Element Syntax. XML Elements Must Have a Start and End Tag. Naming rules/conventions:XML element names are case sensitive.You cannot start an element name with a number or a special character (such as !,@, #, $, %, and so on).Element names cannot contain spaces.Element names cannot start with the word XML in any case. Immediately after the opening < and </ of the XML tags, you must start the element name

Entity References: < : &lt; > : &gt; “ : &quot; ' : &apos; & : &amp; use CDATA for large sections needing entity refs <Value><![CDATA[Do if 5 < 3]]></Value>

Shema- Global declarations are declarations that appear as direct children of the root <schema> element.Global element declarations can be reused throughout the XML. Local declarations can be used only in their specific context.In XML Schema, the xs:simpleContent element is used to define complex types that allow the modification or extension of simple types. It is primarily used when you want to add attributes or additional content to a simple type.

XMLandSchemaFileUpload is the model class used to bind form data on the view for uploading XML and schema files. IFormFile is an interface defined in Microsoft.AspNetCore.Http namespace for the text stream received via HTTP from the network as an uploaded file.

```
[HttpPost]

public IActionResult Index(XMLandSchemaFileUpload xmlAndSchemaFiles)

{

        IFormFile schemaFile = xmlAndSchemaFiles.SchemaFile;

        IFormFile xmlFile = xmlAndSchemaFiles.XmlFile;

        XmlReader schemaReader = XmlReader.Create(schemaFile.OpenReadStream());

        XmlSchemaSet sc = new XmlSchemaSet();

        sc.Add(null, schemaReader);

        XmlReader xmlReader = XmlReader.Create(xmlFile.OpenReadStream(), settings);

}

// Set the validation settings.

XmlReaderSettings settings = new XmlReaderSettings();

settings.ValidationType = ValidationType.Schema;

settings.Schemas = sc;

List<XmlValidationError> validationResults = new

List<XmlValidationError>();

//the event handler will be called whenever an error is encountered while reading the XML file.
```

XML serialization refers to the process of converting data from an object-oriented programming language, such as Java, C#, or Python, into XML format. deserialization is the process of converting XML data into classes

XmlSerializer serializor = new XmlSerializer(typeof(ClassName)); serializor.Serialize(System.IO.TextWriter, Object);

Change root element name: XmlRootAttribute ra = new XmlRootAttribute(); ra.ElementName = "AlgonquinBookStoreBookOrder";

Add namespace: ra.Namespace = "hppt://www.algonquincollege.com/onlineservice";

```
FileStream xs = new FileStream("C:/bookorder.xml", FileMode.Open);XmlSerializer serializor = new XmlSerializer(typeof(BookOrder));

BookOrder order = (BookOrder ) serializor.Deserialize(xs);xs.Close( );  //deserializing

string xmlFilePath = Path.GetFullPath("Data/restaurant_reviews.xml");

            FileStream xs = new FileStream(xmlFilePath, FileMode.Open);

            XmlSerializer serializor = new XmlSerializer(typeof(Restaurants));

            Restaurants r = (Restaurants)serializor.Deserialize(xs);

$xml = "<root><node1>content</node1></root>";$sxe = new SimpleXMLElement($xml);
```

Or load and XML from a file:$sxe = simplexml_load_file("filename.xml");

$book = simplexml_load_file('book.xml'); //Access the bookinfo child element of the book element $bookinfo = $book->{'book-info'};

attributes of an element are kept in an associative array named after the element.

```
<?xml version="1.0" encoding="utf-8"?> <book category="Computer"><title lang="en">PHP Programming</title><author>John Simth</author></book>

$xml=simplexml_load_file("book.xml");echo (string)$xml->book['category’]; echo (string)$xml->book->title['lang'];?>
```

To access the contents of an element, use the same SimpleXMLElement objects. When an element is a simple element (contains no child elements), the content of the element is the first element of SimpleXMLElment object. When the print statement is used with the object, the content is returned as a string.

For repeating elements, the corresponding SimpleXMLElement object is an array and iterate-able.

```
<?php $book = simplexml_load_file('Book.xml');$para = $book->chapter->para; print "<h1>".$book->chapter->title."</h1>";

foreach($para AS $node){print "<p>$node</p>";}?>
```

Repeating child elements can also be accessed using 0 based index.When object is accessed without index, the first element in the array is accessed. To read from and write to an attributes of an element, use the name of the attribute as the index.

```
<?php $book = simplexml_load_file('Book.xml');print $book['lang'];print $book->chapter[0]['id'];$book['lang'] = "fr";print $book['lang'];?>
```

You can edit elements containing child elements, in the same way as with text content.The child elements are removed from the document and replaced with the text content. Even if the new text content contains XML elements, they will be used as strictly text content.

To replace the subtree with another subtree, you can use the DOM

```
// Remove all child elements of the bookinfo element while ($bookinfo->firstChild){$bookinfo->removeChild($bookinfo->firstChild);}

$bookinfo->appendChild(new DOMElement("title", "SimpleXML in PHP 5"));

$book->chapter->para[1] = ""; //empty an element //remove an element unset($book->chapter->title); //remove an element
```

Attributes can be created using SimpleXML<?php $book = simplexml_load_file('Book.xml');$book->bookinfo->author["prefix"] = "Mr.";?>

```
//remove attribute unset( $book->bookinfo->author["prefix"]);
```