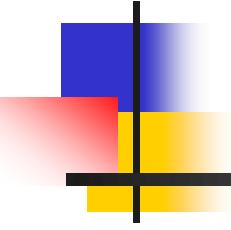


Lesson 4 - Declarations

In this last lesson, **Declarations**, you will learn to:

- Explain how to declare elements.
- Explain how to declare attributes.
- Describe entity declaration in a DTD.

For Aptech Center Use Only



Declaring Elements

- XML elements are declared with an element declaration.

Syntax

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

where,

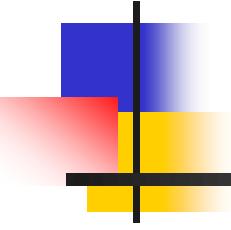
ELEMENT is the keyword,

element-name is the name of the element,

element-rule can be one of the following: No Content, Only Parsed Character Data, Any Contents, Children, Only One Occurrence, Minimum One Occurrence, Zero or More Occurrences, Zero or One Occurrence, Either/Or Content or Mixed Content

Testing XML for Validity

```
15 <!ELEMENT Mobile ( Company, Model, Price, Accessories)>
16 <!ELEMENT Company (#PCDATA)>
17 <!ELEMENT Model (#PCDATA)>
18 <!ELEMENT Price (#PCDATA)>
19 <!ELEMENT Accessories (#PCDATA)>
20 <!ATTLIST Model Type CDATA "Camera">
21 <!ENTITY HP "Head Phones">
22 <!ENTITY CH "Charger">
23 <!ENTITY SK "Starters Kit">
```



Declaring Attributes 1-2

Syntax

```
<!ATTLIST element-name attribute-name attribute-
type default-value>
```

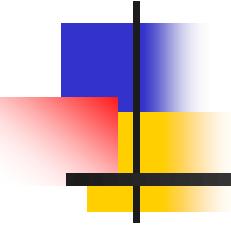
where,

`element-name` is the element the attribute belongs

`attribute-name` is the name of the attribute

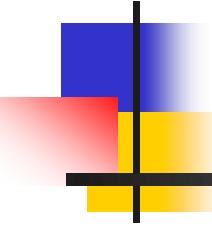
`attribute-type` is type of data the attribute can accept

`default-value` is the default value for the attribute



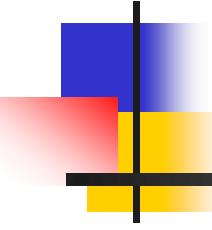
Declaring Attributes 2-2

Value	Description
PCDATA	Parsed character data
CDATA	Character data
(en1 en2 ...)	Enumerated list
ID	A unique id
IDREF	Id of another element
IDREFS	List of other ids
NMTOKEN	Valid XML name
NMTOKENS	List of valid XML names
ENTITY	An entity
ENTITIES	List of entities
NOTATION	Name of a notation
xml:	Predefined xml value



Specifying Attribute Values 1-3

Value	Description
value	Default value
#REQUIRED	Value must be included
#IMPLIED	Value does not have to be included
#FIXED	Value is fixed
en1 en2 ...	Listed enumerated values



Specifying Attribute Values 2-3

Syntax

➤ #IMPLIED

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

➤ #REQUIRED

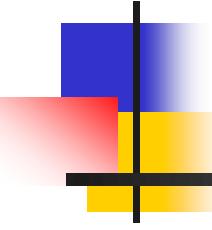
```
<!ATTLIST element-name attribute-name attribute-
type #REQUIRED>
```

➤ #FIXED

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

➤ Enumerated Attribute Values

```
<!ATTLIST element-name attribute-name
(en1|en2|..) default-value>
<!ATTLIST payment type (check|cash) "cash">
```



Specifying Attribute Values 3-3

Code Snippet

➤ **Default Value**

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

➤ **#IMPLIED**

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

➤ **#REQUIRED**

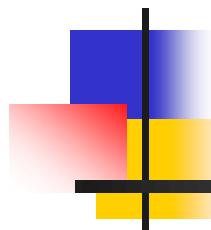
```
<!ATTLIST Model Type CDATA #REQUIRED>
```

➤ **#FIXED**

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

➤ **Enumerated Attribute Values**

```
<!ATTLIST Model Type (Camera|Bluetooth) "Camera">
```



Entities in DTD 1-2

- It is a placeholder that consists of a name and a value.
- It is declared once and then repeatedly used throughout the document.

Syntax

➤ Entity declaration:

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

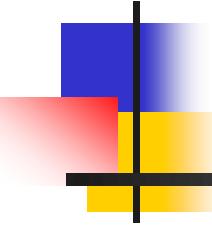
➤ Entity Reference:

```
&entity-name;
```

Entities in DTD 2-2

Code Snippet

```
□ <!DOCTYPE Mobile [  
    <!ELEMENT Mobile (Company, Model, Price, Accessories)>  
    <!ELEMENT Company (#PCDATA)>  
    <!ELEMENT Model (#PCDATA)>  
    <!ELEMENT Price (#PCDATA)>  
    <!ELEMENT Accessories (#PCDATA)>  
    <!ATTLIST Model Type CDATA "Camera">  
    <!ENTITY HP "Head Phones">  
    <!ENTITY CH "Charger">  
    <!ENTITY SK "Starters Kit">  
>]  
□ <Mobile>  
    <Company> Nokia </Company>  
    <Model Type="Camera"> 6600 </Model>  
    <Price> 9999 </Price>  
    <Accessories> &HP;, &CH; and a &SK; </Accessories>  
</Mobile>
```



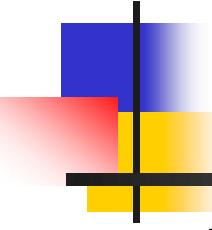
Kinds of Entity Declarations 1-4

Internal Entity Declaration

- The entity value is explicitly mentioned in the entity declaration.

Syntax

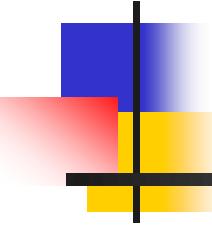
```
<!ENTITY entity-name "entity-value">
```



Kinds of Entity Declarations 2-4

Code Snippet

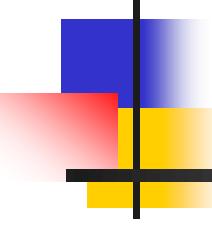
```
<!DOCTYPE Mobile [  
  <!ELEMENT Mobile (Company, Model, Price, Accessories)>  
  <!ELEMENT Company (#PCDATA)>  
  <!ELEMENT Model (#PCDATA)>  
  <!ELEMENT Price (#PCDATA)>  
  <!ELEMENT Accessories (#PCDATA)>  
  <!ATTLIST Model Type CDATA "Camera">  
  <!ENTITY HP "Head Phones">  
  <!ENTITY CH "Charger">  
  <!ENTITY SK "Starters Kit">  
>  
<Mobile>  
  <Company> Nokia </Company>  
  <Model Type="Camera"> 6600 </Model>  
  <Price> 9999 </Price>  
  <Accessories> &HP;, &CH; and a &SK; </Accessories>  
</Mobile>
```



Kinds of Entity Declarations 3-4

External Entity Declaration

- A link or path to the entity value is mentioned in place of the entity value

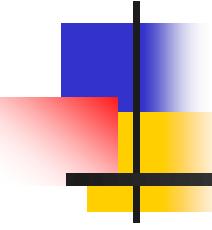


Kinds of Entity Declarations 4-4

Syntax

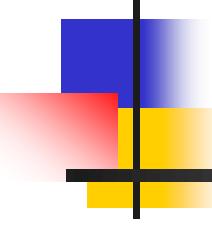
```
<!ENTITY entity-name SYSTEM "URI/URL">
```

```
<!DOCTYPE Mobile [  
    <!ELEMENT Mobile (Company, Model, Price, Accessories)>  
    <!ELEMENT Company (#PCDATA)>  
    <!ELEMENT Model (#PCDATA)>  
    <!ELEMENT Price (#PCDATA)>  
    <!ELEMENT Accessories (#PCDATA)>  
    <!ATTLIST Model Type CDATA "Camera">  
    <!ENTITY HP SYSTEM "hp.txt">  
    <!ENTITY CH SYSTEM "ch.txt">  
    <!ENTITY SK SYSTEM "sk.txt">  
>  
<Mobile>  
<Company> Nokia </Company>  
<Model Type="Camera"> 6600 </Model>  
<Price> 9999 </Price>  
<Accessories> &HP;,&CH; and a &SK; </Accessories>  
</Mobile>  
hp.txt  
Head Phones  
ch.txt  
Charger  
sk.txt  
Starters Kit
```



Summary 1-2

- **Document Type Definition**
 - A DTD is a non XML document made up of element, attribute and entity declarations.
- **Working with DTDs**
 - The DTD structure is composed of element declarations, attribute declarations, and entity declarations.
 - A document type declaration declares that the XML file in which it is present adheres to a certain DTD.



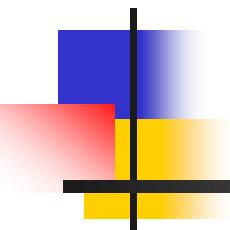
Summary 2-2

■ **Valid XML Documents**

- A well-formed XML document adheres to the basic XML syntax rules.
- A valid XML document is a well-formed XML document that adheres to its DTD.

■ **Declarations**

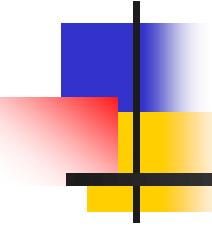
- XML elements are declared with an element declaration in the DTD.
- An entity is a placeholder that consists of a name and a value.



Module 4

XML Schema

For Aptech Center Use Only

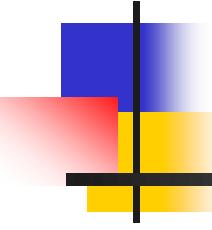


Module Overview

In this module, you will learn about:

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

For Aptech Center Use Only



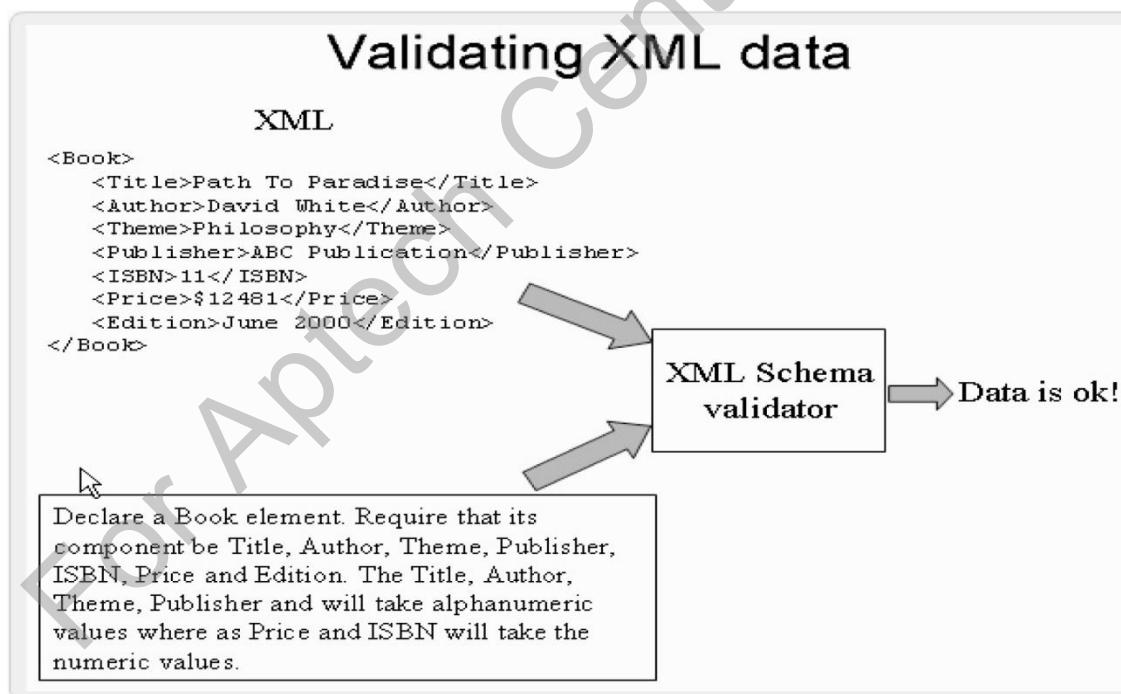
Lesson 1 – XML Schema

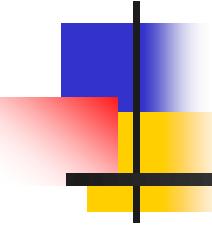
In this first lesson, **XML Schema**, you will learn to:

- Define and describe what is meant by schema.
- Identify the need for schema.
- Compare and differentiate the features of DTDs and schemas

XML Schema 1-2

- It defines the valid building blocks of an XML document.
- XML Schema language is referred as XML Schema Definition (XSD).
- It describes the data that is marked up, and also specifies the arrangement of tags and text.
- The dictionary defines a schema as “An outline or a model”.

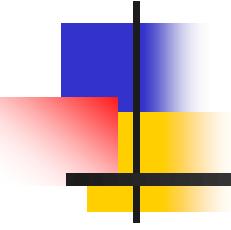




XML Schema 2-2

Code Snippet

```
<Book>
    <Title>Path To Paradise</Title>
    <Author>David White</Author>
    <Theme>Philosophy</Theme>
    <Publisher>ABC Publication</Publisher>
    <ISBN>11</ISBN>
    <Price>$12481</Price>
    <Edition>June 2000</Edition>
</Book>
```



XML Schema Objectives 1-2

An XML Schema defines:

- Elements and attributes that can appear in a document
- Which elements are child elements
- The order and number of child elements
- Whether an element is empty or can include text
- Data types for elements and attributes
- Default and fixed values for elements and attributes

XML Schema Objectives

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!--
4      Document    : books.xml
5      Author      : vincent
6      Description:
7          This document defines a schema for a library of books.
8  -->
9
10 <library>
11 <fiction>
12     <book>The Firm, John Grisham</book>
13     <book>Coma, Robin Cook</book>
14 </fiction>
15 <nonfiction>
16     <book>Freakonomics , Malcolm Gladwell</book>
17 </nonfiction>
18 <?latest editions only?>
19 </library>
```

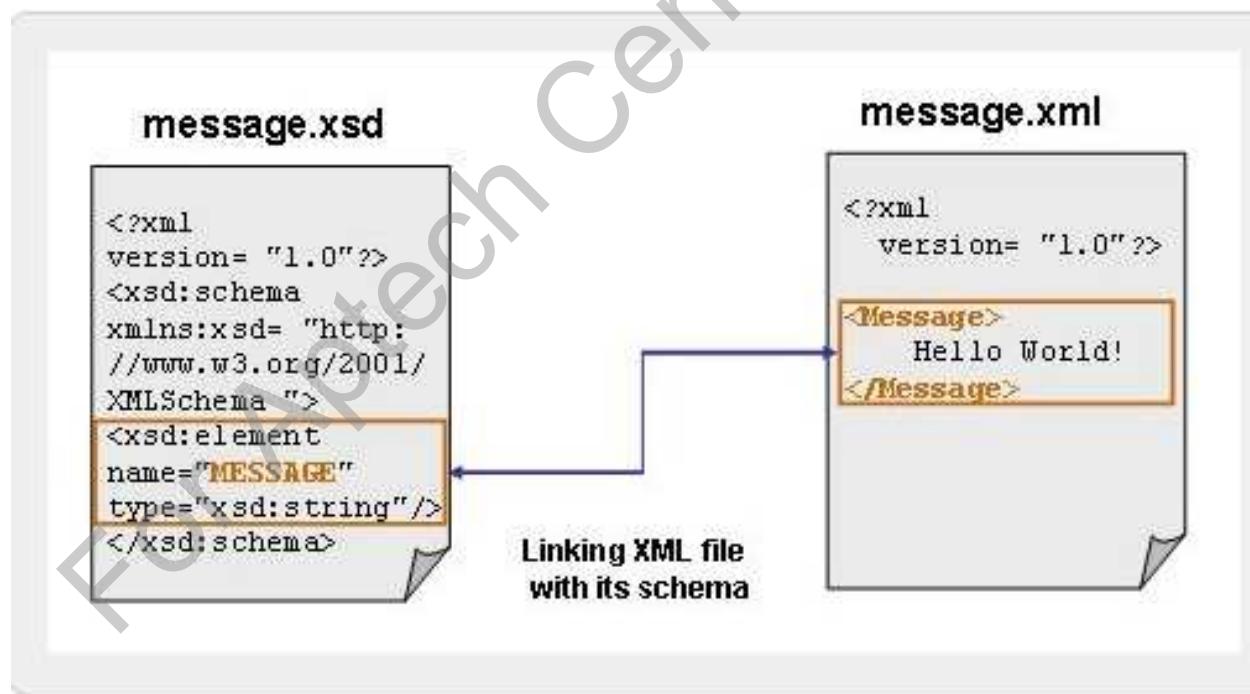
How to write an XML Schema? 1-2

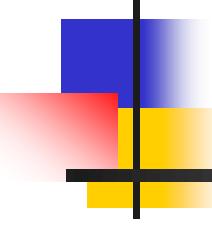
XML File

- The XML document contains a single element, <Message>.

XSD File

- The file is saved with ".xsd" as the extension for storing schema documents





How to write an XML Schema? 2-2

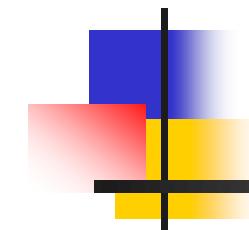
Code Snippet

XML File: message.xml

```
<?xml version= "1.0"?>
<Message>
    Hello World!
</Message>
```

XSD File: message.xsd

```
<?xml version= "1.0"?>
<xsd:schema xmlns:xsd= "http://www.w3.org/2001/XMLSchema
">
<xsd:element name= "MESSAGE " type= "xsd:string "/>
</xsd:schema>
```

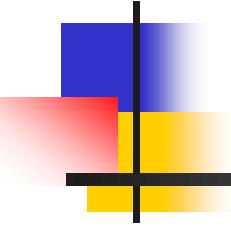


What is an XML Schema? 1-3

- XML schemas allow Web applications to exchange XML data more robustly using the following range of new features:
 - Schemas support data types
 - Schemas are portable and efficient
 - Schemas secure data communication
 - Schemas are extensible
 - Schemas catch higher-level mistakes
 - Schemas support Namespace

Schemas support data types

- Support and ability to create the required datatypes has overcome the drawbacks of DTDs
- Easy to define and validate valid document content and data formats
- Easy to implement the restrictions on data



What is an XML Schema? 2-3

Schemas are portable and efficient

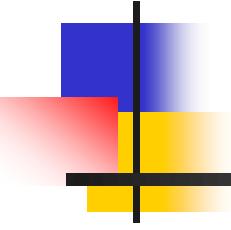
- Portable and efficient
- No need to learn any new language or any editor
- Similar XML parser can be used to parse the Schema files

Schemas secure data communication

- Sender can specify the data in a way that the receiver will understand

Schemas are extensible

- It is possible to reuse an existing schema to create another schema
- It is possible to create own data types derived from the standard types
- Support reference of multiple schemas in the same document



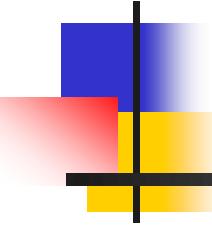
What is an XML Schema? 3-3

Schemas catch higher-level mistakes

- Catch higher-level mistakes that arise, such as a required field of information is missing or in a wrong format, or an element name is mis-spelled

Schemas support Namespace

- Support for XML Namespaces allows the programmer to validate documents that use markup from multiple namespaces
- Constructs can be re-used from schemas already defined in a different namespace

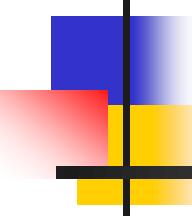


Comparing DTDs with Schemas 1-2

- Drawbacks of using DTDs are:
 - DTDs are written in a non-XML syntax
 - DTDs are not extensible
 - DTDs do not support namespaces
 - DTDs offer limited data typing

Sample External DTD File: program.dtd

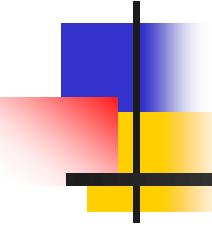
```
<!ELEMENT program (comments, code)>
<!ELEMENT comments (#PCDATA)>
<!ELEMENT code (#PCDATA)>
```



Comparing DTDs with Schemas 2-2

Sample XML File with a reference to dtd: program.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE program SYSTEM "program.dtd">
<program>
    <comments>
        This is a simple Java Program. It will display the
        message
        "Hello world!" on execution.
    </comments>
    <code>
        public static void main(String[] args)
            System.out.println("Hello World!"); // Display the
        string.
    </code>
</program>
```



Advantages of XML Schemas over DTD 1-3

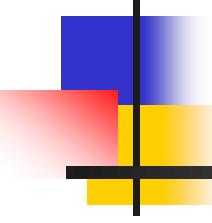
XML schema offers a range of new features:

- Richer data types
- Archetypes
- Attribute grouping
- Refinable archetypes

Advantages of XML Schemas over DTD 2-3

Sample schema File: mail.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.abc.com"
xmlns="http://www.abc.com"
elementFormDefault="qualified">
<xs:element name="mail">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="header" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

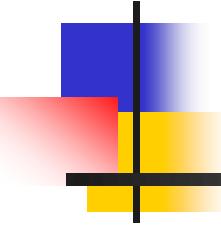


Advantages of XML Schemas over DTD 3-3

Sample XML File with a reference to schema: mail.xml

```
<?xml version="1.0"?>
<mail
  xmlns="http://www.abc.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.abc.com mail.xsd">

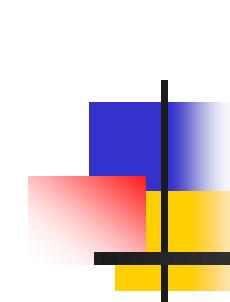
  <to>John</to>
  <from>Jordan</from>
  <header>Scheduler</header>
  <body>3rd March Monday, 7:30 PM: board meeting!</body>
</mail>
```



Lesson 2 – Exploring XML Schemas

In this second lesson, **Exploring XML Schemas**, you will learn to:

- List the data types supported by schemas.
- Explain the XML Schema vocabulary.



Data Types Supported by Schema 1-6

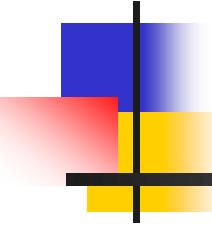
- XML Schema describes a number of built-in data types, which can be used to specify and validate the intended data type of the content.
- It also allows a user to create a user-defined data type by extending the built-in data types using facets.
- XML Schema recommendation defines two sorts of data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

Built-in data types

- Available to all XML Schema authors, and should be implemented by a conforming processor.

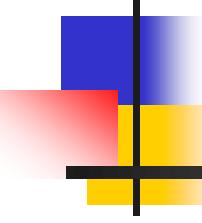
User-derived data types

- Defined in individual schema instances, and are particular to that schema.



