SYLLABUS

XML: Working with Basics of XML, Implementing Advanced Features of XML, Working with XSLT.
AJAX: Overview of AJAX, Asynchronous Data Transfer with XMLHttpRequest, Implementing AJAX Frameworks, Working with jQuery.

XML: Working with Basics of XML

Exploring XML:

- E**x**tensible **M**arkup **L**anguage (XML) is a markup language on simple and platform-independent tool for storing and transporting data.
- HTML describes how to display data on the screen.
- **For example:**

```
<p>
        <B>Mr. Ram Sharma</B><br/>
        6-3-2, Main Street<br/>
        Bapatla, 522101
</p>
```

Output:

- Here, it's not possible to extract pin code directly using HTML. Because HTML does not allow specific part of the text.
- Let see the XML representation:

```
<?xml version="1.0">
  <address>
    <NAME>
        <title>Mr.</title>
        <first-name>Ram</first-name>
        <last-name>Sharma</last-name>
    </NAME>
    <street>6-3-2, Main Street</street>
    <city>Bapatla</city>
    <zip-code>522101</zip-code>
  </address>
```

- Here, it's possible to extract specific part of text directly using tag name using XML.

Features of XML

- **Structured language:** Defines the structure of document using tags.
- **Platform independent:** XML supports any operating system.
- **Open standard:** creates fair, competitive market for implementation of the standard.
- **Language independent:** XML support all technologies or languages like java, .NET, and PHP.
- **Web enable:** HTML content can be re used to work with XML.
- **Extensible:** allows you to create user defined tags.

The Difference between XML and HTML

No.	HTML	XML
1)	HTML is used **to display data** and focuses on how data looks.	XML is a software and hardware independent tool used **to transport and store data**. It focuses on what data is.
2)	HTML is a **markup language** itself.	XML provides a **framework to define markup languages**.
3)	HTML is **not case sensitive**.	XML is **case sensitive**.
4)	HTML is a presentation language.	XML is neither a presentation language nor a programming language.
5)	HTML **has its own predefined tags**.	You **can define tags according to your need**.
6)	In HTML, it is **not necessary to use a closing tag**.	XML **makes it mandatory to use a closing tag**.
7)	HTML is **static** because it is used to display data.	XML is **dynamic** because it is used to transport data.
8)	HTML **does not preserve whitespaces**.	XML **preserve whitespaces**.

Advantages of XML:

- Store the data in the form plain text.
- Provides basic syntax that can be used to share information among different applications.
- Allows you not to restrict to limited set of tags defined by proprietary vendors.
- Represents the international standards of web.

Disadvantages of XML:

- Represents redundant and similar syntax for binary data.
- Does not support intrinsic data (i.e, integer, string, and Boolean etc).
- Requires a processing application so that anyone, anywhere in the world, can read your document.
- Requires XML specific browsers, which is not in market.

Structure of an XML document:

- XML define certain rules for its syntax that specify how to create or structure an XML document.
- The syntax used to create an XML document is called markup syntax.
- While creating an XML document you must remember the following points.
  - XML document must have starting and closing taga.
  - XML tags are case sensitive.
  - XML elements must be properly nested.
  - XML document must have one root element.
  - XML attributes values must be enclosed in double quotes.
- The structure of XML document as follow

```
<?xml version="1.0" encoding="UTF-8" ?>
<Employee>
        <FirstName>Mabasha</FirstName>
        <LastName>Shaik</LastName>
        <Age>28</Age>
        <EmpID id=”BEC14”></EmpID>
</Employee>
```

- Xml document having the following sections:
  - XML declaration
  - XML elements
  - XML attributes
  - XML tree
  - XML comments

XML declaration:

- XML declaration is the first line in the document.
- The XML declaration statement is used to indicate that the specified document is an XML document
- XML declaration defines the version and character encoding.
- Syntax: **<?xml version="1.0" encoding="UTF-8"?>**
- XML declaration is optional but it is good practice to include it.