Data Types Supported by Schema 2-6

string	<ul style="list-style-type: none">■ can contain characters, line feeds, carriage returns, and tab characters
boolean	<ul style="list-style-type: none">■ legal values for boolean data type are true and false■ true can be replaced by the numeric value 1 and false can be replaced by the value 0
numeric	<ul style="list-style-type: none">■ represents a numerical value■ includes numbers such as whole numbers, and real numbers
dateTime	<ul style="list-style-type: none">■ represents a particular time on a given date, written as a string
binary	<ul style="list-style-type: none">■ include graphic files, executable programs, or any other string of binary data
anyURI	<ul style="list-style-type: none">■ represents a file name or location of the file



Data Types Supported by Schema 3-6

Syntax: string

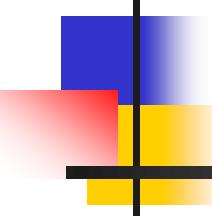
```
<xs:element name="element_name" type="xs:string"/>
```

Syntax: boolean

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

Syntax: numeric

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



Data Types Supported by Schema 4-6

Syntax: dateTime

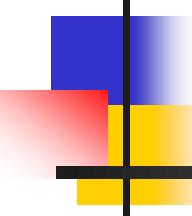
```
<xs:element name="element_name" type="xs:dateTime"/>
```

Syntax: binary

```
<xs:element name="image_name" type="xs:hexBinary"/>
```

Syntax: anyURI

```
<xs:attribute name="image_name" type="xs:anyURI"/>
```



Data Types Supported by Schema 5-6

Code Snippet: string

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

```
<Customer>John Smith</Customer>
```

Code Snippet: boolean

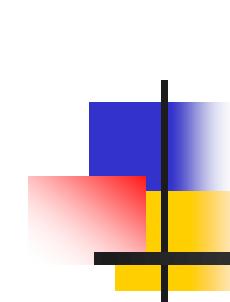
```
<xs:attribute name="Disabled" type="xs:boolean"/>
```

```
<Status Disabled="true">OFF</Status>
```

Code Snippet: numeric

```
<xs:element name="Price" type="xs:numeric"/>
```

```
<Price>500</Price>
```



Data Types Supported by Schema 6-6

Code Snippet: dateTime

```
<xs:element name="BeginAt" type="xs:dateTime"/>
```

```
<start>2001-05-10T12:35:40</start>
```

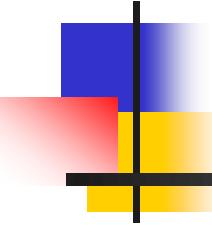
Code Snippet: binary

```
<xs:element name="Logo" type="xs:hexBinary"/>
```

Code Snippet: anyURI

```
<xs:attribute name="flower" type="xs:anyURI"/>
```

```
<image  
flower="http://www.creativepictures.com/gallery/flower.gif  
" />
```



Additional Data types 1-3

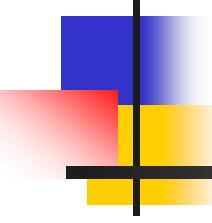
- Additional data types are derived from the basic built-in data types, which are called base type data types.
- The generated or derived data types, supported by the XML schema include:
 - integer
 - decimal
 - time

Integer

- Base type is numeric data type. Includes both positive and negative numbers. Used to specify a numeric value without a fractional component.

Decimal

- Represent exact fractional parts such as 3.26. Base type is numeric data type. Used to specify a numeric value.



Additional Data types 2-3

Time

- Base type is the dateTime data type. Default representation is 16:35:26. time data type is used to specify time. Specified as "hh:mm:ss".

Syntax: integer

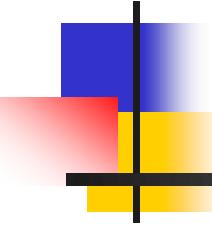
```
<xs:element name="element_name" type="xs:integer"/>
```

Syntax: decimal

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

Syntax: time

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



Additional Data types 3-3

Code Snippet: integer

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

```
<Age>999</Age>
```

Code Snippet: decimal

```
<xs:element name="Weight" type="xs:decimal"/>
```

```
<prize>+70.7860</prize>
```

Code Snippet: time

```
<xs:element name="BeginAt" type="xs:time"/>
```

```
<BeginAt>09:30:10.5</BeginAt>
```

Schema Vocabulary 1-3

- Creating a schema using XML schema vocabulary is like creating any other XML document using a specialized vocabulary.

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.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4    targetNamespace="http://www.abc.com"
5    xmlns="http://www.abc.com"
6    elementFormDefault="qualified">
7    <xs:element name="Mail">
8      <xs:complexType>
9        <xs:sequence>
10       <xs:element name="To" type="xs:string"/>
11       <xs:element name="From" type="xs:string"/>
12       <xs:element name="Header" type="xs:string"/>
13       <xs:element name="Body" type="xs:string"/>
14     </xs:sequence>
15   </xs:complexType>
16   </xs:element>
17 </xs:schema>
```

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

Schema Vocabulary 2-3

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <xsschema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4      targetNamespace="http://www.abc.com"
5      xmlns="http://www.abc.com"
6      elementFormDefault="qualified">
7      <xselement name="Mail">
8          <xsccomplexType>
9              <xssequence>
10             <xselement name="To" type="xs:string"/>
11             <xselement name="From" type="xs:string"/>
12             <xselement name="Header" type="xs:string"/>
13             <xselement name="Body" type="xs:string"/>
14         </xssequence>
15     </xsccomplexType>
16   </xselement>
17 </xsschema>
```

This line of code points to "http://www.abc.com" as the default namespace.

It indicates that elements used by the XML instance document which were declared in this schema needs to be qualified by the namespace.

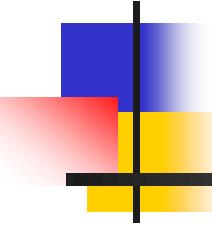
Schema Vocabulary 3-3

It indicates the default namespace declaration. This declaration informs the schema-validator that all the elements used in this XML document are declared in the default ("http://www.abc.com") namespace.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <Mail xmlns="http://www.abc.com"
4      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5      xsi:schemaLocation="http://www.abc.com/mail_schema.xsd">
6      <To>John</To>
7      <From>Jordan</From>
8      <Header>Scheduler</Header>
9      <Body>3rd March Monday, 7:30 PM: board meeting!</Body>
10     </Mail>
```

This is the instance namespace available for the XML schema.

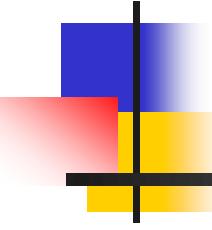
This schemaLocation attribute has two values. The first value identifies the namespace. The second value is the location of the XML schema where it is stored.



Lesson 3 – Working with Complex Types

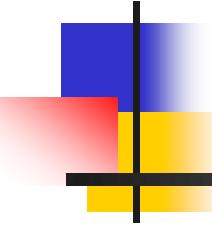
In this third lesson, **Working with Complex Types**, you will learn to:

- Describe complex type elements.
- Describe `minOccurs` and `maxOccurs`.
- Explain element content and mixed content.
- Describe how grouping can be done.



Complex Types 1-4

- A schema assigns a type to each element and attribute it declares.
- Elements with complex type may contain nested elements and have attributes.
- Only elements can contain complex types. Complex type elements have four variations. They are:
 - Empty Elements
 - Only Elements
 - Only Text
 - Mixed



Complex Types 2-4

Empty elements

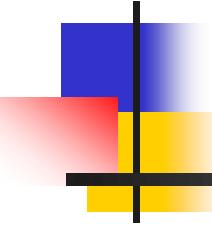
- Empty elements optionally specify attributes types, but do not permit content.

```
<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.

```
<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>
```

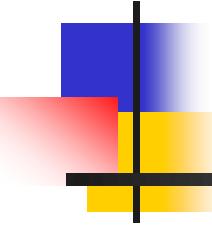


Complex Types 3-4

Only Text

- These elements can only contain text and optionally may or may not have attributes.

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



Complex Types 4-4

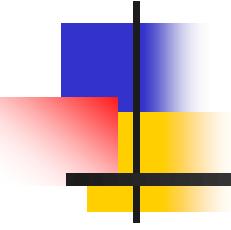
Mixed

- These are elements that can contain text content as well as sub-elements within the element. They may or may not have attributes.

```
<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>
```

A complex element can be defined in two different ways:

- By directly naming the element
- By using the name and type attribute of the complex type

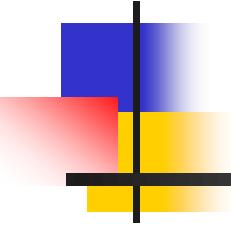


Defining Complex Types 1-2

- **By directly naming the element**

Code Snippet

```
<xs:element name="Student">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="MiddleName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

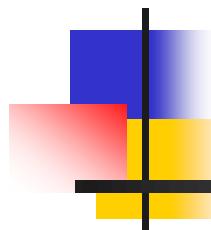


Defining Complex Types 2-2

- **By using the name and type attribute of the complex type**

Code Snippet

```
<xs:element name="Student" type="PersonInfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="FirstName" type="xs:string"/>
    <xs:element name="LastName" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```



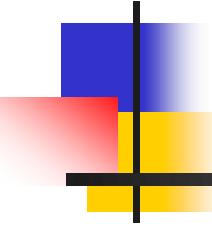
minOccurs and maxOccurs 1-3

- **minOccurs**

- It specify the minimum number of occurrences of the element in an XML document.

- **maxOccurs**

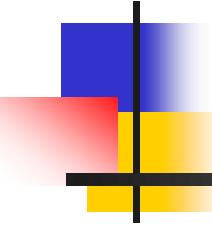
- It specify the maximum number of occurrences of the element in an XML document.



minOccurs and maxOccurs 2-3

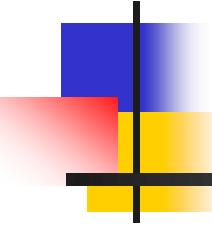
Code Snippet

```
...
<xs:element name= "Books">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ISBN" type="xs:string"/>
      <xs:element name="Quantity" type="xs:string"
maxOccurs="100" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
```



minOccurs and maxOccurs 3-3

minOccur	MaxOccur	Number of times an element can occur
0	1	0 or 1
1	1	1
0	*	Infinite
1	*	At least once
>0	*	At least minOccurs times
>maxOccurs	>0	0
Any value	<minOccurs	0

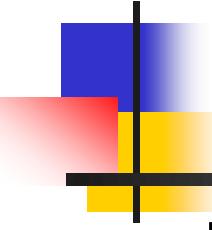


Element Content 1-2

Code Snippet

```
// Books.xml

<?xml version="1.0"?>
<Books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation= "Books.xsd">
<Title>A cup of tea</Title>
<Author>
<Name>Dennis Compton</Name>
</Author>
<Author>
<Name>George Ford</Name>
</Author>
<Publisher>
<Name>Orange</Name>
</Publisher>
</Books>
```

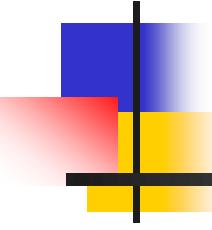


Element Content 2-2

Code Snippet

```
// Books.xsd

<?xml version="1.0"?>
<xss: schema xmlns:xss="http://www.w3.org/2001/XMLSchema">
  <xss:element name= "Books" type= "BookType">
    <xss:complexType name= "AuthorType">
      <xss:sequence>
        <xss:element name= "Name" type="xs:string"/>
      </xss:sequence>
    </xss:complexType>
    <xss:complexType name= "PublisherType">
      <xss:sequence>
        <xss:element name= "Name" type= "xs:string"/>
      </xss:sequence>
    </xss:complexType>
    <xss:complexType name= "BookType">
      <xss:sequence>
        <xss:element name= "Title" type= "xs:string"/>
        <xss:element name= "Author" type="ComposerType"
maxOccurs= "unbounded"/>
        <xss:element name= "Publisher" type="PublisherType"
minOccurs="0" maxOccurs= "unbounded"/>
```

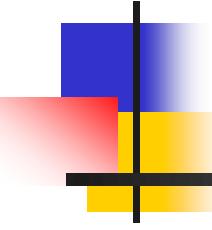


Mixed Content

Code Snippet

```
// Book.xml
<Books>
Apocalypse written by<Author>Mary Jane</Author>
is of Genre<Category>Fiction</Type>.
</Books>
```

```
// Book.xsd
<xs:element name="Books">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="Author" type="xs:string"/>
      <xs:element name="Category" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



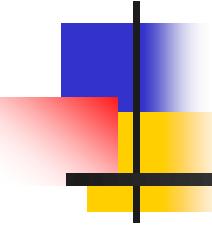
Grouping Constructs 1-3

xs:all

- This grouping construct requires that each element in the group must occur at most once

Code Snippet

```
<xs:element name= "Books">
  <xs:complexType>
    <xs:all>
      <xs:element name="Name" type="xs:string" minOccurs= "1"
      maxOccurs= "1"/>
      <xs:element name="ISBN" type="xs:string" minOccurs= "1"
      maxOccurs= "1"/>
      <xs:element name="items" type="Items" minOccurs= "1" />
    </xs:all>
  </xs:complexType>
</xs:element>
```



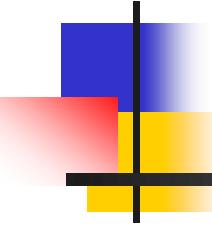
Grouping Constructs 2-3

xs : sequence

- It specifies each member of the sequence to appear in the same order

Code Snippet

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



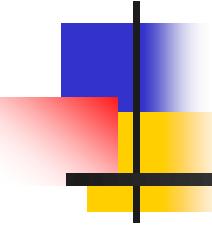
Grouping Constructs 3-3

xs : choice

- It will allow only one of the choices to appear instead of requiring all the elements to be present

Code Snippet

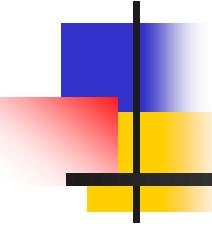
```
<xs:complexType name="AdressInfo">
  <xs:group>
    <xs:choice>
      <xs:element name="Address" type="USAddress" />
      <xs:element name="Address" type="UKAddress" />
      <xs:element name="Address" type="FranceAddress" />
    </xs:choice>
  </xs:group>
```



Lesson 4 – Working with Simple Types

In this last lesson, **Working with Simple Types**, you will learn to:

- Describe simple types.
- List and describe the data types used with simple types.
- Explain restrictions and facets.
- Identify the usage of attributes.



Defining a Simple Type Element

- They are used to form the textual data and specify the type of data allowed within attributes and elements.

Syntax

```
<xs:element name="XXXX" type="YYYY"/>
```

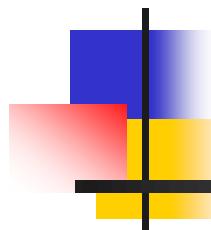
Code Snippet

Book.xml: XML Elements

```
<Book_name>The Da vinci code</Book_name>
<TotalNumberOfPages>360</TotalNumberOfPages>
<Author_name>Dan Brown</Author_name>
```

Book.xsd: Corresponding simple element definitions

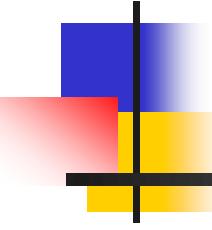
```
<xs:element name="Book_name" type="xs:string" />
<xs:element name="TotalNumberOfPages"
type="xs:integer"/>
<xs:element name="Author_name" type="xs:string"/>
```



Data types used with Simple types

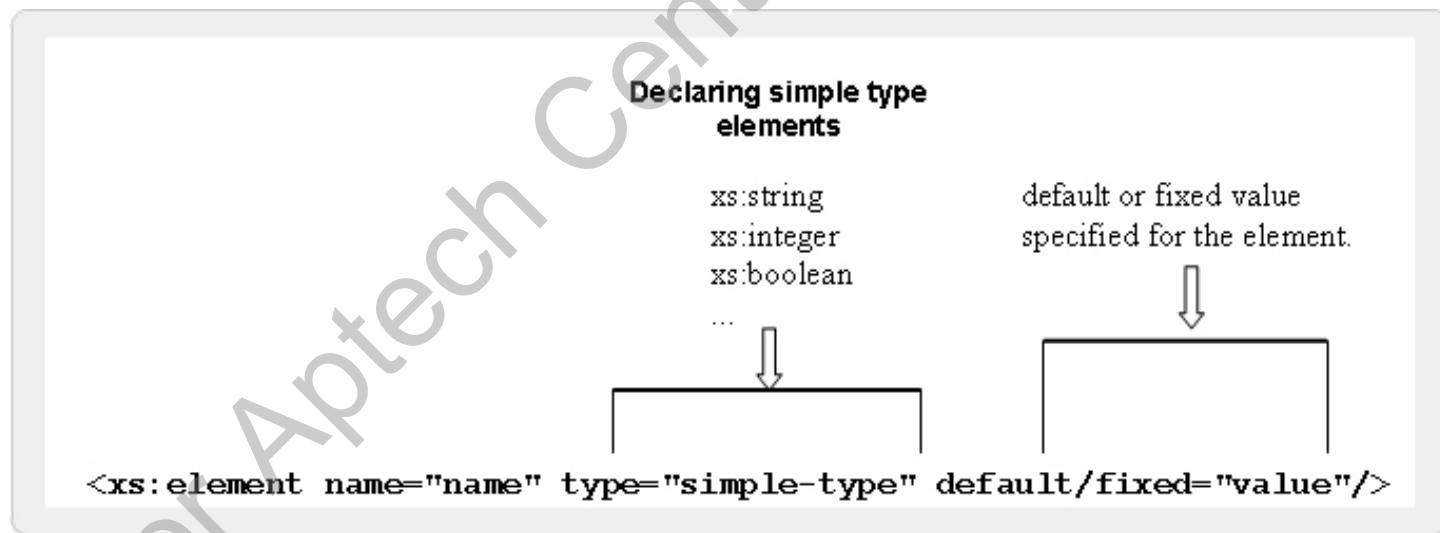
- Built-in simple type
- User-defined simple type

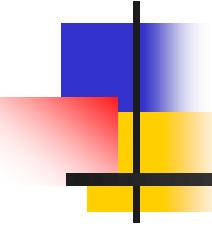
For Aptech Center Use Only



Built-in Simple types 1-2

- There are several built-in simple types, such as integer, date, float and string.
- It can contain a default value or a fixed value.





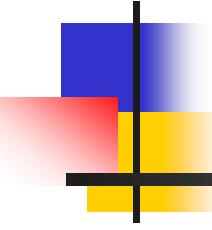
Built-in Simple types 2-2

Syntax

```
<xs:element name="XXXX" type="YYYY" default="ZZZZ"/>
```

Code Snippet

```
<xs:element name="AccountType" type="xs:string"  
fixed="Savings"/>  
<xs:element name="BalanceAmount" type="xs:integer"  
default="5000"/>
```

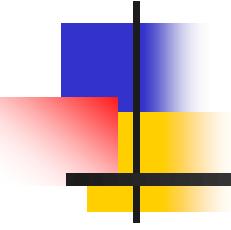


User-defined Simple types 1-2

- Custom user defined datatype can be created using the `<simpleType>` definition.

Syntax

```
<xsd:simpleType name="name of the simpleType">
  <xsd:restriction base="built-in data type">
    <xsd:constraint constraintType="set constraint to limit the
      content"/>
  </xsd:restriction>
</xsd:simpleType>
```



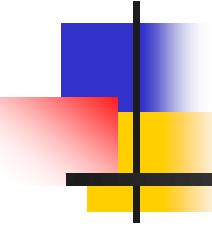
User-defined Simple types 2-2

Code Snippet 1

```
<xs:simpleType name="AngleMeasure">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="360"/>
  </xs:restriction></xs:simpleType>
```

Code Snippet 2

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



Restrictions 1-2

- It can be specified for the simpleType elements.
- They are declared using the <restriction> declaration.

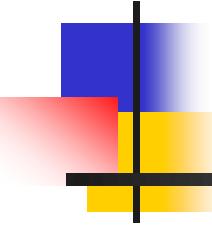
Syntax

```
<restriction base="name of the simpleType you are  
deriving from">
```

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

Code Snippet

```
<xs:simpleType name="Age">  
  <xs:restriction base="xs:integer">  
    ...  
    ...  
  </xs:restriction>  
</xs:simpleType>
```



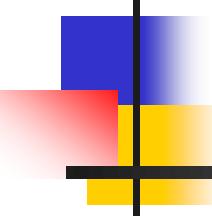
Restrictions 2-2

Syntax

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

Code Snippet

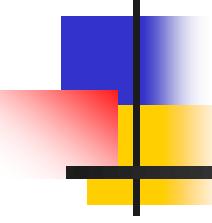
```
<xs:simpleType name="triangle">
    <xs:restriction base="xs:string">
        <xs:enumeration value="isosceles"/>
        <xs:enumeration value="right-angled"/>
        <xs:enumeration value="equilateral"/>
    </xs:restriction>
</xs:simpleType>
```



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.



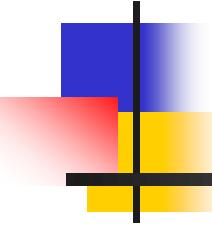
Facets 2-2

Syntax

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

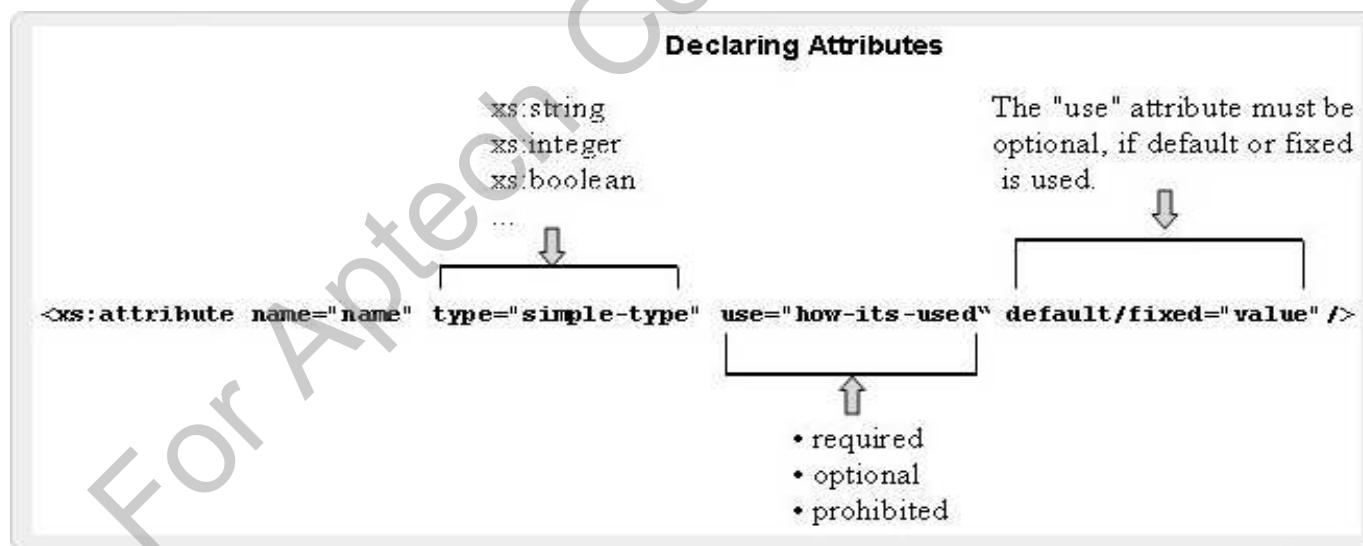
Code Snippet

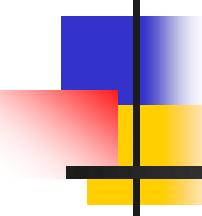
```
<xs:simpleType name="triangle">
  <xs:restriction base="xs:string">
    <xs:enumeration value="isosceles"/>
    <xs:enumeration value="right-angled"/>
    <xs:enumeration value="equilateral"/>
  </xs:restriction>
</xs:simpleType>
```



Attributes 1-3

- Default
- Fixed
- Optional
- Prohibited
- Required





Attributes 2-3

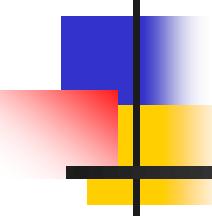
Syntax

```
<xs:attribute name="Attribute_name"  
type="Attribute_datatype"/>
```

where,

Attribute_name is the name of the attribute.

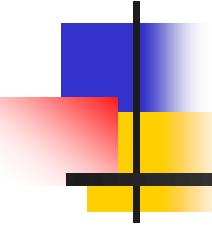
Attribute_datatype specifies the data type of the attribute. There are lot of built in data types in XML schema such as string, decimal, integer, boolean, date, and time.



Attributes 3-3

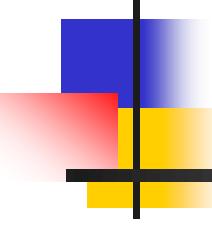
Code Snippet

```
....  
<xss:complexType name= "SingerType">  
  <xss:sequence>  
    <xss:element name= "Name">  
      <xss:complexType>  
        <xss:all>  
          <xss:element name= "FirstName" type="xss:string"/>  
          <xss:element name= "LastName" type="xss:string"/>  
        </xss:all>  
      </xss:complexType>  
    </xss:element>  
  </xss:sequence>  
  <xss:attribute name= "age" type="xss:positiveInteger"  
use= "optional"/>  
</xss:complexType>  
....
```



Summary 1-3

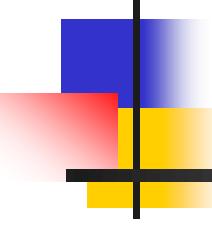
- **XML Schema**
 - An XML Schema is an XML based alternative to DTDs, which describes the structure of an XML document.
 - An XML Schema can define elements, attributes, child elements and the possible values that can appear in a document.
 - Schemas overcome the limitations of DTDs and allow Web applications to exchange XML data robustly, without relying on adhoc validation tools.
- **Exploring XML Schemas**
 - XML schema offers built-in and user defined data types.
 - It supports built-in data types like string, boolean, number, dateTime, binary, and uri.
 - It also supports integer, decimal, time, and user-defined data types.



Summary 2-3

■ Working with Complex Types

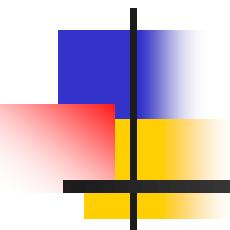
- Elements with complex type may contain nested elements and have attributes.
- A complex element can be defined by directly naming the element and by using the name and the type attribute of the complex type.
- `minOccurs` and `maxOccurs` specify the minimum and maximum number of occurrences of the element in an XML document respectively.
- Element content in an XML document contains only XML elements and mixed content contains text mixed with elements.
- The grouping constructs in XML schema specify the order of XML elements.



Summary 3-3

■ Working with Simple Types

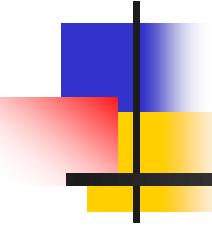
- The elements of simple type describe the content and data type of the element rather than its structure.
- The simple type can have built-in and user defined data types.
- Simple type definition takes one of the two values, default or fixed as per the requirement.
- The user-defined data type can be derived from a built-in type or an existing simple type.
- Use of restrictions and facets restricts its content to user prescribed values.
- Schema supports usage of attributes in elements, which is declared with a simple type definition.



Module 5

Style Sheets

For Aptech Center Use Only

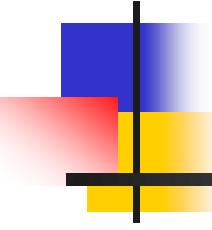


Module Overview

In this module, you will learn about:

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

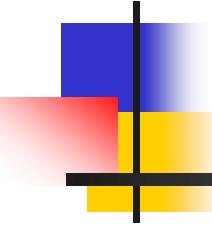
For Aptech Center Use Only



Lesson 1 – Style Sheets

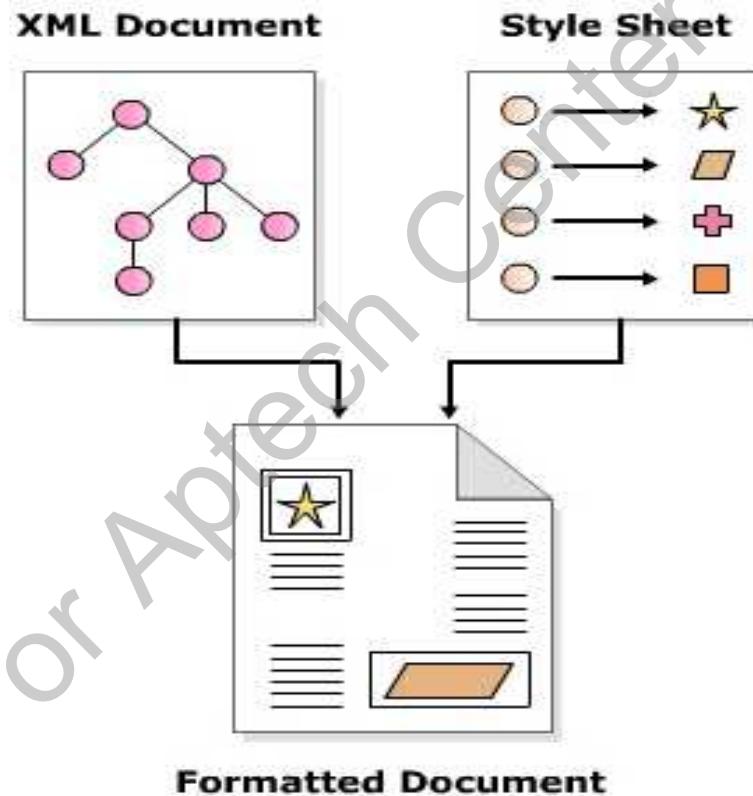
In this first lesson, **Style Sheets**, you will learn to:

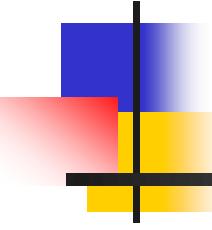
- Define and describe style sheets.
- Define and describe cascading style sheets (CSS).
- Explain how to implement styles with CSS.



Need of Style Sheets

- They are a set of rules that describe the appearance of data in an XML document.





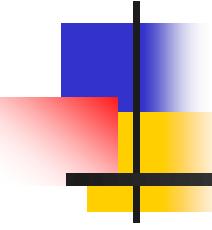
Various Style Sheets

Cascading Style Sheet (CSS)

- It allows you to control the appearance of data in HTML and XML documents.

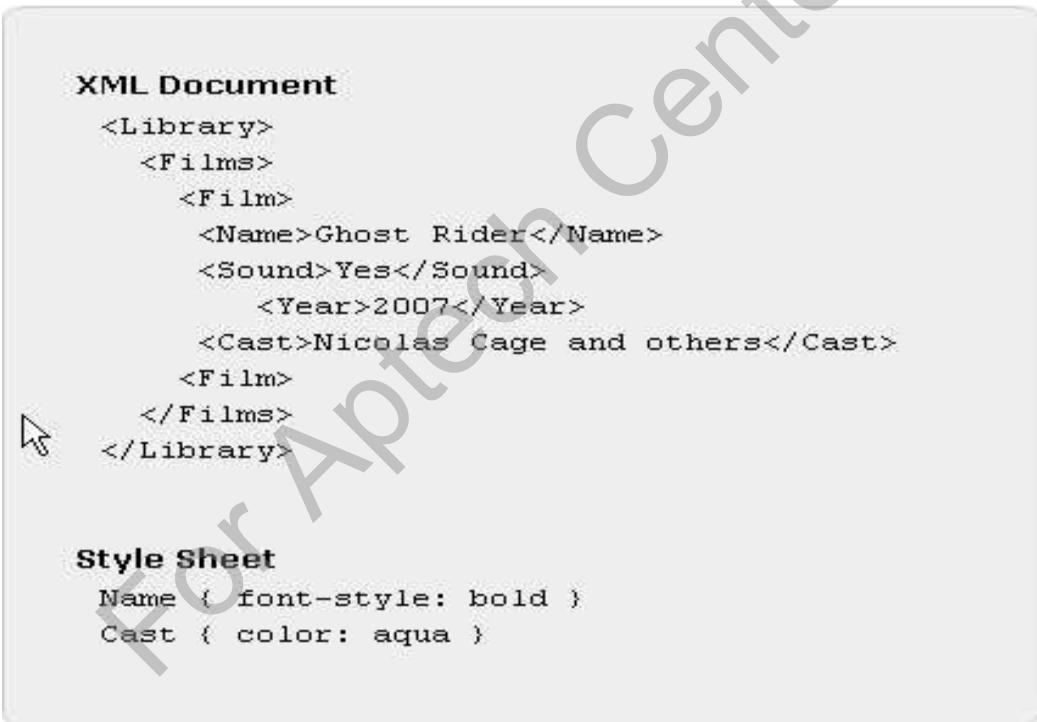
Extensible Style Sheet (XSL)

- It is used to define the appearance of data contained only in XML documents.



Cascading Style Sheets

- It comprises a set of rules for various elements in a document.
- Each rule defines how the data enclosed in the element should be displayed.
- The term cascading is derived from this ability to mix and match rules.

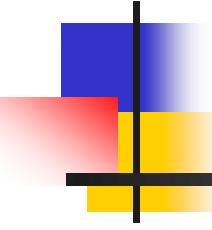


XML Document

```
<Library>
  <Films>
    <Film>
      <Name>Ghost Rider</Name>
      <Sound>Yes</Sound>
      <Year>2007</Year>
      <Cast>Nicolas Cage and others</Cast>
    <Film>
  </Films>
</Library>
```

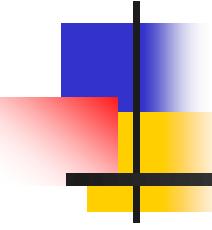
Style Sheet

```
Name { font-style: bold }
Cast { color: aqua }
```



Benefits of CSS

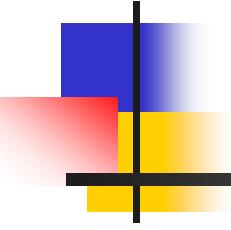
- Any style or presentation changes to data can be achieved faster as changes are to be implemented in one place.
- Device independence can be achieved by defining different style sheets for different device.
- Reduction in document code as the presentation information is stored in a different file and can be reused.



Style Rules

- They define how the content enclosed in an element will be displayed.
- These rules are applicable to all child elements within an element.
 - **Selector**
 - **Property**
 - **Value**

```
selector {property: value}
```



Ways of Writing Style rules

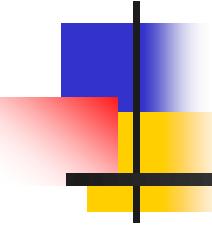
- A single selector can have more than one property-value pairs associated with it.
- A collection of one or more property-value pairs can be associated with more than one selector.

Single selector, multiple property declarations

```
CD { font-family: sans-serif; color: black }
```

Multiple selectors, multiple property declarations

```
CD, Name, Title { font-family: times; color: black }
```



External Style Sheets

- 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.

Syntax

```
<?xml-stylesheet href="headers.css" type="text/css"?>
```

where,

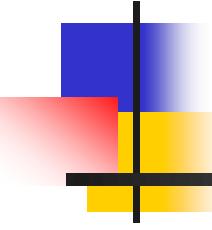
`xml-stylesheet` is the processing instruction.

`url` is the URL of a .css file; the .css file can be on a local system or anywhere on the Internet.

`type="text/css"` is optional; however if a browser does not support CSS, it informs the browser that it does not have to download a .css file.

Code Snippet

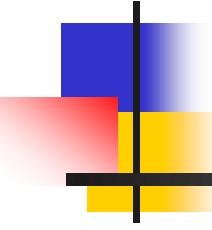
```
<?xml-stylesheet href="url" [type="text/css"] ?>
```



Lesson 2 – Selectors in CSS

In this second lesson, **Selectors in CSS**, you will learn to:

- Identify simple selectors in CSS.
- State the use of universal selector in CSS.
- Describe ID selectors.

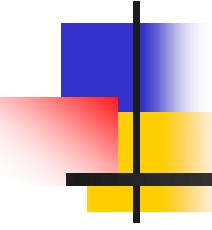


Simple Selectors

- It comprises an element name followed by one or more property declarations.
- It matches every occurrence of the element in a document.

Code Snippet

```
/* Simple selector */
CD { color: black }
/* Single element, multiple property declarations */
CD { color: white; background-color: blue }
/* Multiple elements, multiple property declarations */
CD, Name, Title { color: white; background-color: blue }
```



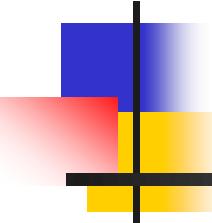
Universal Selector

- It comprises an asterisk followed by property declarations.
- It matches all the elements in a document.

Code Snippet

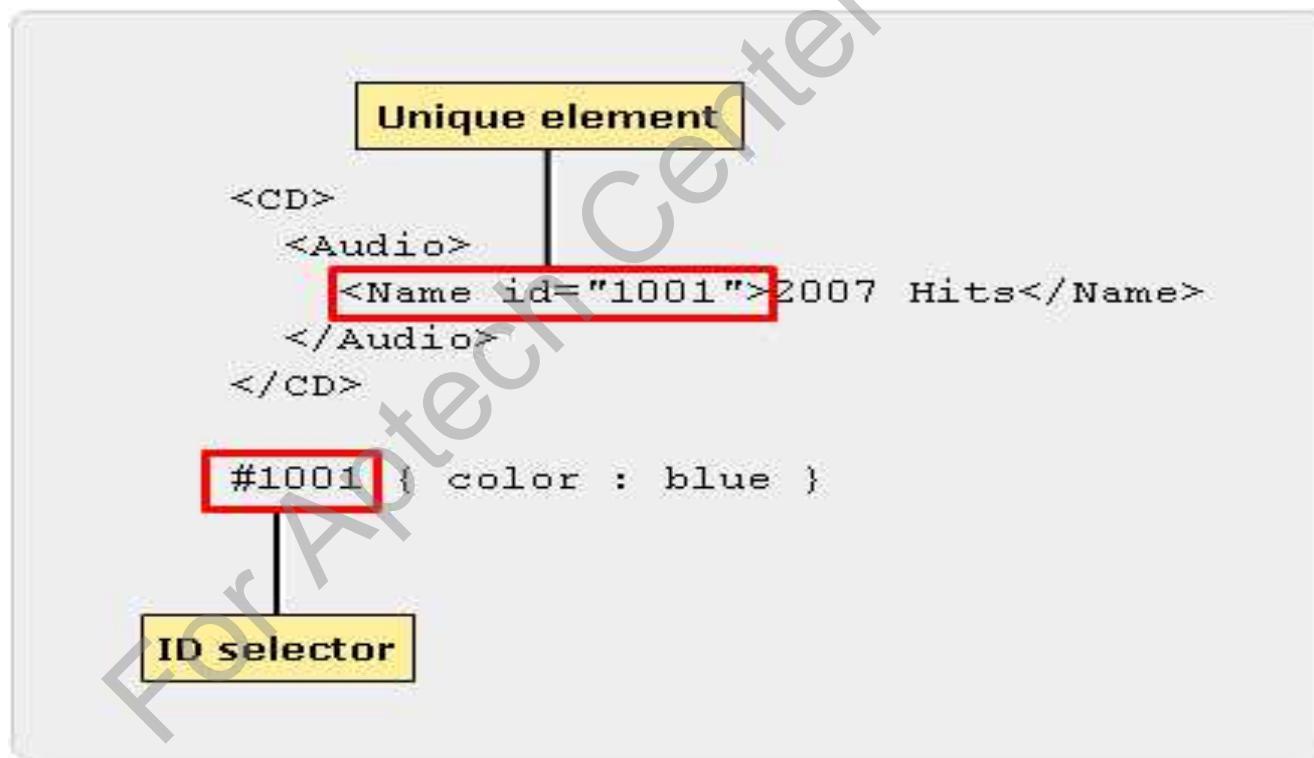
```
* { color : blue }
```

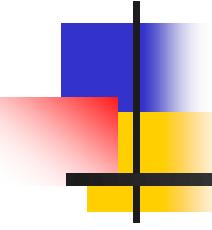
Displays the content of all elements in a document in blue.



ID Selector 1-2

- It 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.





ID Selector 2-2

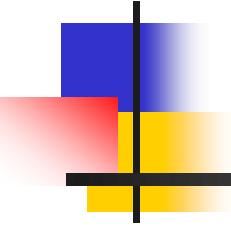
Syntax

```
#attribute_value { property_declarations }
```

Code Snippet

```
#1001 { color : blue }
```

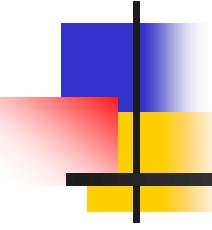
Displays the content of an element in blue if its id attribute's value equals 1001.



Lesson 3 – Properties and Values

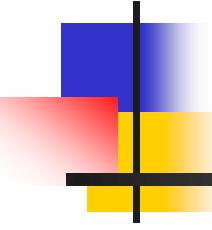
In this third lesson, **Properties and Values**, you will learn to:

- State and describe how to use color properties.
- Describe the font property.
- Describe the other properties such as margins, borders, padding.
- Explain briefly about positioning and alignment.



Color Properties

Color Names	RGB Percentages	RGB Values	Hexadecimal Values
aqua	rgb(0%, 65%, 65%)	rgb(0, 160, 160)	#00a0a0
black	rgb(0%, 0%, 0%)	rgb(0, 0, 0)	#000000
blue	rgb(0%, 32%, 100)	rgb(0, 80, 255)	#0050ff
gray	rgb(65%, 65%, 65%)	rgb(160, 160, 160)	#a0a0a0
green	rgb(0%, 100%, 0%)	rgb(0, 255, 0)	#00ff00
lime	rgb(0%, 65%, 0%)	rgb(0, 160, 0)	#00a000
maroon	rgb(70%, 0%, 32%)	rgb(176, 0, 80)	#b00050
navy	rgb(0%, 0%, 65%)	rgb(0, 0, 160)	#0000a0
olive	rgb(65%, 65%, 0%)	rgb(160, 160, 0)	#a0a000
purple	rgb(65%, 0%, 65%)	rgb(160, 0, 160)	#a000a0
red	rgb(100%, 0%, 32%)	rgb(255, 0, 80)	#ff0050
silver	rgb(90%, 90%, 90%)	rgb(225, 225, 255)	#d0d0d0
teal	rgb(0%, 65%, 100%)	rgb(0, 160, 255)	#00a0ff
white	rgb(100%, 100%, 100%)	rgb(255, 255, 255)	#ffffff
yellow	rgb(100%, 100%, 0%)	rgb(255, 255, 0)	#ffff00



Setting Color Properties

Syntax

```
color: colorValue  
background-color: colorValue
```

where,

color

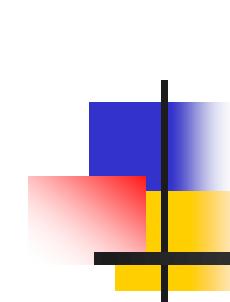
Property to set the foreground color of text in an element.

colorValue

colorValue can take up any value from the CSS color table.

background-color

Property to set the background color of text in an element.

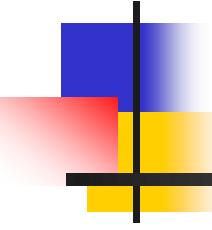


Setting Color Properties

2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Colors.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11   </Cars>
```



Setting Color Properties

3-3

Style Sheet

```
1 Cars { background-color: rgb(0%,32%,100%); color: #fffff }
```

```
2 Price { color: yellow; }
```

where,

```
Cars { background-color:rgb(0%,32%,100%);color:  
#fffffff }
```

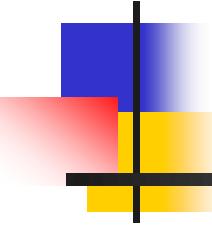
Causes the text enclosed in Cars element to be displayed in white color with a background color of blue.

```
Price{color:yellow;}
```

Causes the text enclosed in Price element to be displayed in yellow color.

Output

BMW M3 Metallic Silver 40,000 Liverpool



Font Properties

Property Name	Description
font-family	To specify the font family
font-size	To specify the size of font
font-style	To specify the style of font
font-weight	To specify the weight of font



Font-family Property 1-2

Syntax

```
font-family : "font-family name(s)"
```

where,

font-family

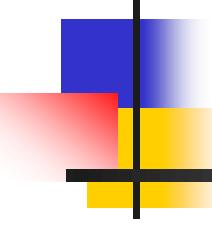
Property to specify the font-family to be used.

font-family name(s)

Comma separated list of font-family names such as serif, san-serif, monospace, cursive, and fantasy. The list should start with the most specific font in which you want to display the data and end with the most generic font.

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="FontFamily.css" type="text/css"?>
3  <Cars>
4  <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11 </Cars>
```



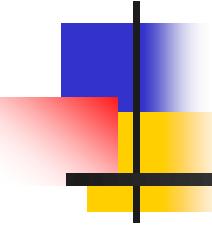
Font-family Property 2-2

Style Sheet

```
1 Cars { font-family: "arial, times, serif" }
```

Output

BMW M3 Metallic Silver 40,000 Liverpool



Font-size Property 1-3

Syntax

```
font-size : "xx-small | x-small | small | medium | large | x-large | xx-large"
```

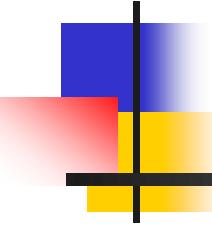
where,

font-size

Property to specify the size of font.

xx-small|x-small|small|medium|large|x-large|xx-large

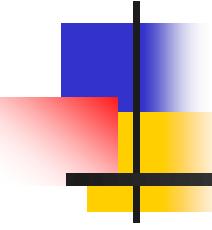
One of various values that can be assigned to the property font-size.



Font-size Property 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="FontSize.css" type="text/css"?>
3  <Cars>
4  <Vehicle>
5    <Manufacturer>BMW</Manufacturer>
6    <Model>M3</Model>
7    <Color>Metallic Silver</Color>
8    <Price>40,000</Price>
9    <Location>Liverpool</Location>
10   </Vehicle>
11 </Cars>
```



Font-size Property 3-3

Style Sheet

```
1 Cars { font-size: medium }
```

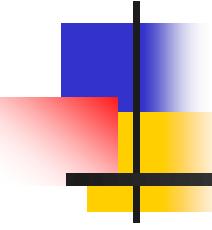
where,

```
Cars { font-size: medium }
```

All the text enclosed in Cars element and its child elements will be displayed with medium font size.

Output

BMW M3 Metallic Silver 40,000 Liverpool



Font Style and Weight Properties 1-3

Syntax

```
font-style: normal | oblique | italic  
font-weight: light | normal | bold
```

where,

`font-style`

Property to specify the style of text in an element.

`normal | oblique | italic`

One of the values that can be assigned to `font-style` property.

`font-weight`

Property to specify the weight style of the text in an element.

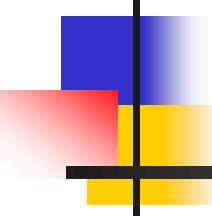
`light | normal | bold`

One of the values that can be assigned to `font-weight` property.

Font Style and Weight Properties 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="FontStyle.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11   </Cars>
```



Font Style and Weight Properties 3-3

Style Sheet

- 1 Manufacturer { font-weight: bold }
- 2 Location { font-style: italic }

where,

Manufacturer { font-weight: bold }

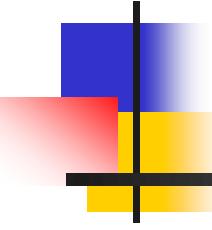
The text enclosed in Manufacturer element will be displayed in bold.

Location {font-style: italic}

The text enclosed in Location element will be displayed in italics.

Output

BMW M3 Metallic Silver 40,000 *Liverpool*



Margins in CSS 1-3

Syntax

```
margin-left | margin-right | margin-top | margin-bottom =  
marginValue
```

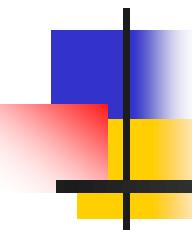
where,

margin-left | margin-right | margin-top | margin-bottom

The various margin properties to set left, right, top and bottom margins.

marginValue

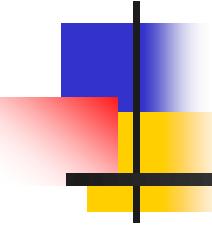
The value to be assigned to one of the margin properties. This value can be a fixed value or a percentage.



Margins in CSS 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Margin.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11   </Cars>
```



Margins in CSS 3-3

Style Sheet

```
1 Manufacturer { margin-left: 20; margin-right: 50; }
2 Manufacturer { background-color: aqua }
3 Vehicle { background-color: orange }
```

where,

Manufacturer {margin-left:20; margin-right:50;}

Inserts a space of 20 pixels to the left and a space of 50 pixels to the right of text enclosed in element Manufacturer.

Manufacturer {background-color: aqua}

Sets the background of text enclosed in element Manufacturer to aqua.

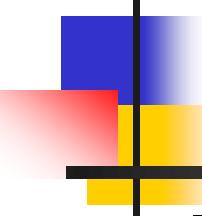
Vehicle {background-color: orange}

Sets the background of text enclosed in element Vehicle to orange.

Output

BMW

M3 Metallic Silver 40,000 Liverpool



Border Properties in CSS 1-3

Syntax

```
border : border_width border_style border_color
```

where,

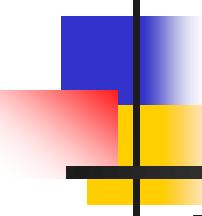
border: Property to set the border of the box surrounding an element's data.

border_width: Specifies the thickness of the border.

Possible values are thin, medium and thick.

border_style: Specifies the style of the border. Possible values are solid, dashed, dotted, groove, ridge, double, inset, outset.

border_color: Specifies the color of the border. All values that are applicable to CSS color property are allowed.