XML elements:

- Elements are the building blocks of XML document.
- These divide the document into a hierarchy of sections, each serving a specific purpose.
- Elements are represented by tags.
- A start tag is delimited by the < and > and an end tag is delimited by the </ and > characters.
- Example: **<Employee></Employee>**
- A document must have single root element and it is the top most parent element in an XML.
- Example:

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

- Elements are categorized into two types
- **Empty element:**
  - An empty element does not contain any content or other element within it.
  - It can have attribute that helps in identifying an entity.
  - Example:
```
<?xml version="1.0" encoding="UTF-8"?>
<items>
    <item id=”1234” />
</ items >
```
- **Nested element:**
  - An element contain another elements are known as nested element.
  - In above example <item> is nested in <items> element.
- **Rules while defining elements as follow**
  - Elements name must starts with letters and underscore (_) character.
  - Elements name cannot contain spaces.
  - Elements name cannot contain the **:** character.
  - Elements name cannot start with the word like xml in upper, or lower or mixed.
  - There cannot be space after **<** but may be space before **>** characters.

XML attributes:

- **Attributes** are used to provide additional information about the properties and behavior of HTML elements.
- Attributes are **name-values pairs** separated by the equal (=) sign.
- Attribute values are enclosed within single quotes or double quotes.
- Example: <person gender="male" age="25"> Apple </person>

XML tree:

- An xml tree represents the elements an xml documents in a tree structure.
- An xml contains the following elements.
- **Root element:** contains all other elements and the content of an xml document. They can be only one root element.
- **Parent element:** contains other inherited elements. A parent element can contain multiple child elements.
- **Child element:** Refers to an element that is contained in a parent element. A child element cannot have multiple parent elements.
- **Siblings:** Refers to the elements that are contained in single parent element.

XML comments:

- Comments are used to specify some information or remarks.
- Comments are not a program so that it cannot parse by the parser.
- Syntax: <!--This is Employee element -->

Describing DTD:

- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".
- A DTD (Document Type Definition) defines the structure and the legal elements and attributes of an XML document.
- You can include DTD in an XML document either by declaring it **inline** or as an **external** reference.
- Example for inline

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email  [
<!ELEMENT email (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<email>
    <to>apple@gmail.com</to>
    <from>mango@gmail.com</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</email>
```

- Example for external reference

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email SYSTEM "Email.dtd">
<email>
    <to>apple@gmail.com</to>
    <from>mango@gmail.com</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
</email>
```

Email.dtd

```
<!ELEMENT email (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

- Different ways of defining DTD to the XML structure as follow
  - ✓ Defining DTD for a single element
  - ✓ Defining DTD for a nested element
  - ✓ Define attributes in the DTD
  - ✓ Define entities in the DTD
  - ✓ Refers external entities

Defining DTD for a single element:

- You can defining DTD for a single element, which does not contain any text or other element.
- Example: <!ELEMENT email (to)>
- Here "**email**" is an element and "**to**" is type of element.
- List of qualifiers that can be used in DTD definition as follow.

Qualifier	Name	Description
?	Question mark	Applies to zero or one element
*	Asterisk	Applies to zero or more elements
+	Plus sign	Applies to one or more elements

- **Example1:** <!ELEMENT email (body?)>
  Here "email" element contains zero or one elements of type "body".
- **Example2:** <!ELEMENT email (to*)>
  Here "email" element contains zero or more elements of type "head"
- **Example3:** <!ELEMENT email (from+)>
  Here "email" element contains one or from elements of type "from".

Defining DTD for a nested element:

- You can define DTD for nested elements; in this case the DTD definition informs the parser which elements can contain nested elements or text.
- Example: <!ELEMENT email (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
- Here, the DTD informs the parser that the <email> element must contain the <to>, <from>, <heading>, and <body> elements.
- The <to>, <from>, <heading>, and <body> elements are defined to contain the data of type parsed character data (PCDATA).
- The # sign indicates that it is a special word rather than an element.

- You can define **DTD using the OR condition**.
- Example: **<!ELEMENT note (to, from, header, (message|body))>**
- Here, body element contains message or body.

Define attributes in the DTD:

- Syntax for defining attributes in DTD
  <!ATTLIST element-name attribute-name attribute-type attribute-value>
- XML example: <payment type="check" />
- DTD example: <!ATTLIST payment type CDATA "check">

- The **attribute-type** can be one of the following:

Type	Description
CDATA	The value is character data
(*val1*|*val2*|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

- The **attribute-value** can be one of the following:

Value	Explanation
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED *value*	The attribute value is fixed

Define entities in the DTD:

- Entities are used to define shortcuts or aliases that are replaced by complete definition at runtime.
- Entities can be declared internal or external.

An Internal Entity Declaration

- **Syntax: <!ENTITY entity-name "entity-value">**
- Example
  XML example: <author>&writer;&copyright;</author>
  DTD Example: <!ENTITY writer "Donald Duck.">
  <!ENTITY copyright "Copyright 2022 becbapatla.">
- **Note:** An entity has three parts: an ampersand (&), an entity name, and a semicolon (;).

An External Entity Declaration

- **Syntax: <!ENTITY entity-name SYSTEM "URI/URL">**
- Example
  XML example: <author>&writer;&copyright;</author>
  DTD Example: <!ENTITY writer SYSTEM "entities.dtd">
  <!ENTITY copyright SYSTEM "entities.dtd">

Refers external entities:

- **SYSTEM** or **PUBLIC** identifiers are used to reference the external entities defined in DTD or XML document.
- Example
  XML example: <author>&writer;&copyright;</author>
  DTD Example: <!ENTITY writer SYSTEM "entities.dtd">
  <!ENTITY copyright SYSTEM "entities.dtd">

Example: Create a XML file that contains student information name, regd, branch, and address with DTD validation.

Internal DTD:

```
<?xml version="1.0" encoding=”UTF-8”?>
<!DOCTYPE student [
<!ELEMENT student (name, regd,address)>
        <!ELEMENT regd EMPTY>
        <!ATTLIST regd id CDATA "401">
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT branch (#PCDATA)>
        <!ELEMENT address (dno,street,city)>
        <!ELEMENT dno (#PCDATA)>
        <!ELEMENT street (#PCDATA)>
        <!ELEMENT city (#PCDATA)>
  ]>
<student>
        <regd id="401" />
        <name>apple</name>
        <branch>IT</branch>
        <address>
                <dno>5-2-3</dno>
                <street>Old Buststand</street>
                <city>Bapatla</city>
        </address>
</student>
```

External DTD:

```
<?xml version="1.0" encoding=”UTF-8”?>
<!DOCTYPE student SYSTEM “student.dtd”>
<student>
        <regd id="401" />
        <name>apple</name>
        <branch>IT</branch>
        <address>
                <dno>5-2-3</dno>
                <street>Old Buststand</street>
                <city>Bapatla</city>
        </address>
</student>
```

Student.dtd:

```
<!ELEMENT student (name, regd,address)>
<!ELEMENT regd EMPTY>
<!ATTLIST regd id CDATA "401">
<!ELEMENT name (#PCDATA)>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT address (dno,street,city)>
<!ELEMENT dno (#PCDATA)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
```

Implementing Advanced Features of XML

Exploring XML Namespaces:

- A namespace is a URI that provides uniqueness to the names of elements and attributes of an XML document.
- It helps to avoid the conflicts between the names of elements or attributes while combining different XML document.
- It is not mandatory that every xml document must have namespaces.
- Here we will learn about following aspects of namespaces.
  - ✓ Need of namespaces
  - ✓ Namespace syntax
  - ✓ Scope of namespace declaration
  - ✓ Default namespace
  - ✓ Namespace with DTD

Need of namespaces:

- Namespace provide a method to avoid the element name conflicts.
- An xml we can use a namespace by assigning a name prefix to it.
- A name prefix is a short name assigned to a particular namespace.
- For example:
  The XML carries HTML table information:

```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
```

  This XML carries information about a table (a piece of furniture):

```
<table>
    <name>Coffee Table</name>
    <width>80</width>
    <length>120</length>
</table>
```

- If these XML fragments were added together, there would be a name conflict. Both contain a <table> element, but the elements have different content and meaning.
- A user or an XML application will not know how to handle these differences.
- Name conflicts in XML can easily be avoided using a name prefix.
- Example:

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
<f:table>
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
</f:table>
```

Namespace syntax:

- You can include a namespace in your xml document by using the **xmlns** attribute in a start tag of an element.
- Syntax : **xmlns: prefix=”uri”**
- Example:

```
<root>
    <h:table xmlns:h="http://example1.com">
        <h:tr>
            <h:td>Apples</h:td>
            <h:td>Bananas</h:td>
        </h:tr>
    </h:table>
    <f:table xmlns:f=" http://example2.com ">
        <f:name>African Coffee Table</f:name>
        <f:width>80</f:width>
        <f:length>120</f:length>
    </f:table>
</root>
```

- The xmlns attributes is usually declares in root element

```
<root  xmlns:h=”http://example1.com”  >
  <h:table  >
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
  </h:table>
</root>
```

Scope of namespace declaration:

- This is similar to the scope of variable declaration in a programming language.
- The scope of namespace declaration begins with start tag and ends with any tag.
- For example

```
<h:table xmlns:h="http://example1.com">
        <h:tr>
            <h:td>Apples</h:td>
            <h:td>Bananas</h:td>
        </h:tr>
</h:table>
```

- Here the scope of the namespace declaration starts with the <table> and ends with </table>.
- You can also define multiple namespaces in an xml document.
- For example

```
<root   xmlns:h=”http://example1.com”   xmlns:f=" http://example2.com">
    <h:table >
        …
    </h:table>
    <f:table >
        …
    </f:table>
</root>
```

Default namespace:

- The default namespace applies to the elements and the child elements that do not have any prefixes.
- If the child elements of an element refer to another default namespace, you can override this reference by including a new default namespace in the parent element.
- Example:

```
<root>
        <table xmlns="http://example1.com">
                …
        </table>
        <table xmlns="http://example2.com">
                …
        </table>
</root>
```

- You can declare multiple namespace in document in which there should be one default namespace.
- Example:

```
<root xmlns=”http://example1.com”   xmlns:f="http://example2.com">
      <table  >
         …
      </table>
      <f:table  >
         …
     </f:table>
</root>
```

Namespace with DTD:

- In a DTD, you can specify a namespace by adding the xmlns attribute to the definition of an element.
- Example:

```
<!ELEMENT title   (%inline;)*>
<!ATTLIST   title   xmlns   CDATA   #FIXED   “http://www.example1.com”>
```

- Here the attribute of the title element belongs to the http://www.example1.com namespace.

Describing XML Schema:

- An XML schema is similar to DTD file.
- Xml schema is a document that describes the structure of an xml document.
- It is used to validate the structure of an xml document with the help of an xml parser.
- The main difference between DTD and XML is that the XML schema use XSD (XML schema document)
- XSD is an XML based language describes the structure of an XML document where as the DTD file used a list of elements to describe the document.
- Syntax for define element: **<xs:element name="name" type="type"/>**
- Example: **<xs:element name="apple" type="xs:string"/>**
- Syntax for define attribute: **<xs:attribute name="name" type="type"/>**
- Example: **<xs:attribute name="lang" type="xs:string"/>**
- Here the **name** attribute specifies the name of element or attribute and the **type** attribute specifies the data type of content.
- The schema document can contain various data types which can be divided into two categories.

Pre-define:

- the predefine data types available in XML scheme are as follows:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time

user-define:

- The user-define data types divides into
  i) Simple type element.
  ii) Complex type elements

i) Simple type element:

- Simple type element can contain only text cannot have any child elements or attributes
- Syntax: **<xs:element name="name" type="type"/>**
- Example:

XML document elements

```
<lastname>Apple</lastname>
<age>36</age>
<dateborn>1970-03-27</dateborn>
```

Simple type element for preceding XML elements

```
<xs:element  name="lastname"  type="xs:string"/>
<xs:element  name="age"       type="xs:integer"/>
<xs:element  name="dateborn"  type="xs:date"/>
```

Default and Fixed Values for Simple Elements:

- Simple **elements** may have a **default value** OR a **fixed value** specified.
- A default value is automatically assigned to the element when no other value is specified.
- Example the default value is "red":
  <xs:element name="color" type="xs:string" default="red"/>
- A fixed value is also automatically assigned to the element, and you cannot specify another value.
- Example the fixed value is "red":
  <xs:element name="color" type="xs:string" fixed="red"/>