Border Properties in CSS 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Border.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11   </Cars>
```

Border Properties in CSS 3-3

Style Sheet

```
1 Manufacturer {border: thick dashed magenta }
2 Model { border: thick solid olive }
3 Color { border: thick groove aqua }
4 Price { border: thick inset gray }
```

where,

Manufacturer {border: thick dashed magenta }

Displays a thick and dashed magenta border around the content of Manufacturer element.

Model { border: thick solid olive }

Displays a thick and solid olive border around the content of Model element.

Color {border: thick groove aqua}

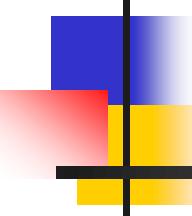
Displays a thick and groove aqua border around the content of Color element.

Price {border: thick inset gray}

Displays a thick and inset gray border around the content of Price element.

Output





Padding Properties in CSS 1-2

Syntax

```
padding : padding_width
```

Code Snippet

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet href="Padding.css" type="text/css"?>
3 <Cars>
4   <Vehicle>
5     <Manufacturer>BMW</Manufacturer>
6     <Model>M3</Model>
7     <Color>Metallic Silver</Color>
8     <Price>40,000</Price>
9     <Location>Liverpool</Location>
10    </Vehicle>
11 </Cars>
```

Padding Properties in CSS 2-2

Style Sheet

```
1 Manufacturer { border: thick dashed magenta }
2 Model { border: thick solid olive }
3 Color { border: thick groove aqua }
4 Price { border: thick inset gray }
5 Manufacturer { padding: 2 }
6 Model { padding: 2 5 }
7 Color { padding: 2 5 8 }
8 Price { padding: 2 5 8 10 }
```

where,

Manufacturer { padding: 2 }

Inserts padding of 2 pixels between the four borders and text of Manufacturer element. The four borders applicable are top, right, bottom, and left.

Model { padding: 2 5 }

Inserts padding between the borders and text of Model element. The value 2 is applied to top and bottom borders and 5 is applied to left and right borders.

Color { padding: 2 5 8 }

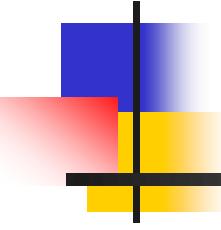
Inserts padding between the borders and text of Color element. The value 2 is applied to top border, 5 to left and right borders, and value 8 to bottom border.

Price { padding: 2 5 8 10 }

Inserts padding between the borders and text of Price element. The value 2 is applied to top border, 5 to right border, 8 to bottom border, and 10 to left border.

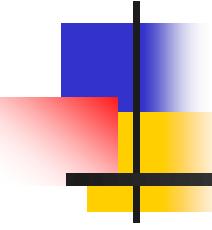
Output





CSS Units

Unit Type	Unit Designator
Relative	em – defines the height of element's font. ex – defines the x-height of element's font. px – defines the pixel relative to display device. % - percentage.
Absolute	in – inches cm – centimeters mm – millimeters pt – 1/72 inch pc – 12 pt



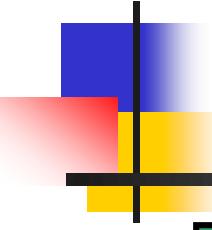
Position Properties

Property	Description	Possible Values
position	Property to place an element in a static, relative, absolute or fixed position	static, fixed, relative or absolute
top	Property specifying how far the top edge of an element is above/below the top edge of the parent element	auto, integer or floating point values adhering to CSS length units
left	Property specifying how far the left edge of an element is to the right/left of the left edge of the parent element	auto, integer or floating point values adhering to CSS length units
bottom	Property specifying how far the bottom edge of an element is above/below the bottom edge of the parent element	auto, integer or floating point values adhering to CSS length units
right	Property specifying how far the right edge of an element is to the left/right of the right edge of the parent element	auto, integer or floating point values adhering to CSS length units

Position Properties Example 1-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Position.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11   </Cars>
```



Position Properties Example 2-3

Style Sheet

```
1 Location { position: relative; top: 20; left: 20 }  
2 Price { position: absolute; top: 40; left: 20 }
```

where,

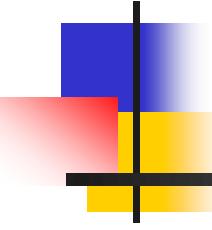
```
Location {position:relative;top:20;left:20}
```

Positions the text of Location element relative to the previous element.

However, inside the box, the content is placed at 20 pixels from top and 20 pixels from left.

```
Price {position:absolute;top:40;left:20}
```

Positions the text of Price element at an absolute location of 40 pixels from the top and 20 pixels from the left.



Position Properties Example 3-3

Output

BMW M3 Metallic Silver

40,000

Liverpool

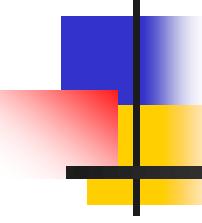
where,

40,000

Text displayed using absolute positioning

Liverpool

Text displayed using relative positioning



Display Property 1-3

Syntax

```
display: value
```

```
where value = none | inline | block
```

where,

display

Property to specify how the element is to be rendered.

none

No rendering is applied to the element.

inline

Displays the element text as in line. This is the default value if no value is specified.

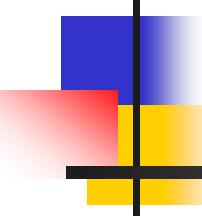
Block

Displays the element text on a new line in a block of its own.

Display Property 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Display.css" type="text/css"?>
3  <Cars>
4      <Vehicle>
5          <Manufacturer>BMW</Manufacturer>
6          <Model>M3</Model>
7          <Color>Metallic Silver</Color>
8          <Price>40,000</Price>
9          <Location>Liverpool</Location>
10     </Vehicle>
11     <Vehicle>
12         <Manufacturer>TOYOTA</Manufacturer>
13         <Model>INNOVA</Model>
14         <Color>Gray</Color>
15         <Price>38,000</Price>
16         <Location>California</Location>
17     </Vehicle>
18 </Cars>
```



Display Property 3-3

Style Sheet

```
1 | Price { display: block }
```

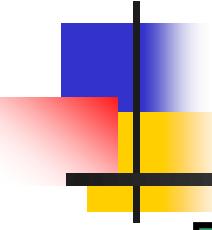
where,

```
Price { display: block }
```

Inserts a new line before and after the text of element Price.

Output

```
BMW  
M3 Metallic Silver  
40,000  
Liverpool  
TOYOTA  
INNOVA Gray  
38,000  
California
```



Text Alignment and Indentation 1-3

Syntax

```
text-align : alignment_value
```

```
text-indent : value
```

where,

text-align

Property to align the text in a block.

alignment_value

Can be one of the following: left(default), right, center and justify.

text-indent

Property to indent the text in a block.

value

Floating point value followed by absolute units designators or relative units designators; or an integer value followed by percentage (%) symbol.

Text Alignment and Indentation 2-3

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet href="Align.css" type="text/css"?>
3  <Cars>
4    <Vehicle>
5      <Manufacturer>BMW</Manufacturer>
6      <Model>M3</Model>
7      <Color>Metallic Silver</Color>
8      <Price>40,000</Price>
9      <Location>Liverpool</Location>
10     </Vehicle>
11    <Vehicle>
12      <Manufacturer>TOYOTA</Manufacturer>
13      <Model>INNOVA</Model>
14      <Color>Gray</Color>
15      <Price>38,000</Price>
16      <Location>California</Location>
17    </Vehicle>
18  </Cars>
```

Text Alignment and Indentation 3-3

Style Sheet

```
1 Price { display: block }  
2 Price { text-align: left }  
3 Price { text-indent: 20 }
```

where,

Price {display: block}

Inserts a new line before and after the text of element Price and displays the text in a separate block.

Price {text-align: left}

Left aligns the text of Price element.

Price {text-indent: 20}

Indents the text of Price element at 20 pixels.

Output

BMW M3 Metallic Silver

40,000

Liverpool TOYOTA INNOVA Gray

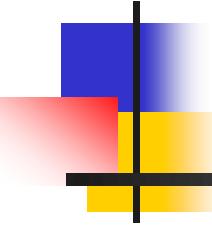
38,000

California

where,

40,000

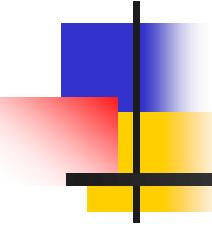
The text in element Price is display on a separate row and left indented at 20 pixels.



Lesson 4 - Inheritance and Cascades in CSS

In this last lesson, **Inheritance and Cascades in CSS**, you will learn to:

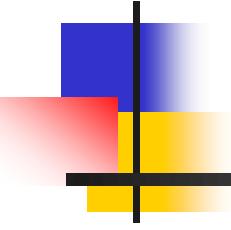
- Define the process of cascading.
- Explain inheritance.



Cascading in CSS

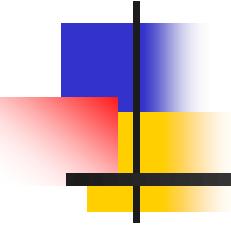
- All declarations that apply to an element and property are grouped.
- Declarations are sorted by weight and origin.
- Declarations are sorted by specificity of selector.
- Declarations are sorted by the order specified.

For Aptech
Conferenc
e Only



Inheritance in CSS 1-2

- Inheritance is the ability of one entity to acquire the characteristics of another entity.
- In CSS, the child elements inherit the styles 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.



Inheritance in CSS 2-2

XML Document

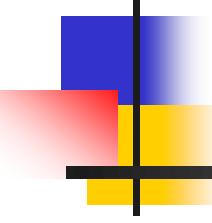
```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="CD.css" type="text/css"?>
<CD>
    <Audio>
        <Name>2007 Hits</Name>
    </Audio>
</CD>
```

Style Sheet

```
CD { font-style: italic }
```

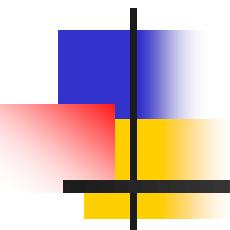
Output

2007 Hits



Summary

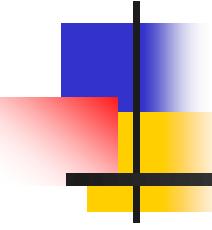
- **Style Sheets**
 - Style Sheets are a set of rules that define the appearance of data.
 - These rules are written in a file with the extension .css.
 - The .css file is associated with an XML document using the xml-processing instruction.
- **Selectors in CSS**
 - Selectors define the elements to which the styles will be applied.
 - The various types of selectors are simple, universal and ID selectors.
- **Properties and Values**
 - CSS provides several properties to define the appearance of data.
 - Some of these properties are color, background-color, position, padding, font, text-align to name a few.
- **Inheritance and Cascades in CSS**
 - In CSS, a child element inherits the styles rules applied to its ancestor element.
 - Also there are several sources of style sheets, hence CSS follows W3C defined cascading order when applying style rules to elements.



Module 6

XSL and XSLT

For Aptech Center Use Only

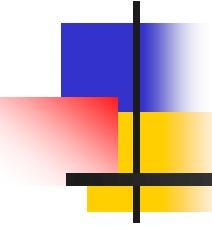


Module Overview

In this module, you will learn about:

- Introduction to XSL
- Working with XSL

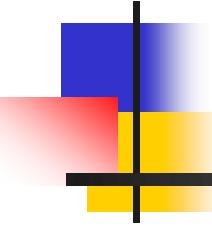
For Aptech Center USE Only



Lesson 1 – Introduction to XSL

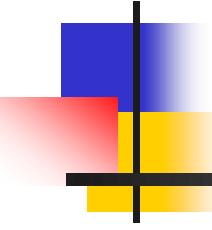
In this first lesson, **Introduction to XSL**, you will learn to:

- Define Extensible Stylesheet Language (XSL), Extensible Stylesheet Language Transformations (XSLT) and their purpose.
- Explain the structure and syntax of XSL.
- Distinguish between Cascading Style Sheet (CSS) and XSL.



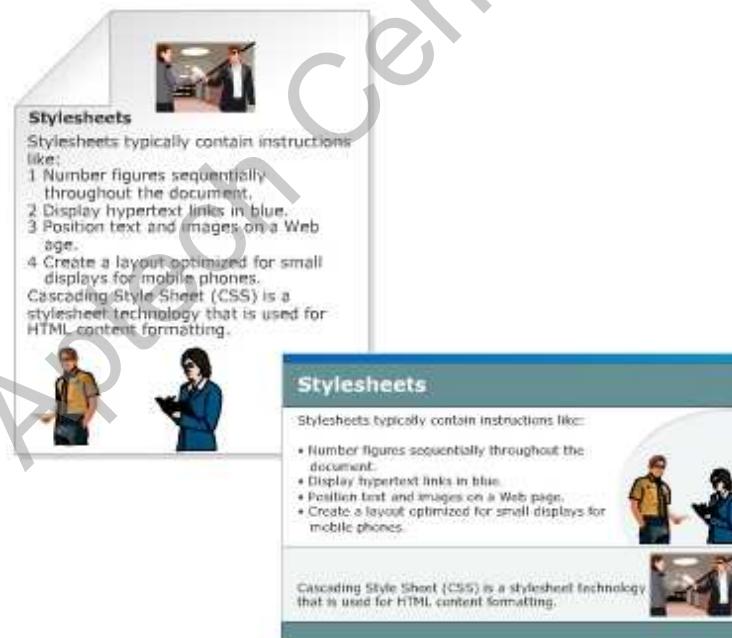
Stylesheets 1-2

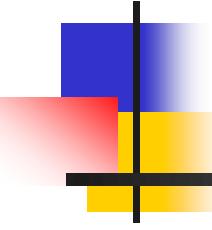
- It is a collection of commands that tells a processor how to render the visual appearance of content in a web page.
- Stylesheets typically contain instructions like:
 - Number figures sequentially throughout the document.
 - Display hypertext links in blue.
 - Position text and images on a Web page.
 - Create a layout optimized for small displays for mobile phones.



Stylesheets 2-2

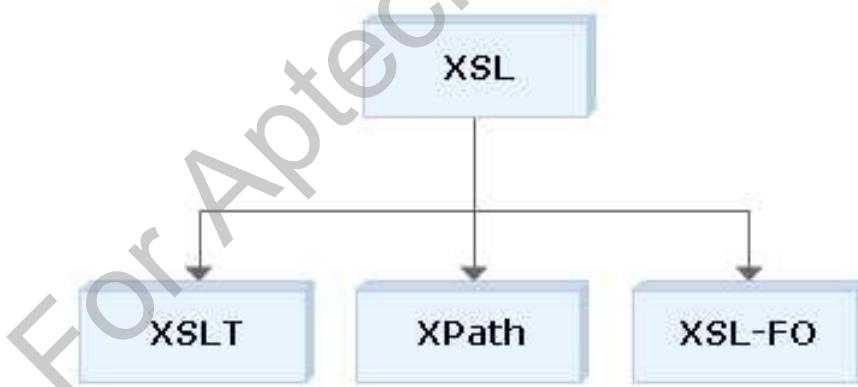
- CSS is used for HTML content formatting.
- XSL is used to describe how the XML document should be displayed.

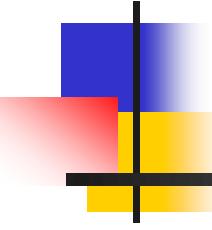




Extensible Stylesheet Language (XSL)

- XSL Transformations (XSLT)
 - An XML language for transforming XML documents.
- XML Path Language (XPath)
 - A language for navigating the XML document.
- XSL Formatting Objects (XSL-FO)
 - An XML language for formatting XML documents.

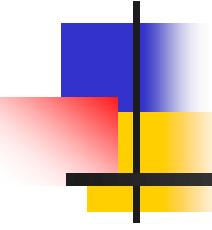




XSL Transformations

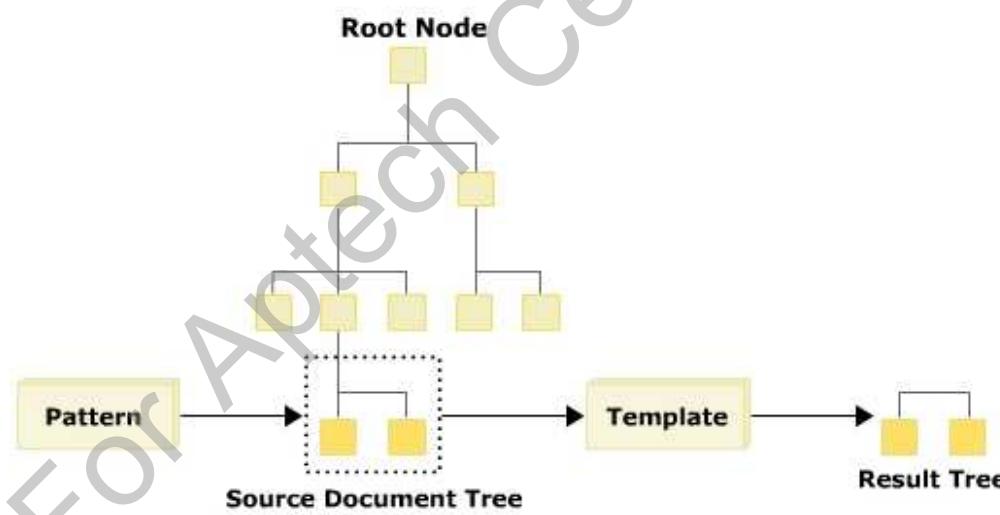
- The transformation component of the XSL stylesheet technology is XSLT.
- It describes the process of transforming an XML document, using a transformation engine and XSL.

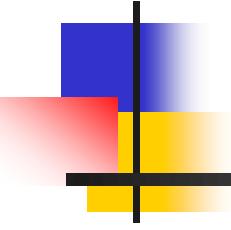
For Aptech Center USE Only



XSL Processing Model

- It reads an XML document and processes it into a hierarchical tree.
- It starts with the root node in the tree and performs pattern matching in the stylesheet.





XSLT Structure and Syntax

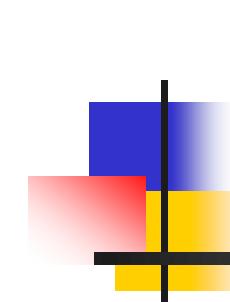
- It uses a standard document introduction, matching closing tags for any opening tags that contain content, and a proper syntax for empty elements.
- The style rules are written in a file with the extension `.xsl`.

Syntax

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

where,

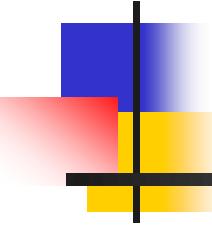
`<xsl:stylesheet>`: Root element of the stylesheet.
`xmlns:xsl="http://www.w3.org/1999/XSL/Transform"`: refers to the official W3C XSLT namespace. You must include the attribute `version="1.0"` if you use this namespace.



Top Level XSLT Elements

- An element occurring as a child of an `xsl:stylesheet` element is called a top-level element.
- It can occur directly inside the `xsl:stylesheet` element.

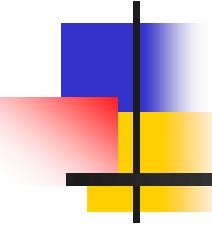
Element Name	Description
<code>xsl:attribute-set</code>	Adds a list of attributes to the output node tree
<code>xsl:import</code>	Used to import contents of one stylesheet into another. The importing stylesheet takes precedence over the imported stylesheet
<code>xsl:namespace-alias</code>	Replaces source document Namespace with a new Namespace in the output tree node
<code>xsl:output</code>	Specifies the output for the result tree. It contains a list of attributes. The most important one is the <code>method</code> attribute which dictates if the type of output is HTML, text, or XML
<code>xsl:template</code>	Used to define a template that can be applied to a node to produce a desired output display
<code>xsl:variable</code>	Defines a <code>variable</code> in a stylesheet or template, and to assign it a value



CSS and XSL

- They are two different style languages recommended by the World Wide Web Consortium (W3C).
- XSL is more powerful and complex than CSS.

CSS	XSL
Stylesheet language to create a style for HTML and XML documents	Stylesheet language to create a style for XML documents
Determines the visual appearance of a page, but does not alter the structure of the source document	Provides a means of transforming XML documents
Does not support decision structures and it cannot calculate quantities or store values in variables	Supports decision structures and can calculate quantities or store values in variables
Uses its own notation	Uses an XML notation
Highly effective and easy to learn for simple applications	Designed to meet the needs of more complex applications for richer style sheets



Lesson 2 – Working with XSL

In this last lesson, **Working with XSL**, you will learn to:

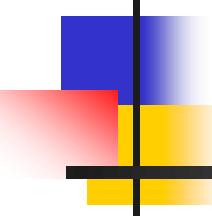
- Explain XSL templates.
- Describe the use of `select` attribute.
- State how to use `xsl:value-of` element.
- Describe how to use `xsl:for-each` element.
- Explain briefly how to use `xsl:text` element.
- Describe how to use `xsl:number` element.
- Describe how to use `xsl:if` element.
- Describe how to use `xsl:choose` element.
- Explain how to perform sorting using XSL.

XSL Templates

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="customer.xsl"?>
<CustomerList>
  <Customer>
    <Name>
      <First>David</First>
      <Last>Blake</Last>
    </Name>
    <Order>100 brown suitcases</Order>
    <Order>12 bottles wine</Order>
  </Customer>

<TABLE BORDER="1" bgcolor = "pink">
  <xsl:for-each select="CustomerList/Customer">
    <TR>
      <TH ALIGN="left">
        <xsl:apply-templates select="Name"/>
      </TH>
    </TR>
    <xsl:for-each select="Order">
      <TR>
        <TD>
          <xsl:apply-templates/>
        </TD>
      </TR>
    </xsl:for-each>
  </xsl:for-each>
</TABLE>
```

Blake , David
100 brown suitcases
12 bottles wine
Honai , John
120 red T-shirts
120 bottles mango juice



The xsl:template Element 1-2

- It is used to define a template that can be applied to a node to produce desired output.

Syntax

```
<xsl:template  
    match="pattern"  
    mode="mode"  
    name="name"  
    priority="number"  
>  
</xsl:template>
```

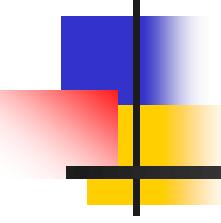
where,

match: Is a pattern that is used to define which nodes will have which template rules applied to them. If this attribute is omitted there must be a **name** attribute.

mode : Allows the same nodes to be processed more than once.

name: Specifies a name for the template. If this attribute is omitted there must be a **match** attribute.

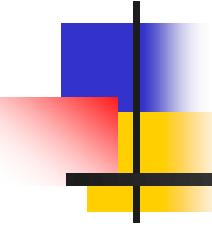
priority: Is a real number that sets the priority of importance for a template. The higher the number, the higher the priority.



The xsl:template Element 2-2

Code Snippet

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0"
3      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4      <xsl:template match="/">
5          <html>
6              <body>
7                  <h1> XSL TEMPLATE SAMPLE</h1>
8              </body>
9          </html>
10     </xsl:template>
11 </xsl:stylesheet>
```



The xsl:apply-templates element 1-4

- It defines a set of nodes to be processed.

Syntax

```
<xsl:apply-templates  
    select="expression"  
    mode="name"  
    >  
</xsl:apply-templates>
```

where,

select

Used to process nodes selected by an expression.

mode

Allows the same nodes to be processed more than once. Each time the nodes are processed, they can be displayed in a different manner.

The xsl:apply-templates element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="GEM_Sheet.xsl"?>
3  <GEMEmployees>
4      <Designer>
5          <Name>David Blake</Name>
6          <DOJ>18/11/1973</DOJ>
7          <Address>512-B Lamington Road</Address>
8          <Phone>1564-754-111</Phone>
9      </Designer>
10     <Designer>
11         <Name>Susan Jones</Name>
12         <DOJ>03/05/1953</DOJ>
13         <Address>Palm Beach Road</Address>
14         <Phone>8755-211-111</Phone>
15     </Designer>
16 </GEMEmployees>
```

The xsl:apply-templates element 3-4

Style Sheet

```
1 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
2   <xsl:template match="GEMEmployees/Designer">
3     <html>
4       <body>
5         <xsl:apply-templates select="Name"/>
6         <xsl:apply-templates select="DOJ"/>
7         <br/>
8       </body>
9     </html>
10   </xsl:template>
11   <xsl:template match="Name">
12     Name:
13     <span style="font-size:22px; color:green">
14       <xsl:value-of select="."/>
15     </span>
16     <br/>
17   </xsl:template>
18   <xsl:template match="DOJ">
19     Date of Join:
20     <span style="color:blue;">
21       <xsl:value-of select="."/>
22     </span>
23     <br/>
24   </xsl:template>
25 </xsl:stylesheet>
```

where,

xsl:match="GEMEmployees/Designer"

Represents the Designer child element by specifying the GEMEmployees parent element.

xsl:apply-templates select="Name"

Applies the template on Name element.

xsl:apply-templates select="DOJ"

Applies the template on DOJ element.

xsl:value-of

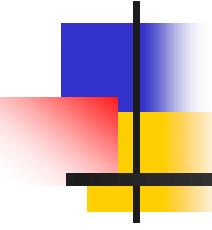
Used to extract the value of a selected node.

xsl:template match="Name"

If the template is matched with Name element, the value of the Name element is displayed in green color with font size 22 pixels.

xsl:template match="DOJ"

If the template is matched with DOJ element, the value of the DOJ element is displayed in blue color.