Default and Fixed Values for Simple Attributes:

- **Attributes** may have a **default value** OR a **fixed value** specified.
- A default value is automatically assigned to the attribute when no other value is specified.
- Example: <xs:attribute name="lang" type="xs:string" default="EN"/>
- A fixed value is also automatically assigned to the attribute, and you cannot specify another value.
- Example: <xs:attribute name="lang" type="xs:string" fixed="EN"/>

Optional and Required Attributes:

- Attributes are optional by default.
- To specify that the attribute is required, use the "**use**" attribute:
- Example: **<xs:attribute name="lang" type="xs:string" use="required"/>**

Restrictions/Facets:

- Restrictions are rules that allow an XML element to accept only particular type of data.
- Restrictions on XML elements are called facets.
- Restrictions are implemented by using the restriction element.
- Syntax: **<restriction id=any_ID base=data_type other_attributes>**
- Restrictions for Data types

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

- You can implement restrictions on various data types as follows
  - ✓ Restrictions on values
  - ✓ Restrictions on a set of values
  - ✓ Restrictions on series of values
  - ✓ Restrictions on whitespace characters
  - ✓ Restrictions on length

Restrictions on values:

- You can limit the value of an element by applying restrictions on values with the help of XML schema.
- The following example defines an element called "age" with a restriction.
- The value of age cannot be lower than 0 or greater than 120:

```
<xs:element name="age">
 <xs:simpleType>
  <xs:restriction       base="xs:integer">
   <xs:minInclusive  value="0"/>
   <xs:maxInclusive  value="120"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Restrictions on a set of values:

- To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.
- The example below defines an element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```
<xs:element name="car">
 <xs:simpleType>
  <xs:restriction       base="xs:string">
   <xs:enumeration   value="Audi"/>
   <xs:enumeration   value="Golf"/>
   <xs:enumeration   value="BMW"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

- The example above could also have been written like this:

```
<xs:element    name="car"    type="carType"/>
<xs:simpleType        name="carType">
 <xs:restriction       base="xs:string">
  <xs:enumeration    value="Audi"/>
  <xs:enumeration    value="Golf"/>
  <xs:enumeration    value="BMW"/>
 </xs:restriction>
</xs:simpleType>
```

Restrictions on series of values:

- To limit the content of an XML element to define a series of numbers or letters that can be used, we would use the pattern constraint.
- The example below defines an element called "letter" with a restriction. The only acceptable value is ONE of the LOWERCASE letters from a to z:

```
<xs:element name="letter">
 <xs:simpleType>
  <xs:restriction      base="xs:string">
   <xs:pattern         value="[a-z]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

- The next example defines an element called "prodid" with a restriction. The only acceptable value is FIVE digits in a sequence, and each digit must be in a range from 0 to 9:

```
<xs:element name="prodid">
 <xs:simpleType>
  <xs:restriction      base="xs:integer">
   <xs:pattern         value="[0-9][0-9][0-9][0-9][0-9]"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Restrictions on whitespace characters:

- To specify how whitespace characters should be handled, we would use the whiteSpace constraint.
- This example defines an element called "address" with a restriction. The whiteSpace constraint is set to "preserve", which means that the XML processor WILL NOT remove any white space characters:

```
<xs:element name="address">
 <xs:simpleType>
  <xs:restriction      base="xs:string">
   <xs:whiteSpace      value="preserve"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

Restrictions on length:

- To limit the length of a value in an element, we would use the length, maxLength, and minLength constraints.
- This example defines an element called "password" with a restriction. The value must be exactly eight characters:

```
<xs:element name="password">
 <xs:simpleType>
  <xs:restriction      base="xs:string">
   <xs:length          value="8"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

ii) Complex type elements:

- Complex type elements contain child elements, attributes, and text.
- They can also contain mixed type of content.
- A complex XML element, "employee", which contains only other elements:

```
<employee>
    <firstname>John</firstname>
    <lastname>Smith</lastname>
</employee>
```

- There are four kinds of complex elements:
  a) empty elements
  b) elements only
  c) Text only
  d) Mixed content

a) empty elements:

- An empty complex element cannot have contents, only attributes.
- An empty XML element: **<product prodid="1345" />**

b) elements only:

- An "elements-only" complex type contains an element that contains only child elements.
- An XML element, "person", that contains only other elements:

```
<person>
 <firstname>John</firstname>
 <lastname>Smith</lastname>
</person>
```

- You can define the "person" element in a schema, like this:

```
<xs:element name="person">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="firstname" type="xs:string"/>
   <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

- Notice the <xs:sequence> tag. It means that the elements defined ("firstname" and "lastname") must appear in that order inside a "person" element.

c) Text only:

- A complex text-only element can contain text and attributes.

```
<xs:element    name="somename">
 <xs:complexType>
  <xs:simpleContent>
   <xs:extension       base="basetype">
    ....
    ....
   </xs:extension>
  </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

d) Mixed content

- A mixed complex type element can contain attributes, elements, and text.
- An XML element, "letter", that contains both text and other elements:

```
<letter>
        Dear Mr. <name>John Smith</name>.
        Your order <orderid>1032</orderid>
        will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

- The following schema declares the "letter" element:

```
<xs:element    name="letter">
  <xs:complexType     mixed="true">
    <xs:sequence>
      <xs:element        name="name"         type="xs:string"/>
      <xs:element        name="orderid"      type="xs:positiveInteger"/>
      <xs:element        name="shipdate"     type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Example: Create an XML to store shipping order details of item with XSD validation.

XML document called "shiporder.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<shiporder   orderid="889923"   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                    xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
        <name>Ola Nordmann</name>
         <address>Langgt 23</address>
        <city>4000 Stavanger</city>
        <country>Norway</country>
  </shipto>
  <item>
        <title>Empire Burlesque</title>
        <note>Special Edition</note>
        <quantity>1</quantity>
        <price>10.90</price>
  </item>
  <item>
        <title>Hide your heart</title>
        <quantity>1</quantity>
        <price>9.90</price>
  </item>
</shiporder>
```

XML schema file called "shiporder.xsd":

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

<xs:element name="shiporder">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="orderperson" type="xs:string"/>
   <xs:element name="shipto">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="address" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string" minOccurs="0"/>
      <xs:element name="quantity" type="xs:positiveInteger"/>
      <xs:element name="price" type="xs:decimal"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="orderid" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>