The xsl:apply-templates element 4-4

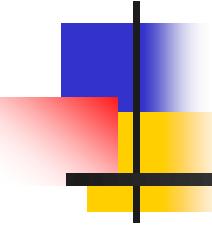
Output

Name: **David Blake**

Date of Join: 18/11/1973

Name: **Susan Jones**

Date of Join: 03/05/1953



The select attribute 1-4

- It can be used to process nodes selected by an expression instead of processing all children.

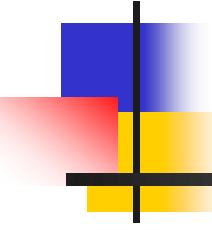
Syntax

```
<xsl:template match = "element">  
    <xsl:apply-templates select ="name of the element"/>  
</xsl:template>
```

where,

select

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.



The select attribute 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="book_stylesheet.xsl"?>
3  <Catalog>
4      <Book>
5          <Title>XML By Example</Title>
6          <Author>David Blake</Author>
7          <Price>$20.90</Price>
8          <Year>1990</Year>
9      </Book>
10
11     <Book>
12         <Title>XML Bible 1.1</Title>
13         <Author>David Troff</Author>
14         <Price>$53</Price>
15         <Year>2004</Year>
16     </Book>
17
18     <Book>
19         <Title>XML Cookbook</Title>
20         <Author>Susan Jones</Author>
21         <Price>$11.10</Price>
22         <Year>1995</Year>
23     </Book>
24 </Catalog>
```

The select attribute 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="/">
4      <html>
5        <body>
6          <h1>Popular XML Books</h1>
7          <xsl:apply-templates/>
8        </body>
9      </html>
10     </xsl:template>
11     <xsl:template match="Book">
12       <p>
13         <xsl:apply-templates select="Title"/>
14         <xsl:apply-templates select="Author"/>
15       </p>
16     </xsl:template>
17     <xsl:template match="Title">
18       Title:
19         <span style="color:green;font-size:20pt">
20           <xsl:value-of select="."/>
21         </span>
22         <BR/>
23     </xsl:template>
24     <xsl:template match="Author">
25       Author:
26         <span style="color:red;font-size:15pt">
27           <xsl:value-of select="."/>
28         </span>
29         <br/>
30     </xsl:template>
31   </xsl:stylesheet>
```

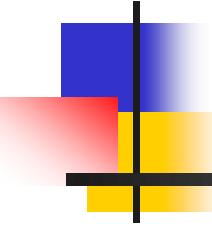
where,

`select="Title"`

Applies the template on Title element.

`select="Author"`

Applies the template on Author element.



The select attribute 4-4

Output

Popular XML Books

Title: **XML By Example**

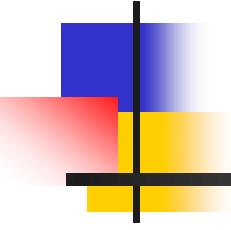
Author: **David Blake**

Title: **XML Bible 1.1**

Author: **David Troff**

Title: **XML Cookbook**

Author: **Susan Jones**



The xsl:value-of element 1-4

- It is used to write or display text string representing the value of the element specified by the select attribute.

Syntax

```
<xsl:value-of  
    select="expression"  
    disable-output-escaping="yes" | "no"  
/>
```

where,

select

A mandatory attribute that assigns the node (by name) to the element.

disable-output-escaping

Specifies how special characters should appear in the output string.

yes

Indicates that special characters should be displayed as is (for example, a < or >).

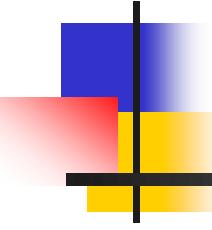
no

Indicates that special characters should not be displayed as is (for example, a > is displayed as >);.

The xsl:value-of element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="person_stylesheet.xsl"?>
3  <House>
4    <Person>
5      <FirstName age="20">David</FirstName>
6      <LastName>Blake</LastName>
7    </Person>
8    <Person>
9      <FirstName age="34">Susan</FirstName>
10     <LastName>Jones</LastName>
11   </Person>
12   <Person>
13     <FirstName>Martin</FirstName>
14     <LastName>King</LastName>
15   </Person>
16   <Person>
17     <FirstName>Justin</FirstName>
18     <LastName>Nora</LastName>
19   </Person>
20 </House>
```



The xsl:value-of element 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="Person">
4      <p>
5        <xsl:value-of select="FirstName"/>
6        ,
7        <xsl:value-of select="LastName"/>
8      </p>
9    </xsl:template>
10   </xsl:stylesheet>
```

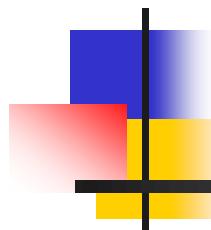
where,

xsl:value-of select="FirstName"

Display the value of the element FirstName.

xsl:value-of select="LastName"

Display the value of the element LastName.



The `xsl:value-of` element 4-4

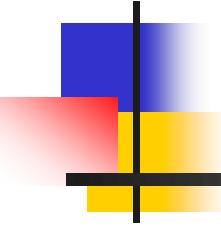
Output

David , Blake

Susan , Jones

Martin , King

Justin , Nora



The xsl:for-each element 1-4

- It can be used to iterate through the XML elements of a specified node set.

Syntax

```
<xsl:for-each select="expression">  
</xsl:for-each>
```

where,

select="expression"

The expression is evaluated on the current context to determine the set of nodes to iterate over.

The xsl:for-each element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="APTEmployees_stylesheet.xsl"?>
3  <Employees>
4   <Employee>
5     <Name>John Thorp</Name>
6     <Department>Marketing</Department>
7     <Language>EN</Language>
8     <Salary>$1000</Salary>
9   </Employee>
10  <Employee>
11    <Name>David Blake</Name>
12    <Department>Admin</Department>
13    <Language>GR</Language>
14    <Salary>$1200</Salary>
15  </Employee>
16  <Employee>
17    <Name>Ben Johns</Name>
18    <Department>Physical Education</Department>
19    <Language>TR</Language>
20    <Salary>$800</Salary>
21  </Employee>
22  <Employee>
23    <Name>Susan Lopez</Name>
24    <Department>External Affairs</Department>
25    <Language>EN</Language>
26    <Salary>$2800</Salary>
27  </Employee>
28 </Employees>
```

The xsl:for-each element 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3      <xsl:output method="html"/>
4      <!-- Template for "employees" elements --&gt;
5      &lt;xsl:template match="Employees"&gt;
6          &lt;h1&gt;Employee Information&lt;/h1&gt;
7          &lt;!-- Table Header Creation --&gt;
8          &lt;table border="1"&gt;
9              &lt;tr&gt;
10                 &lt;th&gt;Department&lt;/th&gt;
11                 &lt;th&gt;Name&lt;/th&gt;
12                 &lt;th&gt;Salary&lt;/th&gt;
13                 &lt;th&gt;Language&lt;/th&gt;
14             &lt;/tr&gt;
15             &lt;xsl:for-each select="Employee"&gt;
16                 &lt;tr&gt;
17                     &lt;td&gt;
18                         &lt;xsl:value-of select="Department"/&gt;
19                     &lt;/td&gt;
20                     &lt;td&gt;
21                         &lt;xsl:value-of select="Name"/&gt;
22                     &lt;/td&gt;
23                     &lt;td&gt;
24                         &lt;xsl:value-of select="Salary"/&gt;
25                     &lt;/td&gt;
26                     &lt;td&gt;
27                         &lt;xsl:value-of select="Language"/&gt;
28                     &lt;/td&gt;
29                 &lt;/tr&gt;
30             &lt;/xsl:for-each&gt;
31             &lt;!-- End of Table --&gt;
32         &lt;/table&gt;
33     &lt;/xsl:template&gt;
34 &lt;/xsl:stylesheet&gt;</pre>
```

where,

xsl:for-each select="Employee"
Iterates through the Employee node and
applies a template.

xsl:value-of select="Department"
Displays the value of Department element.

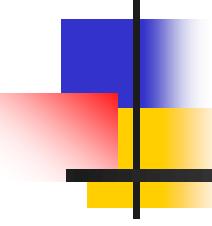
xsl:value-of select="Name"
Displays the value of Name element.

xsl:value-of select="Salary"
Displays the value of Salary element.

xsl:value-of select="Language"
Displays the value of Language element.

</xsl:for-each>

End of for-each loop.

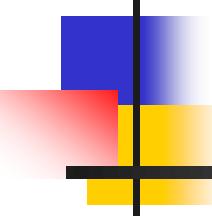


The xsl:for-each element 4-4

Output

Employee Information

Department	Name	Salary	Language
Marketing	John Thorp	\$1000	EN
Admin	David Blake	\$1200	GR
Physical Education	Ben Johns	\$800	TR
External Affairs	Susan Lopez	\$2800	EN



The xsl:text element 1-4

- It is used to add literal text to the output.

Syntax

```
<xsl:text  
    disable-output-escaping="yes"|"no">  
</xsl:text>
```

where,

disable-output-escaping

Turns on or off the ability to escape special characters.

yes

If the value is yes, a > will appear as a >.

no

If the value is no, a > will appear as a > in the text.

The xsl:text element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Orders_stylesheet.xsl"?>
3  <Orders>
4  <Item>
5      <Name>Dell Laptop</Name>
6      <Price>$ 45000</Price>
7      <ShippingDate>10-Mar-07</ShippingDate>
8  </Item>
9  <Item>
10     <Name>Mouse</Name>
11     <Price>$ 450</Price>
12     <ShippingDate>10-Mar-07</ShippingDate>
13 </Item>
14 <Item>
15     <Name>Dell Keyboard</Name>
16     <Price>$ 150</Price>
17     <ShippingDate>10-Mar-07</ShippingDate>
18 </Item>
19 </Orders>
```

The xsl:text element 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3      <xsl:output method="html"/>
4      <xsl:template match="/">
5          <html>
6              <body>
7                  <xsl:text>
8                      Following are the items which are shipped on 10th March
9                  </xsl:text>
10                 <br/>
11                 <xsl:for-each select="Orders/Item">
12                     <xsl:value-of select="Name"/>
13                     <xsl:text>, </xsl:text>
14                 </xsl:for-each>
15                 <xsl:text>!!!!</xsl:text>
16             </body>
17         </html>
18     </xsl:template>
19 </xsl:stylesheet>
```

where,

xsl:for-each select="Orders/Item"

Iterates through the Item element.

xsl:value-of select="Name"

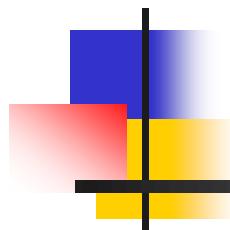
Displays the value of Name element.

<xsl:text>, </xsl:text>

Inserts a comma (,) after each name value .

<xsl:text>!!!!</xsl:text>

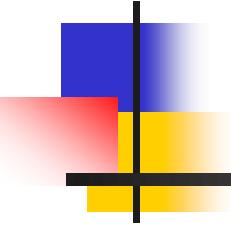
Inserts four exclamation marks (!) at the end of the output.



The xsl:text element 4-4

Output

Following are the items which are shipped on 10th March
Dell Laptop,Mouse,Dell Keyboard,!!!



The xsl:number element 1-4

- It can be used to determine the sequence number for the current node.

Syntax

```
<xsl:number  
    count="pattern"  
    format="{ string }"  
    value="expression"  
>  
</xsl:number>
```

where,

count="pattern"

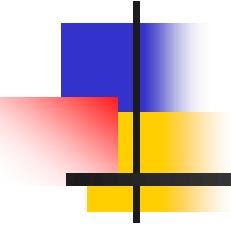
Indicates what nodes are to be counted. Only nodes that match the pattern are counted.

format="{ string }"

Sequence of tokens that specifies the format to be used for each number in the list.

value="expression"

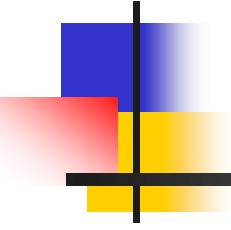
Specifies the expression to be converted to a number and output to the result tree.



The xsl:number element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="item_stylesheet.xsl" ?>
3  <Items>
4      <Item>Water Bottle</Item>
5      <Item>Chocolates</Item>
6      <Item>Computer Book</Item>
7      <Item>Mobile Phone</Item>
8      <Item>Personal Computer</Item>
9  </Items>
```



The xsl:number element 3-4

Style Sheet

```
1  <?xml version="1.0"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4  <xsl:template match="Items">
5      <h3> Items numbered in Western and then upper-case Roman numbering</h3>
6      <xsl:for-each select="Item">
7          <xsl:number value="position()" format="1."/>
8          <xsl:value-of select="."/>
9          ,
10         <xsl:number value="position()" format="I."/>
11         <xsl:value-of select="."/>
12         <br/>
13     </xsl:for-each>
14   </xsl:template>
15 </xsl:stylesheet>
```

where,

position()

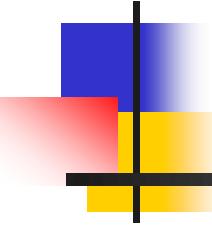
The current node's position in the source document.

format="1."

User-provided number starts with 1.

format="I."

User-provided roman number starts with I.



The xsl:number element 4-4

Output

Items numbered in Western and then upper-case Roman numbering

1. Water Bottle , I. Water Bottle
2. Chocolates , II. Chocolates
3. Computer Book , III. Computer Book
4. Mobile Phone , IV. Mobile Phone
5. Personal Computer , V. Personal Computer

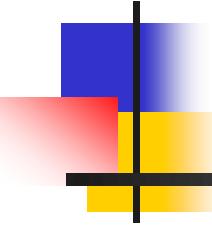
where,

1. Water Bottle, I. Water Bottle

The first item is numbered with number 1 and roman number I.

4. Mobile Phone, IV. Mobile Phone

The fourth item is numbered with number 4 and roman number IV.



The xsl:if element 1-4

- Evaluates a conditional expression against the content of the XML file.

Syntax

```
<xsl:if  
    test="expression"  
>  
</xsl:if>
```

where,

test=expression

The condition in the source data to test with either a true or false.

The xsl:if element 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Librarybooks_stylesheet.xsl"?>
3  <Catalog>
4   <Book>
5     <Title>XML By Example</Title>
6     <Author>David Blake</Author>
7     <Price>20.90</Price>
8     <Year>1990</Year>
9   </Book>
10  <Book>
11    <Title>XML Bible 1.1</Title>
12    <Author>David Troff</Author>
13    <Price>53</Price>
14    <Year>2004</Year>
15  </Book>
16  <Book>
17    <Title>XML Cookbook</Title>
18    <Author>Susan Jones</Author>
19    <Price>11.10</Price>
20    <Year>1995</Year>
21  </Book>
22  <Book>
23    <Title>XML Complete Reference </Title>
24    <Author>Andrew Nel</Author>
25    <Price>193</Price>
26    <Year>2001</Year>
27  </Book>
28 </Catalog>
```

The xsl:if element 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0"
3      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4      <xsl:template match="/">
5          <html>
6              <body>
7                  <h2>Library Books</h2>
8                  <table border="1">
9                      <tr bgcolor = "lime">
10                         <th>Book Title</th>
11                         <th>Author</th>
12                         <th>Year</th>
13                     </tr>
14                     <xsl:for-each select="Catalog/Book">
15                         <xsl:if test="Price &gt; 50">
16                             <tr>
17                                 <td><xsl:value-of select="Title"/></td>
18                                 <td><xsl:value-of select="Author"/></td>
19                                 <td><xsl:value-of select="Year"/></td>
20                             </tr>
21                         </xsl:if>
22                     </xsl:for-each>
23                 </table>
24             </body>
25         </html>
26     </xsl:template>
27 </xsl:stylesheet>
```

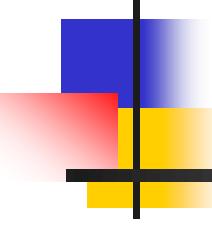
where,

```
<xsl:for-each select="Catalog/Book">
```

Iterates through the Book node.

```
<xsl:if test="Price &gt; 50">
```

Checks if the price of the book is greater than 50. If true, Author, Title and Year are displayed.

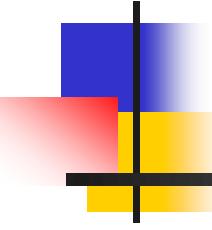


The xsl:if element 4-4

Output

Library Books

Book Title	Author	Year
XML Bible 1.1	David Troff	2004
XML Complete Reference	Andrew Nel	2001



The xsl:choose element

- It is used in conjunction with `xsl:when` and `xsl:otherwise` to express multiple conditional tests.

Syntax

```
<xsl:choose>
  <xsl:when test="expression">
    template body
  </xsl:when>
  ...
  <xsl:otherwise>
    template body
  </xsl:otherwise>
</xsl:choose>
```

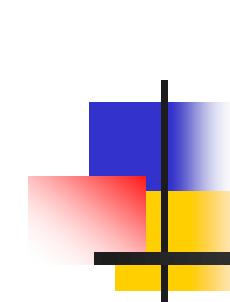
where,

`xsl:when test="expression"`

The `xsl:when` element is examined in the order of occurrence. If the test expression is true, the code contained in that element is executed.

`xsl:otherwise`

If all the test conditions in any `xsl:when` element are false, then the `xsl:otherwise` element is automatically selected and the code associated with that element is executed.



The xsl:choose element

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Publisherbooks_stylesheet.xsl"?>
3  <Publisher>
4    <Book>
5      <Title>XML By Example</Title>
6      <Author>David Blake</Author>
7      <Price>20.90</Price>
8      <Year>1990</Year>
9    </Book>
10   <Book>
11     <Title>XML Cookbook</Title>
12     <Author>Susan Jones</Author>
13     <Price>11.10</Price>
14     <Year>1995</Year>
15   </Book>
16   <Book>
17     <Title>XML Complete Reference </Title>
18     <Author>Andrew Nel</Author>
19     <Price>193</Price>
20     <Year>2001</Year>
21   </Book>
22 </Publisher>
```

The xsl:choose element 3-5

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3    <xsl:template match="/">
4      <html>
5        <body>
6          <h2>Publisher Books</h2>
7          <table border="1">
8            <tr bgcolor="pink">
9              <th>Title</th>
10             <th>Author</th>
11             <th>Price</th>
12             <th>Year</th>
13           </tr>
14           <xsl:for-each select="Publisher/Book">
15             <tr>
16               <td>
17                 <xsl:value-of select="Title"/>
18               </td>
19               <xsl:choose>
20                 <xsl:when test="Price > 100">
21                   <td bgcolor="magenta">
22                     <xsl:value-of select="Author"/>
23                   </td>
24                   <td bgcolor="magenta">
25                     <xsl:value-of select="Price"/>
26                   </td>
27                   <td bgcolor="magenta">
28                     <xsl:value-of select="Year"/>
29                   </td>
30                 </xsl:when>
```

The xsl:choose element

Style Sheet

```
31      <xsl:otherwise>
32          <td>
33              <xsl:value-of select="Author"/>
34          </td>
35          <td>
36              <xsl:value-of select="Price"/>
37          </td>
38          <td>
39              <xsl:value-of select="Price"/>
40          </td>
41          <td>
42              <xsl:value-of select="Year"/>
43          </td>
44      </xsl:otherwise>
45  </xsl:choose>
46  </tr>
47 </xsl:for-each>
48 </table>
49 </body>
50 </html>
51 </xsl:template>
52 </xsl:stylesheet>
```

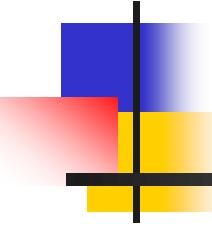
where,

xsl:when test="Price > 100"

Checks whether the price of the book is greater than 100. If true, the details like Author, Price and Year are displayed with magenta as the background color.

xsl:otherwise

If all the conditions are false, this block is executed. Here, all other book details are printed in normal background color.

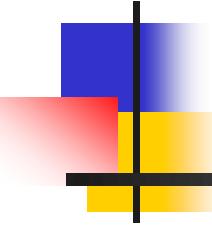


The xsl:choose element

Output

Publisher Books

Title	Author	Price	Year
XML By Example	David Blake	20.90	1990
XML Cookbook	Susan Jones	11.10	1995
XML Complete Reference	Andrew Nel	193	2001



Sorting in XSLT 1-4

- It can be used to sort a group of similar elements.

Syntax

```
<xsl:sort  
    case-order="upper-first" | "lower-first"  
    data-type="number" "qname" | "text"  
    order="ascending" | "descending"  
    select="expression"  
>  
</xsl:sort>
```

where,

case-order

Indicates whether the sort will have upper or lowercase letters listed first in the sort output. The default option is to list uppercase first.

data-type: Specifies the data type of the strings.

Number: Sort key is converted to a number.

Qname: Sort is based upon a user-defined data type.

Text: Specifies that the sort keys should be sorted alphabetically.

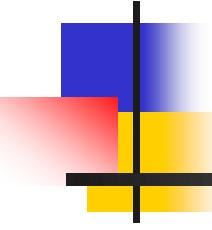
Order: The sort order for the strings. The default value is "ascending".

Select: Expression that defines the key upon which the sort will be based. The expression is evaluated and converted to a string that is used as the sort key.

Sorting in XSLT 2-4

Code Snippet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?xml-stylesheet type="text/xsl" href="Products_stylesheet.xsl"?>
3  <Products>
4      <Item>
5          <ItemCode>CD01</ItemCode>
6          <ItemName>Music CD</ItemName>
7          <UnitPrice>9.90</UnitPrice>
8      </Item>
9      <Item>
10         <ItemCode>PN01</ItemCode>
11         <ItemName>Parker Pens</ItemName>
12         <UnitPrice>12.50</UnitPrice>
13     </Item>
14     <Item>
15         <ItemCode>CK01</ItemCode>
16         <ItemName>Coca Cola</ItemName>
17         <UnitPrice>2.20</UnitPrice>
18     </Item>
19     <Item>
20         <ItemCode>BK01</ItemCode>
21         <ItemName>Computer Books</ItemName>
22         <UnitPrice>8.76</UnitPrice>
23     </Item>
24 </Products>
```



Sorting in XSLT 3-4

Style Sheet

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
3    <xsl:template match="/">
4      <html>
5        <body>
6          <xsl:for-each select="Products/Item">
7            <xsl:sort data-type="number" select="UnitPrice" order="descending"/>
8            <xsl:value-of select="ItemName"/>
9            <xsl:text>-</xsl:text>
10           <xsl:value-of select="UnitPrice"/>
11           <br/>
12         </xsl:for-each>
13       </body>
14     </html>
15   </xsl:template>
16 </xsl:stylesheet>
```

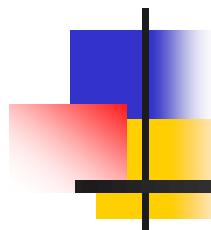
where,

select="UnitPrice"

Sort is based on UnitPrice element.

order="descending"

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



Sorting in XSLT 4-4

Output

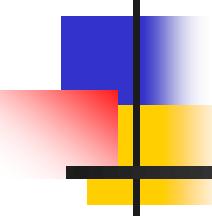
Parker Pens-12.50

Music CD-9.90

Computer Books-8.76

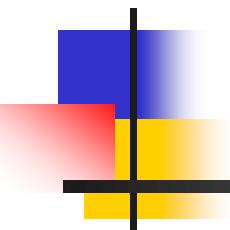
Coca Cola-2.20

For Aptech Center Use Only



Summary

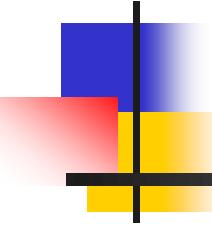
- **Introduction to XSL**
 - XML provides the ability to format document content.
 - XSL provides the ability to define how the formatted XML content is presented.
 - An XSL Transformation applies rules to a source tree read from an XML document to transform it into an output tree written out as an XML document.
- **Working with XSL**
 - An XSL template rule is represented as an `xsl:template` element.
 - You can process multiple elements in two ways: using the `xsl:apply-templates` element and the `xsl:for-each` element.
 - The `xsl:stylesheet` element allows you to include a stylesheet directly in the document it applies to.
 - The `xsl:if` element produces output only if its `test` attribute is true.



Module 7

More on XSLT

For Aptech Center Use Only

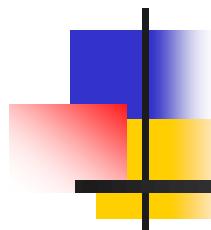


Module Overview

In this module, you will learn about:

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

For Aptech Center Use Only

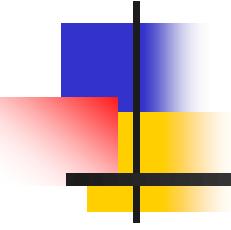


Lesson 1 - XPath

In this first lesson, **XPath**, you will learn to:

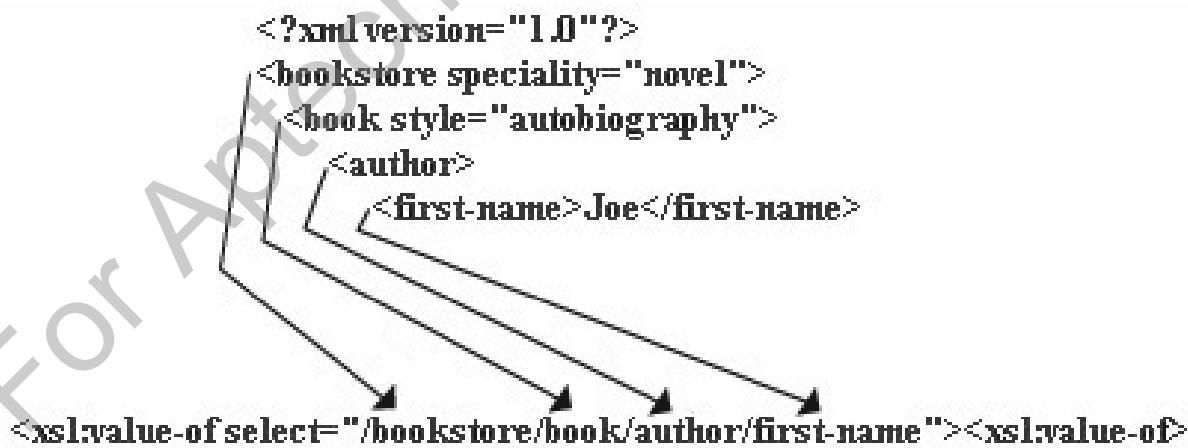
- Define and describe XPath.
- Identify nodes according to XPath.
- List operators used with XPath.
- Describe the types of matching

For Aptech Centres Only



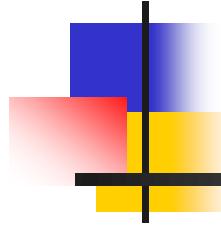
XPath 1-2

- XPath can be considered as a query language like SQL.
- Extracts information from an XML document.
- Language for retrieving information from a XML document.
- Used to navigate through elements and attributes in an XML document.
- Allows identifying parts of an XML document.
- Provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery.



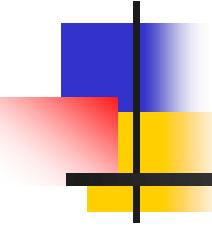
```
<?xml version="1.0"?>
<bookstore speciality="novel">
  <book style="autobiography">
    <author>
      <first-name>Joe</first-name>
    </author>
  </book>
</bookstore>
```

```
<xsl:value-of select="/bookstore/book/author/first-name"><xsl:value-of>
```



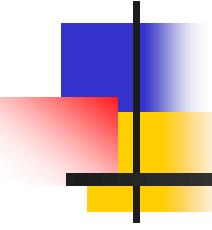
XPath 2-2

- XSLT
 - Language for transforming XML documents into XML, HTML or text.
- XQuery
 - Builds on XPath.
 - Language for extracting information from XML documents.



Benefits of XPath

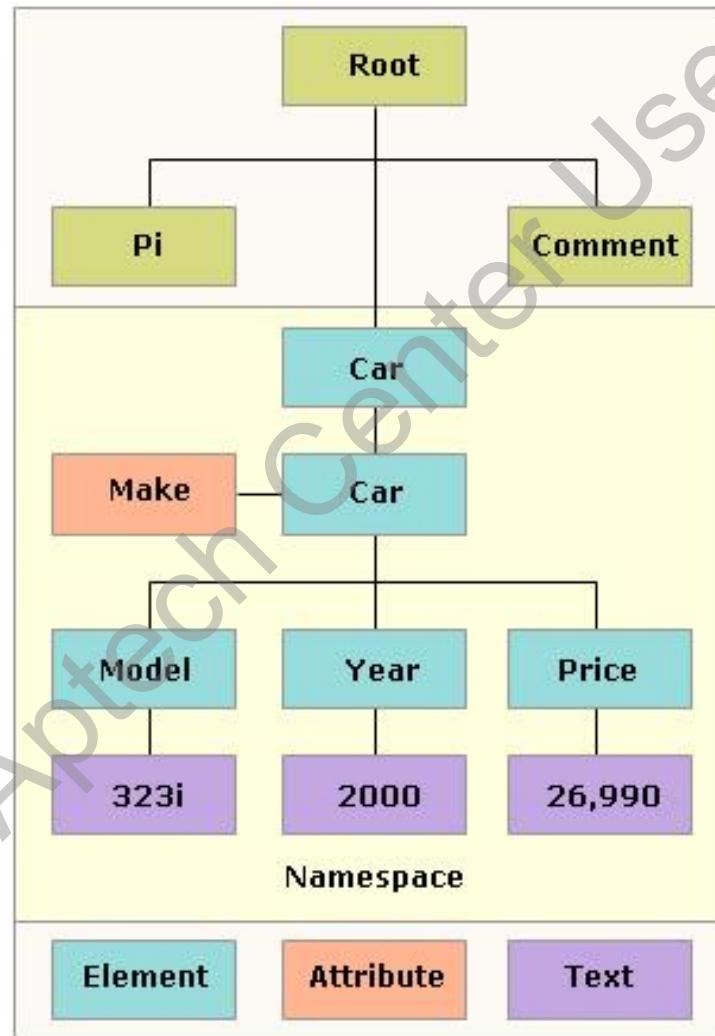
- Provides single syntax that you can use for:
 - Queries
 - Addressing
 - Patterns
- Concise, simple, and powerful.
- Benefits:
 - Syntax is simple for the simple and common cases.
 - Any path that can occur in an XML document.
 - Any set of conditions for the nodes in the path can be specified.
 - Any node in an XML document can be uniquely identified.

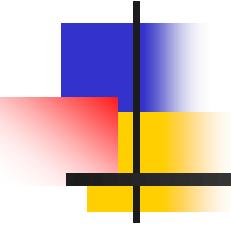


XML Document in XPath 1-4

- An XML document is viewed as a tree.
- Each part of the document is represented as a node.
- Nodes
 - Root
 - A single `root` node that contains all other nodes in the tree.
 - Element
 - Every element in a document has a corresponding `element` node that appears in the tree under the `root` node.
 - Within an `element` node appear all of the other types of nodes that correspond to the element's content.
 - Element nodes may have a unique identifier associated with them that is used to reference the node with XPath.

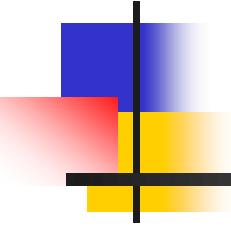
XML Document in XPath 2-4





XML Document in XPath 3-4

- Attribute
 - Each `element` node has an associated set of attribute nodes.
 - Element is the parent of each of these attribute nodes.
 - An attribute node is not a child of its parent element.
- Text
 - Character data is grouped into text nodes.
 - Characters inside comments, processing instructions and attribute values do not produce text nodes.
 - The `text` node has a parent node and it may be the `child` node.
- Comment
 - There is a `comment` node for every comment, except within the document type declaration (DTD).
 - The `comment` node has a parent node and it may be the `child` node too.



XML Document in XPath 4-4

- Processing instruction

- Processing instruction node exists for every processing instruction except for any processing instruction within the document type declaration.
- The processing instruction node has a parent node and it may be the child node too.

- Namespace

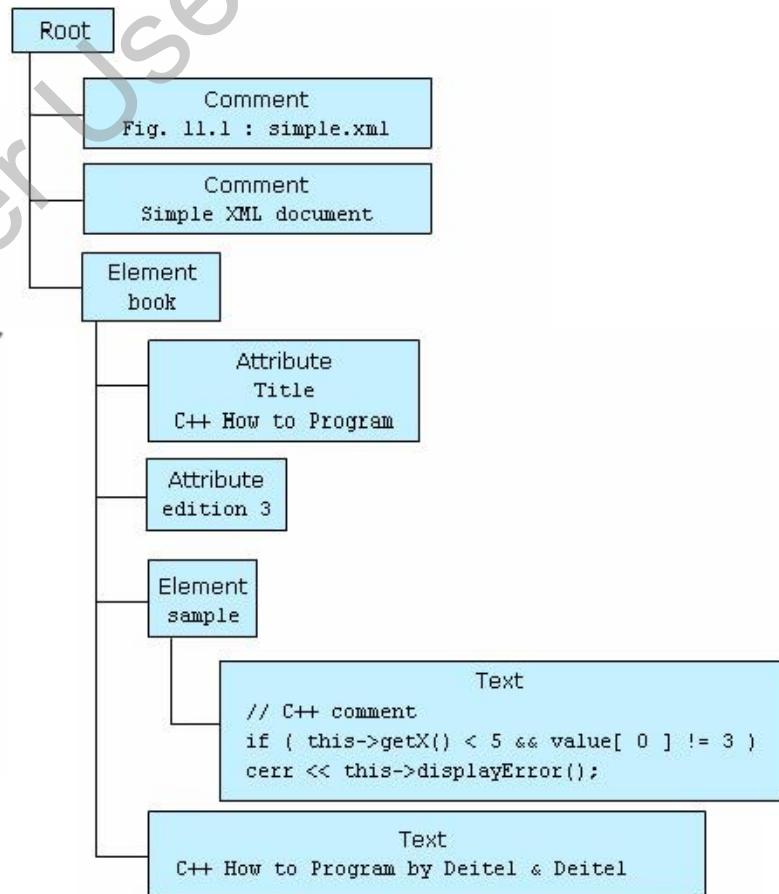
- Each element has an associated set of namespace nodes.
- Provides descriptive information about their parent node.

XPath Representation

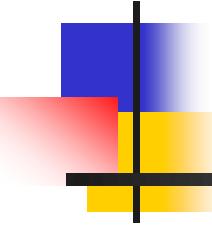
- An XPath query operates on a well-formed XML document after it has been parsed into a tree structure.

```
<?xml version = "1.0"?>  
  
<!-- Fig. 11.1 : simple.xml -->  
<!-- Simple XML document -->  
  
<book title = "C++ How to Program" edition = "3">  
  <sample>  
    <![CDATA[  
      // C++ comment  
      if ( this->getX() < 5 && value[ 0 ] != 3 )  
        cerr << this->displayError();  
    ]]>  
  </sample>  
  
  C++ How to Program by Deitel & Deitel  
  </book>
```

Original XML Document



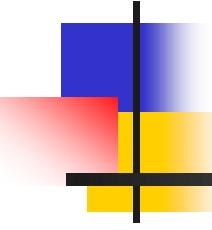
XPath Representation



Operators in XPath

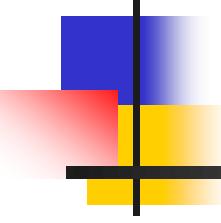
- An XPath expression returns a node set, a boolean, a string, or a number.
- XPath provides basic floating point arithmetic operators and some comparison and boolean operators.

Operator	Description
/	Child operator; selects immediate children of the left-side collection.
//	Recursive descent; searches for the specified element at any depth
.	Indicates the current context
..	The parent of the current context node
*	Wildcard; selects all elements regardless of the element name
@	Attribute; prefix for an attribute name
:	Namespace separator; separates the namespace prefix from the element or attribute name



Examples of XPath Operators

Expression	Refers to
Author/FirstName	All <FirstName> elements within an <Author> element of the current context node.
BookStore//Title	All <Title> elements one or more levels deep in the <BookStore> element.
BookStore/*>Title	All <Title> elements that are grandchildren of <BookStore> elements.
BookStore//Book/Excerpt//Work	All <Work> elements anywhere inside <Excerpt> children of <Book> elements, anywhere inside the <BookStore> element.
.//Title	All <Title> elements one or more levels deep in the current context.



Types of Matching 1-3

- XPath can create of the patterns.
- The match attribute of the `xsl:template` element supports a complex syntax.
- 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 a superset of the syntax.
- Allows to express exactly which nodes to selected and which nodes not to be selected.

Types of Matching 2-3

- Matching by name

- The source element is identified by its name, using the `match` attribute.
- The value given to the `match` attribute is called the pattern.

Code Snippet

```
<xsl:template match = "Greeting">
```

→ Matches all greeting elements in the source document.

- Matching by ancestry

- As in CSS, a match can be made using the element's ancestry.

Code Snippet

```
<xsl:template match = "P//EM">
```

→ Matches any 'EM' element that has 'P' as an ancestor.

- Matching the element names

- The most basic pattern contains a single element name which matches all elements with that name.

Code Snippet

```
<xsl:template match="Product">  
  <xsl:value-of select="Product_ID"/>  
</xsl:template>
```

→ Matches Product elements and marks their Product_ID children bold.

Types of Matching 3-3

- Matching the root

- Enables all descendant nodes to inherit the properties on the root of document.
- Uses a single forward slash to represent the root.

Code Snippet

```
<xsl:template match = "/">
```

Selects the root pattern.

- Matching by attribute

- As in CSS, a match can be made using the element's ancestry.

Syntax

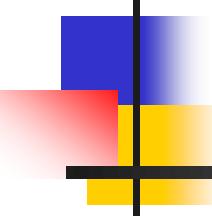
```
<xsl:template match = " element name['attribute'  
(attribute-name)=attribute-value]">
```

Uses square brackets to hold the attribute name and value.

Code Snippet

```
<xsl:template match="Product">  
  <xsl:apply-templates select="@Units"/>  
</xsl:template>
```

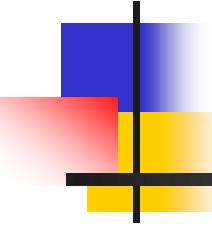
Applies the templates to the non-existent Units attributes of Product elements.



Lesson 2 – XPath Expressions and Functions

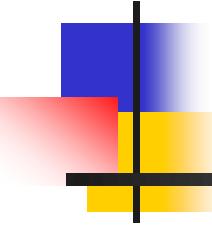
In this second lesson, **XPath Expressions and Functions**, you will learn to:

- State and explain the various XPath expressions and functions.
- List the node set functions.
- List the boolean functions.
- State the numeric functions.
- Describe the string functions.
- Explain what result tree fragments are.



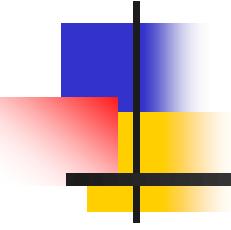
XPath Expressions 1-2

- 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.
- Are like statements in a functional programming language.
- Evaluates to a single value.



XPath Expressions 2-2

- Node-set
 - An unordered group of nodes from the input document that match an expression's criteria.
- Boolean
 - XSLT allows any kind of data to be transformed into a boolean.
 - Often done implicitly when a string or a number or a node-set is used where a boolean is expected.
- Number
 - Numeric values useful for counting nodes and performing simple arithmetic.
 - Numbers like 43 or -7000 that look like integers are stored as doubles.
 - Non-number values, such as strings and booleans, are converted to numbers automatically as necessary.
- String
 - A sequence of zero or more Unicode characters.
 - Other data types can be converted to strings using the `string()` function.



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.
- 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`.

Node-Set Functions 1-4

- Take a node-set argument.
- Return a node-set or information about a particular node within a node-set.
- Are `name()`, `local-name()`, `namespace-uri()` and `root()`.

Syntax

`name()`

`fn:name()`

`fn:name(nodeset)`

`local-name()`

`fn:local-name()`

`fn:local-name(nodeset)`

`namespace-uri()`

`fn:namespace-uri()`

`fn:namespace-uri(nodeset)`

`root()`

`fn:root()`

`fn:root(node)`

where,

`name()` : Returns name of current node or the first node in specified node set.

`local-name()` : Returns name of current node or the first node in specified node set without namespace prefix.

`namespace-uri()` : Returns namespace URI of current node or first node in specified node set.

`root()` : Returns root of tree to which the current node or specified node belongs. This will usually be a document node.

Node-Set Functions 2-4

Code and Schema

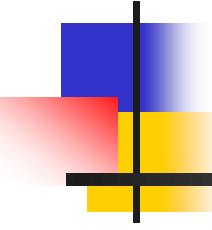
```
<!--Book.xml -->
1  <?xmlstylesheet type="text/xsl" href="Sample.xsl"?>
2  <Catalog xmlns="http://www.BookCatalog.com"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemalocation="http://www.BookCatalog.com BookSchema.xsd">
5    <Book>
6      <Author>Gambardella, Matthew</Author>
7      <Title>XML Developer's Guide</Title>
8      <Genre>Computer</Genre>
9      <Price>44.95</Price>
10     <Publish>2000-10-01</Publish>
11     <Description>An in-depth look at creating applications
12      with XML.</Description>
13    </Book>
14  </Catalog>

<!--BookSchema.xsd -->
1  <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema"
2   targetNamespace="http://www.BookCatalog.com"
3   xmlns="http://www.BookCatalog.com"
4   elementFormDefault="qualified">
5
6  <!--definition for simple elements-->
7
8  <xss:element name="Author" type="xss:string"/>
9  <xss:element name="Title" type="xss:string"/>
10 <xss:element name="Genre" type="xss:string"/>
11 <xss:element name="Price" type="xss:float"/>
12 <xss:element name="Publish" type="xss:string"/>
13 <xss:element name="Description" type="xss:string"/>
14
15 <!--definition for complex elements-->
16 <xss:element name="Book">
17   <xss:complexType>
18     <xss:sequence>
19       <xss:element ref="Author"/>
20       <xss:element ref="Title"/>
21       <xss:element ref="Genre"/>
22       <xss:element ref="Price"/>
23       <xss:element ref="Publish"/>
24       <xss:element ref="Description"/>
25     </xss:sequence>
26   </xss:complexType>
27 </xss:element>
28
29 <xss:element name="Catalog">
30   <xss:complexType>
31     <xss:sequence>
32       <xss:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
33     </xss:sequence>
34   </xss:complexType>
35 </xss:element>
36 </xss:schema>
```

Node-Set Functions 3-4

Stylesheet

```
<!--Sample.xsl -->
1 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
2   <xsl:output method="html"/>
3   <xsl:template match="/">
4     <html>
5       <body>
6         <h3>Node-set Function</h3>
7
8         <table width="100%" border="1">
9           <tr>
10             <td width="25%"><b>namespace-uri()</b></td>
11             <td width="25%"><b>name()</b></td>
12             <td width="25%"><b>local-name</b></td>
13             <td width="25%"><b>text()</b></td>
14           </tr>
15           <xsl:apply-templates />
16         </table>
17       </body>
18     </html>
19   </xsl:template>
20
21   <xsl:template match="*">
22     <tr>
23       <td>
24         <xsl:value-of select="namespace-uri()"/>
25       </td>
26       <td>
27         <xsl:value-of select="name()"/>
28       </td>
29       <td>
30         <xsl:value-of select="local-name()"/>
31       </td>
32       <td>
33         <xsl:value-of select="text()"/>
34       </td>
35     </tr>
36     <xsl:apply-templates select="/" />
37   </xsl:template>
38 </xsl:stylesheet>
```

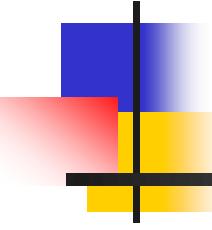


Node-Set Functions 4-4

Formatted Output

Node-set Function

namespace-uri()	name()	local-name	text()
http://www.BookCatalog.com	Catalog	Catalog	
http://www.BookCatalog.com	Book	Book	
http://www.BookCatalog.com	Author	Author	Gambardella, Matthew
http://www.BookCatalog.com	Title	Title	XML Developer's Guide
http://www.BookCatalog.com	Genre	Genre	Computer
http://www.BookCatalog.com	Price	Price	44.95
http://www.BookCatalog.com	Publish	Publish	2000-10-01
http://www.BookCatalog.com	Description	Description	An in-depth look at creating applications with XML.



Boolean Functions 1-5

- **boolean(arg)**
 - Returns a boolean value for a number, string, or node-set.

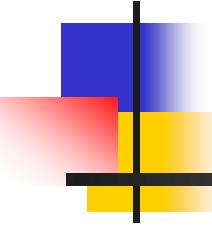
Syntax

```
fn:boolean (arg)
```

Code Snippet

```
<ul>
    <li><b>boolean(0)</b> = <xsl:value-of select="boolean(0)" /></li>
    <li><b>boolean(1)</b> = <xsl:value-of select="boolean(1)" /></li>
    <li><b>boolean(-100)</b> = <xsl:value-of select="boolean(-100)" /></li>
    <li><b>boolean('hello')</b> = <xsl:value-of select="boolean('hello')" /></li>
    <li><b>boolean('')</b> = <xsl:value-of select="boolean('')" /></li>
    <li><b>boolean{//book}</b> = <xsl:value-of select="boolean{//book}" /></li>
</ul>
```

```
boolean(0) = false
boolean(1) = true
boolean(-100) = true
boolean('hello') = true
boolean('') = false
boolean{//book} = false
```

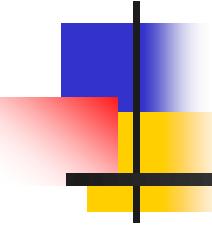


Boolean Functions 2-5

- **not(arg)**
 - The sense of an operation can be reversed by using the `not()` function.

Syntax

```
fn:not(arg)
```



Boolean Functions 3-5

- **not(arg)**
 - The sense of an operation can be reversed by using the `not()` function.

Code Snippet

```
<xsl:template match="PRODUCT[not(position()=1)]">
  <xsl:value-of select=".">
```



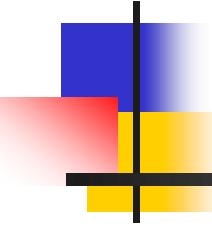
The template rule selects all product elements that are not the first child of their parents.

Code Snippet

```
<xsl:template match="PRODUCT[position() !=1]">
  <xsl:value-of select=".">
```



The same template rule could be written using the not equal operator `!=` instead.



Boolean Functions 4-5

- **true()**
 - Returns the boolean value true.

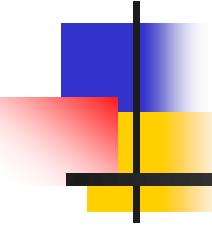
Syntax

```
fn:true()
```

Code Snippet

```
<xsl:value-of select="true()"/>
```

true



Boolean Functions 5-5

- `false()`
 - Returns the boolean value `true`.

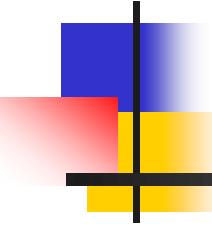
Syntax

```
fn:false()
```

Code Snippet

```
<xsl:value-of select="false() or false()" />
<xsl:value-of select="true() and false()" />
<xsl:value-of select="false() and false()" />
```

↓
true
false
false



Numeric Functions 1-5

- Return strings or numbers.
- Can be used with comparison operators in filter patterns such as:
 - number (arg)
 - ceiling (num)
 - floor (num)
 - round (num)

Syntax

number(arg)

fn: number (arg)

ceiling(num)

fn: ceiling (num)

floor(num)

fn: floor (num)

round(num)

fn: round (num)

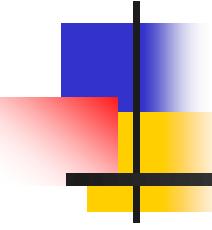
where,

number (arg) : Returns numeric value of argument. Argument could be a boolean, a string, or a node-set.

ceiling (num) : Returns smallest integer greater than the number argument.

floor (num) : Returns largest integer that is not greater than the number argument.

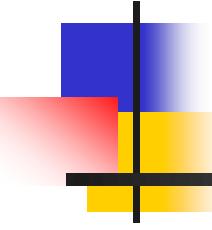
round (num) : Rounds the number argument to the nearest integer.



Numeric Functions 2-5

Stylesheet(1-3)

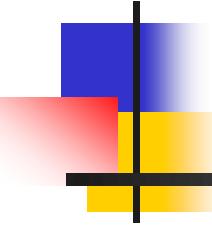
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="Number.xsl"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <h3>Numeric Functions</h3>
        <ul>
          <li>
            <b>number('1548')</b> = <xsl:value-of select="number('1548')"/>
          </li>
          <li>
            <b>number('-1548')</b> = <xsl:value-of select="number('-1548')"/>
          </li>
          <li>
            <b>number('text')</b> = <xsl:value-of select="number('text')"/>
          </li>
          <li>
            <b>number('226.38' div '1')</b> = <xsl:value-of select="number('226.38' div '1')"/>
          </li>
        </ul>
```