</xs:schema>
```

Describing the data types of XML Schema:

- The XML schema contains some predefined element types.
- These predefined element types also called data types of XML schema.
- These predefined element types categorized into 4 types.
  - String
  - Date and Time
  - Decimal
  - Miscellaneous types

String data type:

- String data types are used for a value that contains character strings.
- The string data type can contain characters, line feeds, carriage returns, and tab characters.
- Example:

  XML element: **<customer>John Smith</customer>**
  XSD: **<xs:element name="customer" type="xs:string"/>**
- The list of string data types available as follows.

Name	Description
ID	A string that represents the ID attribute in XML (only used with schema attributes)
IDREF	A string that represents the IDREF attribute in XML (only used with schema attributes)
language	A string that contains a valid language id
Name	A string that contains a valid XML name
NMTOKEN	A string that represents the NMTOKEN attribute in XML (only used with schema attributes)
normalizedString	A string that does not contain line feeds, carriage returns, or tabs
token	A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces

Date and Time data type:

- Date and time data types are used for values that contain date and time.
- The **date data type** is used to specify a date.
- The date is specified in the format of "YYYY-MM-DD".
- Where YYYY indicates the year, MM indicates the month, and DD indicates the day
- An element: **<start>2002-09-24</start>**
- Example: **<xs:element name="start" type="xs:date"/>**

- The **time data type** is used to specify a time.
- The time is specified in the format of "hh:mm:ss.
- Where hh indicates the hour, mm indicates the minute, and ss indicates the second
- An element: **<start>09:00:00</start>**
- Example: **<xs:element name="start" type="xs:time"/>**

- The **dateTime data type** is used to specify a date and a time.
- The dateTime is specified in format of "YYYY-MM-DDThh:mm:ss".
- Where, YYYY indicates the year, MM indicates the month, DD indicates the day, T indicates the start of the required time section, h indicates the hour, mm indicates the minute, and ss indicates the second
- An element: **<startdate>2002-05-30T09:00:00</startdate>**
- Example: **<xs:element name="startdate" type="xs:dateTime"/>**
- **Note:** All components are required!

Decimal data type:

- The **decimal data type** is used to specify a numeric value.
- An element: **<price>999.50</price>**
- Example: **<xs:element name="price" type="xs:decimal"/>**
- The **integer data type** is used to specify a numeric value without a fractional component.
- An element: **<price>999</price>**
- Example: **<xs:element name="price" type="xs:integer"/>**
- The list of decimal data types available as follows.

Name	Description
byte	A signed 8-bit integer
decimal	A decimal value
int	A signed 32-bit integer
integer	An integer value
long	A signed 64-bit integer
negativeInteger	An integer containing only negative values (..,-2,-1)
nonNegativeInteger	An integer containing only non-negative values (0,1,2,..)
nonPositiveInteger	An integer containing only non-positive values (..,-2,-1,0)
positiveInteger	An integer containing only positive values (1,2,..)
short	A signed 16-bit integer
unsignedLong	An unsigned 64-bit integer
unsignedInt	An unsigned 32-bit integer
unsignedShort	An unsigned 16-bit integer
unsignedByte	An unsigned 8-bit integer

Miscellaneous data type:

- Miscellaneous data types are boolean, and anyURI.
- An element: **<price disabled="true">999</price>**
- Example: **<xs:attribute name="disabled" type="xs:boolean"/>**

- The anyURI data type is used to specify a URI.
- An element: **<pic src="https://www.google.com/images/smiley.gif" />**
- Example: **<xs:attribute name="src" type="xs:anyURI"/>**
- The list of miscellaneous data types available as follows.

Name	Description
anyURI	Specifies the URI.
boolean	Specifies the true or false value.

The important differences between DTD and XSD are given below:

No	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.
8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

Working with XSLT

- XSLT stands for Extensible Stylesheet Language Transformations.
- XSLT is an XML based language that allows you to transform the XML documents into other formats, such as HTML, XHTML, or PDF.
- XSLT is a declarative language i.e., it focus on what the program should do rather than how to perform.
- In transformation process, XSLT uses the **XPath language** to define parts of the source document that should be matched with one or more predefined templates.
- When a match is found, the XSLT language transforms the matching part of the source document in the result document.
- To use XSLT language, you need create an **.xsl file** that represents a stylesheet for XML document.
- This stylesheet should contain either the **xsl:stylesheet** element or the **xsl:transform** element to transform files of one format into another.
- You can declare a stylesheet by using either of the following syntaxes.

<xsl:stylesheet>
:
</xsl:stylesheet>

<xsl:transform>
:
</xsl:transform>

- Attributes are

Attribute	Value	Description
version (optional)	version	Specifies the XSLT version of the stylesheet.
Extension-element-prefix (optional)	list	Represents a white space separated list of namespace prefix used for extension elements.
Excluded-result-prefix (optional)	list	Represents a white space separated list of namespace prefix that should not be sent to the output.
id (optional)	name	Represents the unique id for the stylesheet

- **XSLT elements are as follow**

Element	Description
apply-imports	Applies a template rule from an imported style sheet
apply-templates	**Applies a template rule to the current element or to the current element's child nodes**
attribute	**Adds an attribute**
attribute-set	**Defines a named set of attributes**
call-template	**Calls a named template**