Numeric Functions 3-5

Stylesheet(2-3)

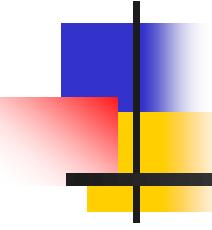
```
<ul>
  <li>
    <b>ceiling(2.5)</b> = <xsl:value-of select="ceiling(2.5)" />
  </li>
  <li>
    <b>ceiling(-2.3)</b> = <xsl:value-of select="ceiling(-2.3)" />
  </li>
  <li>
    <b>ceiling(4)</b> = <xsl:value-of select="ceiling(4)" />
  </li>
</ul>
<ul>
  <li>
    <b>floor(2.5)</b> = <xsl:value-of select="floor(2.5)" />
  </li>
  <li>
    <b>floor(-2.3)</b> = <xsl:value-of select="floor(-2.3)" />
  </li>
  <li>
    <b>floor(4)</b> = <xsl:value-of select="floor(4)" />
  </li>
</ul>
```



Numeric Functions 4-5

Stylesheet(3-3)

```
<ul>
  <li>
    <b>round(3.6)</b> = <xsl:value-of select="round(3.6)" />
  </li>
  <li>
    <b>round(3.4)</b> = <xsl:value-of select="round(3.4)" />
  </li>
  <li>
    <b>round(3.5)</b> = <xsl:value-of select="round(3.5)" />
  </li>
  <li>
    <b>round(-0.6)</b> = <xsl:value-of select="round(-0.6)" />
  </li>
  <li>
    <b>round(-2.5)</b> = <xsl:value-of select="round(-2.5)" />
  </li>
</ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

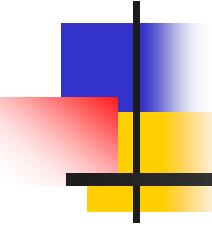


Numeric Functions 5-5

Output

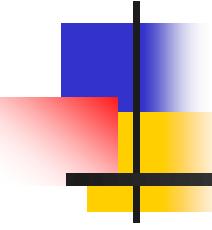
Numeric Functions

- `number('1548')` = 1548
- `number('-1548')` = -1548
- `number('text')` = NaN
- `number('226.38' div '1')` = 226.38
- `ceiling(2.5)` = 3
- `ceiling(-2.3)` = -2
- `ceiling(4)` = 4
- `floor(2.5)` = 2
- `floor(-2.3)` = -3
- `floor(4)` = 4
- `round(3.6)` = 4
- `round(3.4)` = 3
- `round(3.5)` = 4
- `round(-0.6)` = -1



String Functions 1-5

- String functions are used to:
 - Evaluate
 - Format and manipulate string arguments
 - Convert an object to a string
- Different String functions are:
 - `string(arg)`
 - `compare()`
 - `concat()`
 - `substring()`



String Functions 2-5

Syntax

string(arg)

fn: string (arg)

translate()

fn: translate(string, string, string)

concat()

fn: concat(string, string, ...)

substring()

fn: substring(string, start, len)

fn: substring(string, start)

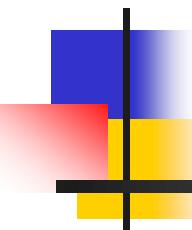
where,

string (arg) : Returns string value of the argument. The argument could be a number, boolean, or node-set.

translate () : Returns first argument string with occurrences of characters in second argument string replaced by character at the corresponding position in third argument string.

concat () : Returns concatenation of the strings.

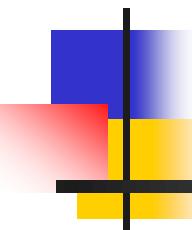
substring () : Returns substring from the start position to specified length.



String Functions 3-5

Code Snippet

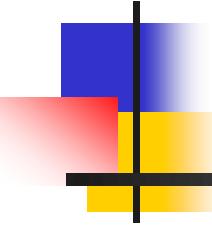
```
1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="BookDetail.xsl"?>
3  <BookStore>
4   <Book>
5     <Title>The Weather Pattern</Title>
6     <Author>Weather Man</Author>
7     <Price>100.00</Price>
8   </Book>
9   <Book>
10    <Title>Weaving Patterns</Title>
11    <Author>Weaver</Author>
12    <Price>150.00</Price>
13  </Book>
14  <Book>
15    <Title>Speech Pattern</Title>
16    <Author>Speaker</Author>
17    <Price>15.00</Price>
18  </Book>
19  <Book>
20    <Title>Writing Style</Title>
21    <Author>Writer</Author>
22    <Price>1500.00</Price>
23  </Book>
24 </BookStore>
```



String Functions 4-5

Stylesheet

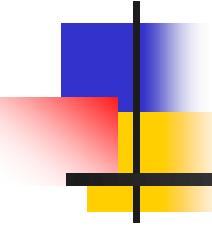
```
1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="BookDetail.xsl"?>
3  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4  <xsl:template match="/">
5  <html>
6  <head>
7  <title>example</title>
8  </head>
9  <body>
10 <xsl:value-of select='translate("-----Boks-----"
11 -----","bok","ook")' />
12 <br/>
13 <xsl:value-of select="concat('AWARD WINNING', 'BOOK DETAILS')"/>
14 <xsl:apply-templates select="/Book"/>
15 </body>
16 </html>
17 </xsl:template>
18 <xsl:template match="Book">
19 <xsl:if test="contains>Title, 'Pattern')">
20 <DIV>
21 <B>
22 <xsl:value-of select="Title"/>
23 </B>
24 by
25 <I>
26 <xsl:value-of select="Author"/>
27 </I>
28 costs
29 <xsl:value-of select="Price"/>
30 .
31 </DIV>
32 </xsl:if>
33 </xsl:template>
34 </xsl:stylesheet>
```



String Functions 5-5

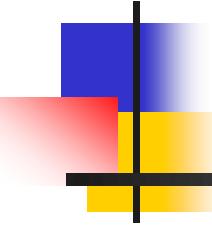
Output

```
-----Books-----  
AWARD WINNINGBOOK DETAILS  
The Weather Pattern by Weather Man costs 100.00.  
Weaving Patterns by Weaver costs 150.00.  
Speech Pattern by Speaker costs 15.00.
```



Result Tree Fragments

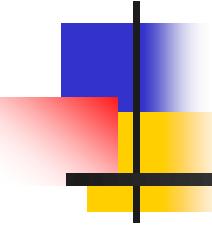
- A portion of an XML document that is not a complete node or set of nodes.
- The only allowed operation in a result tree fragment is on a string.
- The operation on the string may involve first converting the string to a number or a boolean.
- Result tree fragment is an additional data type other than four basic XPath data types, such as, string, number, boolean, node-set.
- A result tree fragment represents a fragment of the result tree.
- It is not permitted to use the /, //, and [] XPath operators on Result tree fragments.



Lesson 3 – Working with different Styles

In this last lesson, **Working with different Styles**, you will learn to:

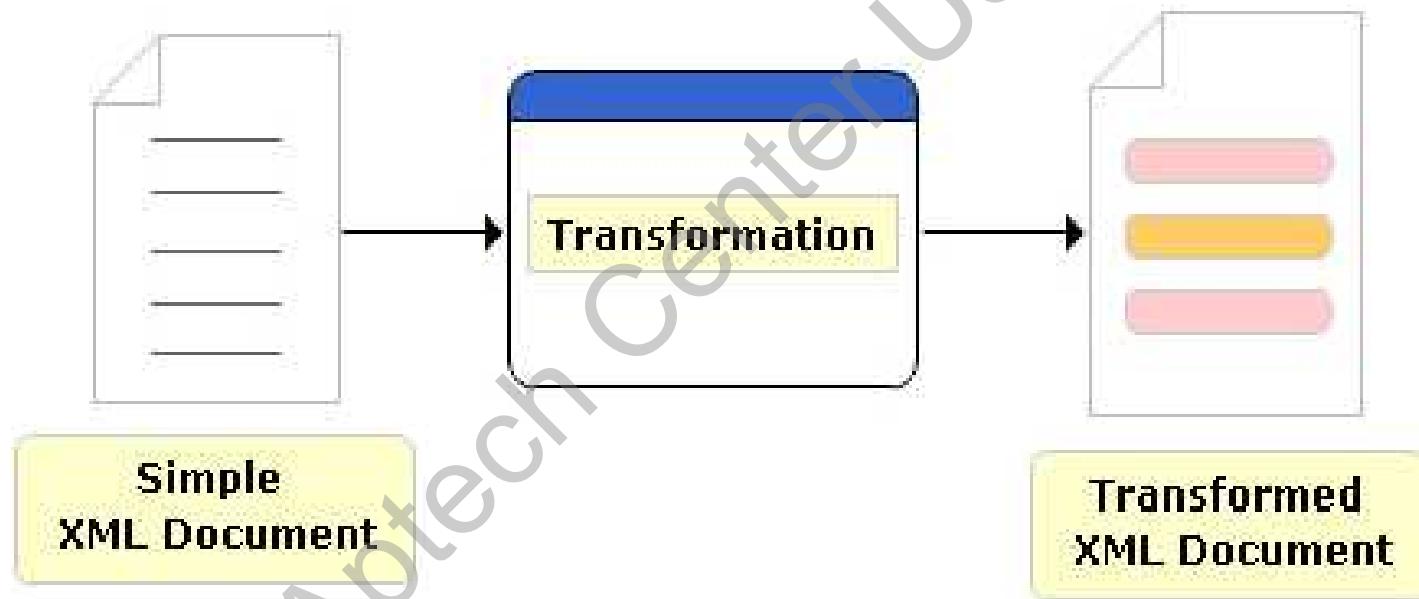
- Explain how to switch between styles.
- Describe how to transform XML documents into HTML using XSLT.

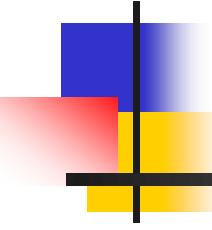


Transformation of XML Document 1-2

- 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.
- Uses of transformation:
 - Formatting a document to create a high-quality presentational format.
 - Changing one XML vocabulary to another.
 - Extracting specific pieces of information and formatting them in another way.
 - Changing an instance of XML into text.
 - Reformatting or generating content.

Transformation of XML Document 2-2

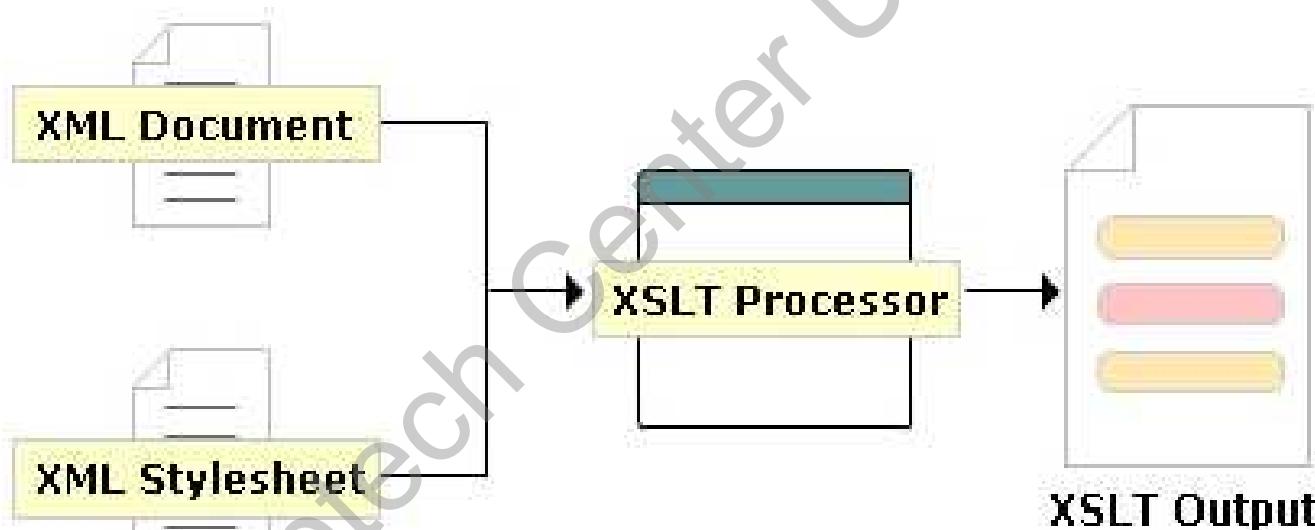


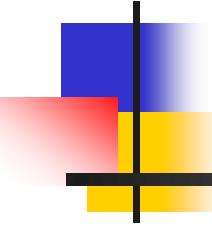


Transformation using XSLT Processor 1-2

- Takes two inputs:
 - An XSLT stylesheet to govern the transformation process.
 - An input document called the source tree.
- The output is called the result tree.
- The XSLT engine begins by reading in the XSLT stylesheet and caching it as a look-up table.
- XPath locates the parts of XML document such as Element nodes, Attribute nodes and Text nodes.
- 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.

Transformation using XSLT Processor 2-2





Transforming XML using XSLT 1-2

- **Step 1**

- Creates a normal XML document.

Code Snippet

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

- **Step 2**

- Add the following lines.

Code Snippet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0" >
    ...
</xsl:stylesheet>
```

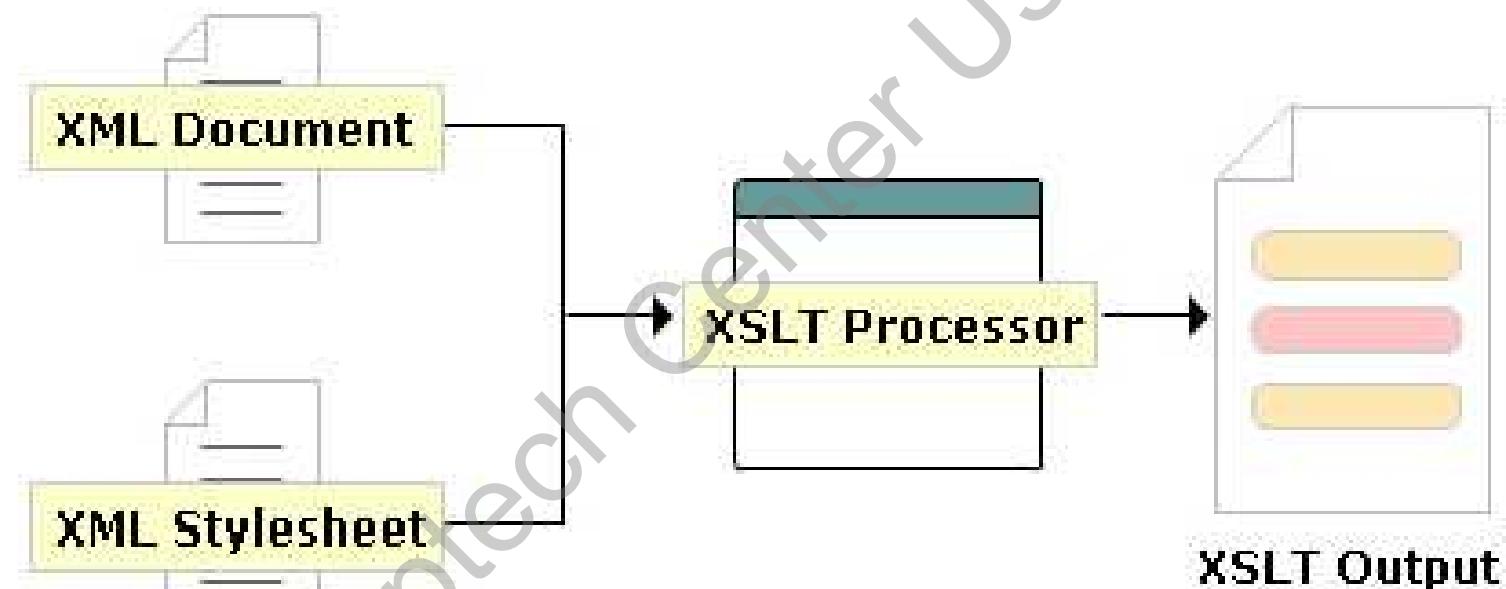
- **Step 3**

- Set it up to produce HTML-compatible output.
- The `<xsl:output>` tag should be used with either “text” or “html” to output anything besides well-formed XML.
- The default value is “xml”.

Code Snippet

```
<xsl:stylesheet>
    <xsl:output method="html"/>
    ...
</xsl:stylesheet>
```

Transforming XML using XSLT 2-2



Transforming XML using XSLT Example 1-2

- Example code of transforming XML documents into HTML using XSLT processor

Code Snippet

```
1  <?xml version="1.0"?>
2  <?xml-stylesheet type="text/xsl" href="ProductInfo.xsl"?>
3
4  <Company>
5
6    <Product>
7      <Product_ID>S002</Product_ID>
8      <Name>Steel</Name>
9      <Price>1000/tonne</Price>
10     <Boiling_Point Units="Kelvin">20.28</Boiling_Point>
11     <Melting_Point Units="Kelvin">13.81</Melting_Point>
12   </Product>
13
14   <Product>
15     <Product_ID>S004</Product_ID>
16     <Name>Iron</Name>
17     <Price>5000/tonne</Price>
18     <Boiling_Point Units="Kelvin">34.216</Boiling_Point>
19     <Melting_Point Units="Kelvin">23.81</Melting_Point>
20   </Product>
21
22 </Company>
```

where,

Company: The root Company element contains Product child elements.

Units: Attribute specifies the units for products melting point and boiling point.

Transforming XML using XSLT Example 2-2

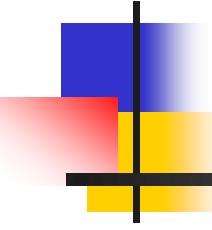
Stylesheet

```
1 <?xml version="1.0"?>
2
3 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4
5 <xsl:template match="Company">
6   <html>
7     <xsl:apply-templates/>
8   </html>
9 </xsl:template>
10
11 <xsl:template match="Product">
12   <P>
13     <xsl:apply-templates/>
14   </P>
15 </xsl:template>
16
17 </xsl:stylesheet>
```

Output

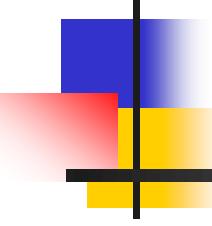
S002Steel1000/tonne20.2813.81

S004Iron5000/tonne34.21623.81



Summary 1-2

- **XPath**
 - Notation for retrieving information from a document.
 - Provides a common syntax for features shared by Extensible Stylesheet Language Transformations (XSLT) and XQuery.
 - Have seven types of node as Root, Element, Attribute, Text, Comment, Processing instruction and Namespace.
 - Used in the creation of the patterns.
- **XPath Expressions and Functions**
 - The four types of expressions in XPath are
 - Node-sets.
 - Booleans.
 - Numbers.
 - Strings.



Summary 2-2

- **XPath Expressions and Functions (contd...)**
 - Functions defined for XPath are Accessor, AnyURI, Node, Error and Trace, Sequence, Context, Boolean, Duration/Date/Time, String, QName and Numeric.
 - A Result tree fragment is a portion of an XML document that is not a complete node or set of nodes.
- **Working with different styles**
 - Transformation is one of the most important and useful techniques for working with XML.
 - To transform XML is to change its structure, its markup, and perhaps its content into another form.
 - Transformation can be carried out using an XSLT processor also.

Session 8

Introduction to JSON

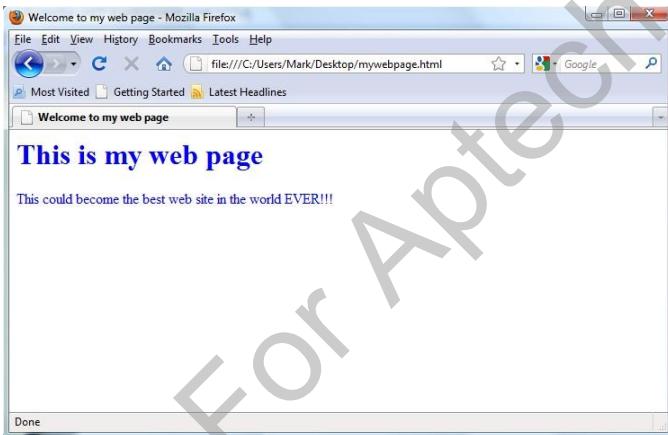
For Aptech Center Use Only

Objectives

- 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

Introduction: The Need of JSON

- XML parses using Document Object Model (DOM) with complex processes and no cross-browser compatibility.
- JavaScript engine incorporates a markup language into an HTML Web page.
- The need of more natural format native to this engine and overcome DOM limitations was felt.



```
25 </head>
26 <body text="#000000"
27   bgcolor="#FFFFFF">
28   <table width="1000"
29     <tr>
30       <td width="200"
31       <td valign="top"
32         <div align="center"
33           </div>
34         <p class="BodyText">
35           <h1 class="HeaderText">
36             <p class="CaptionText">
37               Entertainment</a>
38               | <a href=
```

What is JSON

- A lenient text format derived from JavaScript language and ECMAScript.
- Platform-as well as language-independent.
- Implements conventions of the C-based languages.
- More concise than XML.



JSON is a lightweight, text-based, and open standard data-interchange format.

Uses of JSON

- ❑ For serializing data to be sent over a network
- ❑ For facilitating communication between incompatible platforms
- ❑ For developing JavaScript based applications
- ❑ For sending public data through Web services and Application Programming Interfaces (APIs)
- ❑ For fetching data without hard coding in HTML

History of JSON

Coining the Term

- Originally stated by Douglas Crockford
- Coined initially at the State Software Company

The Co-founders' Decision

- To provide an abstraction layer for devising stateful Web applications without plugins

Participation by Other Companies

- Implementing frames to send information to the browsers without refreshing the page

Crockford's New Discovery

- Implementing JavaScript as an object-based format for messaging
- JSON.org in 2002

Features of JSON 1-2

- Standard structure
- Simplicity
- Open-source
- Self-describing
- Lightweight
- Scalable/Reusable

Features of JSON 2-2

- Clean data
- Efficiency
- Interoperability
- Extensibility
- Internationalization

JSON versus XML 1-3

Characteristic	JSON	XML
Data Types	Offers scalar data types.	Does not offer any idea of data types.
Array Support	Provides native support.	Expresses arrays by conventions.
Object Support	Provides native support.	Expresses objects by conventions.
Null Support	Recognizes the value natively.	Mandates the use of xsi:nil on elements.
Comments	Does not support.	Provides native support (via APIs).

JSON versus XML 2-3

Characteristic	JSON	XML
Namespaces	Does not support namespaces.	Accepts namespaces to prevent name collisions.
Formatting	Is simple and offers more direct data mapping.	Is complex and needs more effort for mapping application types to elements.
Size	Has very short syntax.	Has lengthy documents.
Parsing in JavaScript	Needs no additional application for parsing.	Implements XML DOM and requires extra code for mapping text to JavaScript objects.
Parsing	Has JSONPath.	Has XPath specification.
Learning Curve	Is not steep at all.	Is steep.

JSON versus XML 3-3

Characteristic	JSON	XML
Complexity	Is complex.	Is more complex.
Schema (Metadata)	Has a schema but is not widely used.	Uses many specifications for defining a schema.
Styling	Has no special specification.	Has XSLT specification for styling an XML document.
Querying	Has specifications such as JSON Query Language (JQQL) and JSONiq but are not widely used.	Has XQuery specification that is widely used.
Security	Is less secure, as the browser has no JSON parser.	Is more secure.

JSON versus XML Coding

JSON

```
{  
  "students":  
  [  
    { "name": "Steve",  
      "age": "20",  
      "city": "Denver"},  
  
    { "name": "Bob",  
      "age": "21",  
      "city": "Sacramento"}  
  ]  
}
```

XML

```
<students>  
  
  <student>  
    <name>Steve</name>  
  
    <age>20</age>  
  
    <city>Denver</city>  
  </student>  
  
  <student>  
    <name>Bob</name>  
  
    <age>21</age>  
  
    <city>Sacramento</city>  
  </student>
```

Advantages of JSON

- Quick serialization
- Efficient encoding
- Faster parsing than XML and Yaml Ain't Markup Language (YAML)
- Simpler to work with other languages
- Easy differentiation between data types

Limitations of JSON

- Difficult in reading the format and precision of keeping brackets in its place
- Following snippet illustrates complexity:

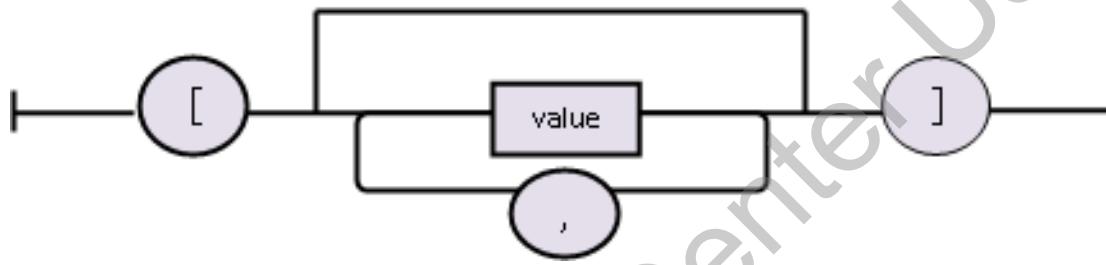
```
{  
  "problems": [ {  
    "Diabetes": [ {  
      "medications": [ {  
        "medicationsClasses": [ {  
          "className": [ {  
            "associatedDrug": [ {  
              "name": "Asprin", "dose": "", "strength": "500 mg"  
            } ]  
          } ]  
        } ]  
      } ]  
    } ]  
  } ]  
}
```

JSON Data Structures

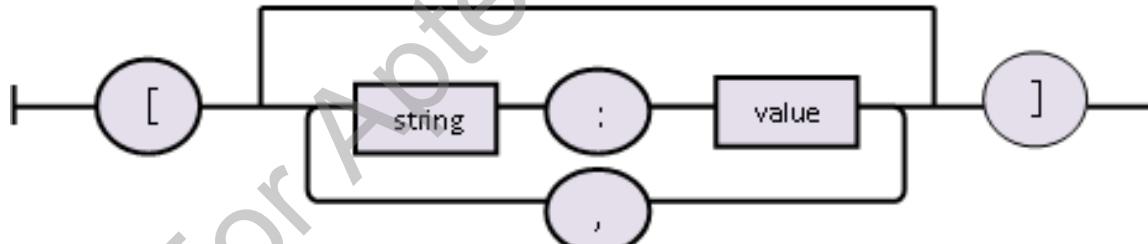
- Support for two universally-supported data structures:
 - Ordered list of values, which can be an array, vector, list, or sequence.
 - Unordered set of name/value pairs, which can be a keyed list, object, record, struct, or a hash table.

JSON Arrays and Objects

- Following figure represents an array (ordered set) in JSON:



- Following figure represents an object (unordered set) in JSON:



JSON Objects

- Following snippet represents a JSON object:

```
{  
  "Name": "Janet George",  
  "Street": "123 Ashley Street",  
  "City": "Chicago",  
  "Code": "60604"  
}
```

JSON Syntax Rules

Arrays

- Always in square brackets

Objects

- Always in curly braces
- Always in name value pairs separated by commas, with name and value delimited by colon

Literal Notation in JavaScript

- Expresses fixed values literally.
- Is a floating-point number, integer, string, Boolean, array, or object.
- Contains zero or more expressions in a sequence in an array literal.
- Contains a list of members in an object literal.

JavaScript Array Literal Notation

1-2

- Following snippet defines an array using literal notation:

```
var fruits = ["Pineapple", "Orange", "Melon",
"Kiwi", "Guava"];

alert (fruits[4] + " is one of the " +
fruits.length + " fruits.");
```

JavaScript Array Literal Notation

2-2

- Following snippet defines an object using literal notation:

```
var Address = {  
    "Street": "123 New Mansion Street.",  
    "City": "Denver",  
    "PostalCode": 80123 };  
  
alert ("The product will be sent to postal code  
" + Address.PostalCode);
```

From JavaScript Literals to JSON

- JSON has stricter rules:
 - Object member name to be a valid JSON string in quotation marks
 - A member value or an array element in JSON to be confined to a restricted set
- JSON does not support Date/time literals.



Creating a JSON File

- The `JSON.stringify()` function converts a JavaScript value to a serialized JSON string.
- Following is the syntax:

```
JSON.stringify(value [,replacer] [,  
space])
```

where,

`value` identifies a JavaScript value.

`replacer` represents a function or array.

`space` adds line break and white space.

Summary 1-2

- 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.

Summary 2-2

- 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.

Session 9

Work with JSON Data

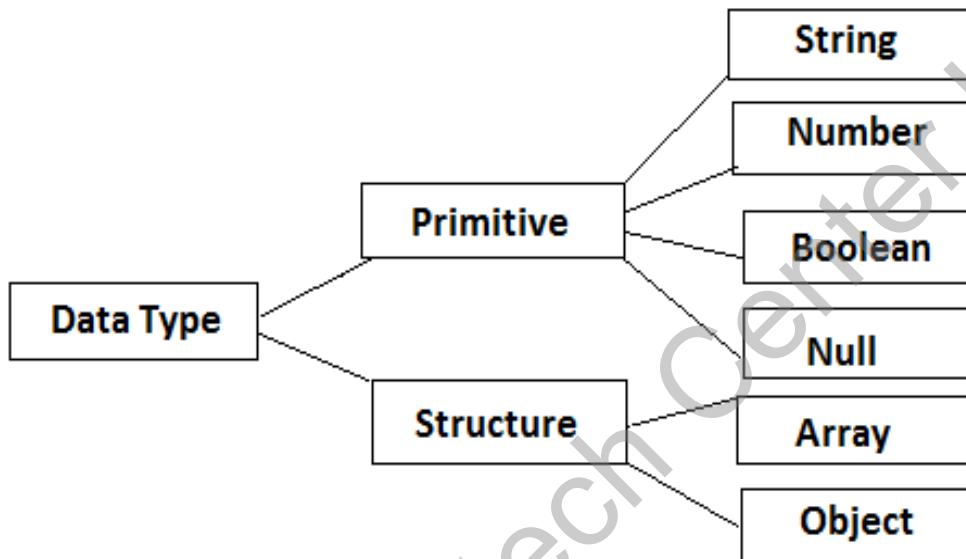
For Aptech Center Use Only

Objectives

- 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

JSON Data Types 1-4

The two primary data types supported by JSON are primitive and structure.



In JSON, the type of a variable is recognized automatically during the parsing phase.

JSON Data Types 1-4

Number

- It is a floating-point format (double precision).
- Does not accept Octal, hexadecimal, NaN, and Infinity values.
- The types allowed are integer, fraction, and exponent.
- **Syntax** `{"string": number_value,
.....}`

String

- A series of zero or other Unicode characters within double quotes and backslash escapes.
- Delimited with double-quotation marks.
- A character denotes a single character string.
- **Syntax** `{"string": "string value",
.....}`

JSON Data Types 2-4

Boolean

- Has only two values: true and false.
- Using quotes for Boolean values treats them as String values.
- **Syntax** {**string**: **true/false**,
.....}

Null

- Is an empty type.

White Space

- A white space can appear between the characters in string values to make code more comprehensible.
- **Syntax** {**string**: " ",

JSON Data Types 3-4

Arrays allow storing various values of the same type in one variable.

Syntax `[value,]`

The characteristics of an array in JSON are:

- It is a sequential collection of values, not necessarily of the same type.
- Indexing begins at 0 or 1.
- It is enclosed in square brackets [].
- Each value is set apart by comma (,).

JSON Data Types 4-4

An object is an independent data type, having its own attributes.

Syntax { string : value, }

The characteristics of an object in JSON are:

It is a non-sequential (having no order) set of key/value pairs.

It is enclosed in curly braces {}.

Each key/value pairs are set apart by comma (,) and every key is proceeded by colon (:).

The keys are only strings, each differing from the other.

It is used when the key names are random strings.

JSON Value

- In JSON, value can be of any primitive and structure data type.
- A JSON value can be a number, string, Boolean such as true or false, null, object, or an array.
- The following Code Snippet shows some examples of JSON values:

```
var emp-no = 1234;  
var name = "Sherill";  
var experience = null;
```



Storing Different Values in Arrays

JSON arrays can store elements of different types.
Following is a sample code:

```
[ 456, "Dog", 123, "Frog", true ]
```

- Elements 0 and 2 in the array are of the type Number.
- Elements 1 and 3 are of String type.
- Element 4 is a Boolean type.

In JSON, arrays can also contain other arrays.
This is called nesting of arrays.

Data Structures Supported by JSON

In JSON, the built-in data structures can be used to develop other data structures.

The two data structures supported by JSON are:

Collection of Name/Value Pairs



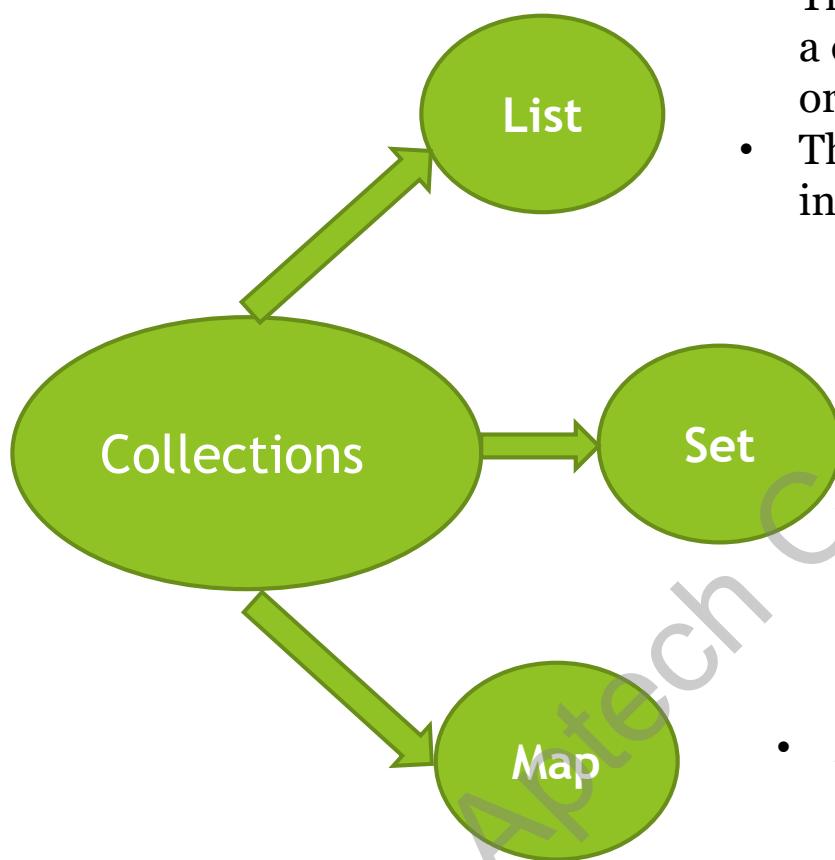
- Supported by different programming languages, in the form of objects.

Ordered List of Values



- Includes arrays, lists, and vectors.

Collections: Maps, Sets, and Lists



- The elements in a list follow a defined order and copies or duplicates are allowed.
- The elements are positioned in a specific location.
- Collection without duplicate elements.
- In JSON, it is not possible to avoid replicas, however, the parser removes the duplicates while serializing the data.
- A type of data structure that mainly aids in quick searching of data.
- Accepts data in the form of key and value pairs and each key is distinct.

JSON Schema

- JSON schema specifies the rules that defines the structure of a JSON document.
- The two main aspects that define why a JSON schema language is required are:

To specify JSON data structures

To validate JSON data structures

Handy when the JSON-based Web services need to be made accessible to a large audience

Handy when validating JSON documents from other applications

Helps to avoid parsing an invalid data structure in valid JSON documents

Schema Overview

- **Application/schema+json** is the media type described by JSON schema and prescribes the design of JSON documents.
- It offers provision for defining the structure of the documents with respect to the permitted values, descriptions, and decoding connections to other resources.
- The JSON schema format is arranged in the following individual definitions:

Core Schema Specification

- Explains a JSON structure and states valid elements in it.

Hyper Schema Specification

- Explains the elements in a JSON structure that can be considered as hyperlinks.

JSON Comments

- JSON does not have any provision for documentation or comments.
- However, comments are still supported by a few JSON parsers and must be provided within /* ... */.

```
{  
    "id":2,  
    "title":"The fallen Hero", /* This books  
is about Harvey Dent */  
    "noOfCopies":14,  
    "tags": [  
        "BatMan",  
        "Gowtam"  
    ],  
}
```

Creating and Parsing JSON Messages with JavaScript

The `JSON.stringify()` method allows converting a JavaScript object into a JSON String.

The `JSON.parse()` method allows parsing a JSON object using JavaScript.

It converts a JSON String to an object in JavaScript.

Start by defining a string in JSON format, using the method for conversion, and then looping through the attributes for printing its values.

JSON with Developer Tools on Browser

- There are many plugins or extensions that help in validating and formatting a JSON document or a JSON HTTP response.
- One such extension in Firefox is **JSONView**, which allows viewing a JSON document.
- With **JSONView** the JSON document is displayed in the browser.
- **JSONView** displays the raw text even though the JSON document has errors.
- Chrome also offers **JSONView** for validating a JSON document.
- For modifying the JSON values during runtime, it is recommended converting the JSON document to a JavaScript object before using the built-in browser developer tools.

Online Tools and Editors

- **JSONLint** is an open source project that helps in validating JSON data.
- Using **JSONLint** or any other online tool is easy, as it only requires copying the JSON data to its online editor.
- In case of an error, the output similar to the following is displayed:

```
Error: Parse error on line 4:  
...": {      "firstName": "James",      "lastname"  
-----^  
Expecting 'STRING', 'NUMBER', 'NULL', 'TRUE', 'FALSE', '{}', '[]', got 'undefined'
```

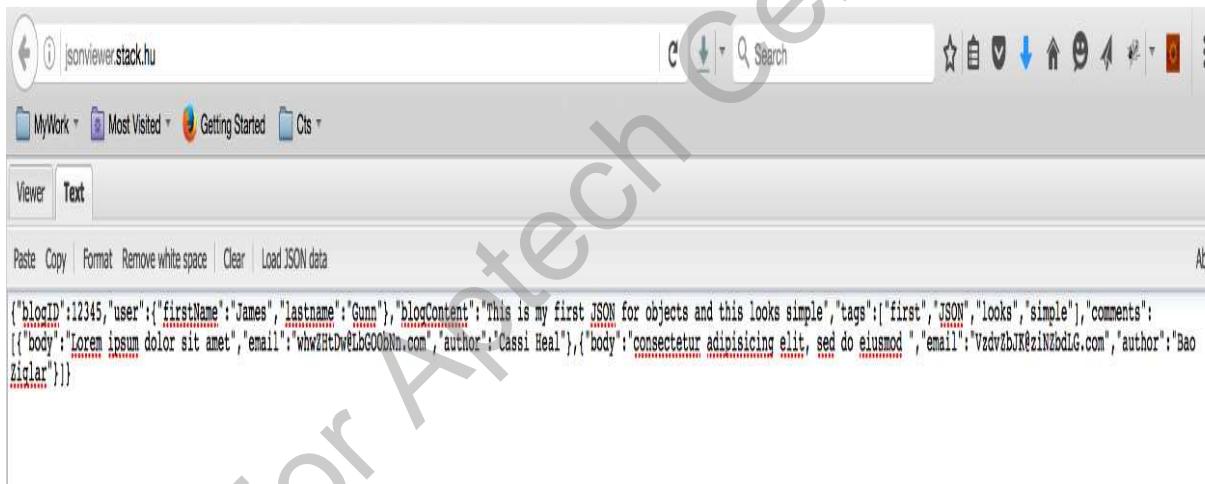
- Output shows the line number and the type of error that occurred.
- If everything is fine, the following output is displayed:

Results

Valid JSON

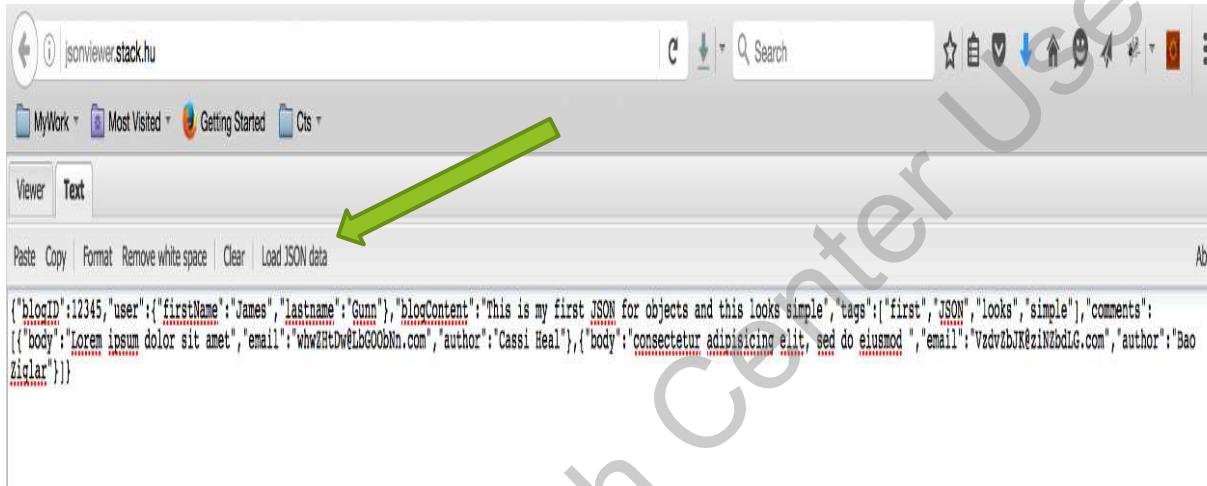
Online JSON Viewers 1-2

- While working with JSON, it is often required to use a JSON viewer to see how the document will look in the browser.
- One of the most widely used is jsonviewer.stack.hu and JSON text can be directly copied to this online viewer.
- The following screenshot shows the **jsonviewer.stack.hu**:



Online JSON Viewers 2-2

- Another way of supplying data to the viewer is by clicking **Load JSON data**.



- The viewer takes data from the URL you provide and loads data from that source.
- Once the JSON code is visible in the viewer, clicking **Format** formats the code.

Summary

- 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.

Session 10

JSON in Real World

For Aptech Center Use Only

Objectives

- 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

Support for JSON

- Each major programming language can incorporate JSON, natively or through libraries.
- This is possible by incorporating two functionalities namely:
 - Parsing
 - Formatting

JSON with C# and Java

- These languages support statically typed classes and not objects of HashMap or Dictionary type.
- The solution is to use a library along with a custom code for converting these structures into static type instances.
- The Gson library from Google is one such library that resolves the issue.

What is Gson

- ❑ Is an open-source Java library for transforming a Java object to JSON data and vice-versa.
- ❑ Offers easy mechanisms such as constructor (factory method) and `toString()`.
- ❑ Functions well with arbitrary Java objects, involving the pre-existing ones.
- ❑ Serializes and deserializes huge data without any issues.
- ❑ Is convenient to learn and use by only using `toJson()` and `fromJson()`.

JSON with PHP

- From 5.2.0 version, the extension for JSON is packaged into PHP.
- Following table shows the functions for encoding and decoding JSON structures:

Function	Description
<code>json_encode</code>	Serializes the stated array or object and returns it in the JSON format if successful or FALSE.
<code>json_decode</code>	Deserializes JSON data and returns the suitable PHP type.
<code>json_last_error</code>	Returns the error that happened last.

MIME Type of JSON

- Is also called media type or content type.
- Is a two-part identifier composed of a type and subtype isolated by a slash.
- Aids in identifying the type of formatted content being sent over the Web.
- Is ‘application/json’ for JSON.
- Is unofficially ‘text/json’ or ‘text/Javascript’.

Applications of JSON

APIs

- For exchanging data to and from APIs
- Popular in social networking sites implementing API

NoSQL

- For storing data easily in NoSQL databases, as JSON is easily convertible into JavaScript

Asynchronous JavaScript and JSON (AJAJ)

- For replacing XML when new data is fetched by a loaded Web page in a browser

JSON-RPC (Remote Procedure Call)

- For replacing Simple Object Access Protocol (SOAP) or XML-RPC and for sending several calls and notifications to a server

Package Management

- For storing metadata

JSON HTTP and Files

- JSON displays the data fetched from a Web server efficiently through the XMLHttpRequest object.
- The object exchanges data in the background, which prevents reloading the full page for updates.
- Following is the syntax of initializing the object:

```
variable = new XMLHttpRequest();
```

JSON for Data Storage

JSON helps in storing data fetched from services and applications.

- **Reasons:** Simplicity and ‘just adequate structure’

RDBMS and Structured Query Language (SQL) are now replaced by NoSQL databases for developers who prefer JSON.

- **Trends:** More implementation of JSON-centric document databases, JSON in traditional RDBMS'
- **Reasons:** Developer-friendly and agility

Document versus Relational Databases

	Document-oriented Databases	Relational Databases
Coupling	Tighter due to the need to navigate the document while querying	Loose
Portability and Standardization	No, due to new query language setup for each document store	Yes, due to well-defined SQL query core
Optimization	Restricted, due to no abstraction between the logical data structure and its physical storage	Fully optimized, due to abstraction ensured by highly queryable stored data definition
Consistency and Normalization	No, due to no support for constraints	Yes

Benefits of Using JSON in RDBMS



Decision Power for:

- The data portions to be abstracted
- Areas where flexibility is essential
- Locations where strict schema and constraints are essential



JSON Query as Native SQL:

- Allows querying randomly across JSON records in tables
- Allows expanding systems to use JSON

JSON versus RDBMS Data Models

	JSON	RDBMS
Storage Structure	Is array or object.	Is a table.
Metadata	Can be in a schema, but is not pre-created.	Is stored in a schema generated while creating a table.
Data Retrieval	Uses evolving languages namely, JSON Query Language (JAQL) and JSONiq	Uses SQL.
Sorting	Is only for arrays.	Is for tables.
Learning Curve	Is smoother.	Is time consuming.
Application	Is used in several programming languages.	Is in the form of many commercial and open-source databases.

Security Issues in JSON

- JSON, as a flexible subset of JavaScript, has some security issues.
- In the following figure the root cause increasing the risk of malicious script, a major issue:

```
var data = eval(' (' + JSONresponse + ') ');
```



```
<img src=x onerror=
"alert('Alert: malware
has been detected on
your computer. Visit
cleanmyinfectedcomputer.com
to clean your computer.');
"
```

Issues with Eval()

- Most JSON text is syntactically JavaScript.
- JavaScript interpreter converts JSON into a JavaScript object without validation.
- This increases the risk of authentication forgery, identity and data theft, and misuse of resources.
- **Solution:** `JSON.parse()` and `JSON.stringify()` functions processing text only in JSON format

XSS and CSRF Issues

Cross Site Scripting (XSS)

Request: http://vulnerablesite.local/index?name=<script>alert("xss")</script>

Response:

```
<html>
  <body>
    <div>
      Hello <script>alert("xss")</script>
    </div>
  </body>
</html>
```

Cross Site Request Forgery (CSRF)

Request:

http://vulnerablesite.local/changepassword?newpwd=MyS3cr3tPa\$\$word

Attack:

```

```



XSS and CSRF Solution

By using the `jsonify()` function

Implementation Issues

Denial of Service (DoS) Attack

- Renders a resource or a machine on a network unavailable to its targeted users

Mass Assignment Vulnerability

- Maltreats the functioning pattern of record in a Web application for illegitimately changing confidential data items

Portability Issues

Unicode Line Terminators

- Are allowed in JSON without being escaped and that they need to be backslash escaped for portability.

Null Character

- Is allowed in JSON string if escaped as “\u0000”, which creates issues with C strings.

UTF Encoding

- Involves having a few escaped characters using UTF-16 surrogate pairs, which a few JSON parsers do not recognize.

JSON Numbers

- Involve numbers and floating integers, which are distinguished by some languages only, not by all.
- Have no specifications for rounding, overflow, and precision loss.

Unsupported Data Types

- Involve Error, Date, Undefined, Function, and Regular Expression, which JSON does not accept.

Handling JSON Securely

- Avoiding `eval()` by using a JavaScript library available at www.json.org, such as JSON sans `eval()`
- Ensuring data integrity via XMLHttpRequests

Summary 1-2

- 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.

Summary 2-2

- 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.