choose	Used in conjunction with <when> and <otherwise> to express multiple conditional tests
comment	Creates a comment node in the result tree
copy	Creates a copy of the current node (without child nodes and attributes)
copy-of	Creates a copy of the current node (with child nodes and attributes)
decimal-format	Defines the characters and symbols to be used when converting numbers into strings, with the format-number() function
element	**Creates an element node in the output document**
fallback	Specifies an alternate code to run if the processor does not support an XSLT element
for-each	Loops through each node in a specified node set
if	Contains a template that will be applied only if a specified condition is true
import	**Imports the contents of one style sheet into another. Note: An imported style sheet has lower precedence than the importing style sheet**
include	**Includes the contents of one style sheet into another. Note: An included style sheet has the same precedence as the including style sheet**

key	Declares a named key that can be used in the style sheet with the key() function
message	Writes a message to the output (used to report errors)
namespace-alias	Replaces a namespace in the style sheet to a different namespace in the output
number	Determines the integer position of the current node and formats a number
otherwise	Specifies a default action for the <choose> element
output	Defines the format of the output document
param	Declares a local or global parameter
preserve-space	Defines the elements for which white space should be preserved
processing-instruction	Writes a processing instruction to the output
sort	Sorts the output
strip-space	Defines the elements for which white space should be removed
stylesheet	Defines the root element of a style sheet
template	**Rules to apply when a specified node is matched**
text	Writes literal text to the output
transform	Defines the root element of a style sheet
value-of	**Extracts the value of a selected node**
variable	Declares a local or global variable
when	Specifies an action for the <choose> element
with-param	Defines the value of a parameter to be passed into a template

xsl:template Element:

- The xsl:template element defines a template are set of rules to produce output.
- It is a top-level element and uses a match attribute for defining different names and patterns for nodes.
- Syntax: <xsl:template name = "name" match = "pattern" mode = "mode" priority = "number">
  <!- content -->
  </xsl:template>
- Attributes of the xsl:template as follow

Attribute	Description
name	Name of the element on which template is to be applied.
match	Pattern which signifies the element(s) on which template is to be applied.
priority	Priority number of a template. Matching template with low priority is not considered in from in front of high priority template.
mode	Allows element to be processed multiple times to produce a different result each time.

xsl:apply-templates Element:

- The xsl:apply-templates element defines a set of nodes to be processed by selecting an appropriate template rule.
- The <xsl:apply-templates> element applies a template rule to the current element or to the current element's child nodes.
- The element selects a set of nodes and processed each node by finding a matching template.
- If xsl:sort element is nested with in xsl:applt-templates element, then it determines the order in which the nodes are processed.
- Syntax: <xsl:apply-templates select = "expression" mode = "mode">
  <!--content -->
  </xsl:apply-templates>

- Attributes of the xsl:apply-templates as follow

Attribute	Description
select	Used to process nodes selected by an XPath expression, instead of processing all the children.
mode	Allows an element as specified by its Qualified Names to be processed multiple times, each time producing a different result.

xsl:import Element:

- The <xsl:import> element is a top-level element that is used to import the contents of one style sheet into another.
- An imported style sheet has lower precedence than the importing style sheet.
- Syntax: **<xsl:import href="URI"/>**
- Attributes are as follow

Attribute	Value	Description
href	URI	Required. Specifies the URI of the style sheet to import

xsl:call-template Element:

- The <xsl:call-template> element calls a named template.
- The xsl:template does not work alone itself. It is manually called by using the xsl:call-template element.
- The name attribute of the xsl:call-template and xsl:template element must be same.
- Xsl:call-template element may have one or more nested xsl:with-param elements.
- Syntax:

```
<xsl:call-template name="templatename" >
    <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

- Attributes are as follow

Attribute	Value	Description
name	templatename	Required. Specifies the name of the template to be called

xsl:include Element:

- The <xsl:include> element is a top-level element that includes the contents of one style sheet into another.
- An included style sheet has the same precedence as the including style sheet.
- This element must appear as a child node of <xsl:stylesheet> or <xsl:transform>.
- Syntax: <xsl:include href="URI"/>
- Attributes

Attribute	Value	Description
href	URI	Required. Specifies the URI of the style sheet to include

xsl:element Element:

- The <xsl:element> element is used to create an element node in the output document.
- Syntax:

```
<xsl:element name="name" namespace="URI" use-attribute-sets="namelist">
    <!-- Content:template -->
</xsl:element>
```

- Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of the element to be created (the value of the name attribute can be set to an expression that is computed at run-time, like this: <xsl:element name="{$country}" />
namespace	URI	Optional. Specifies the namespace URI of the element (the value of the

		namespace attribute can be set to an expression that is computed at run-time, like this: <xsl:element name="{$country}" namespace="{$someuri}"/>
use-attribute-sets	nameli st	Optional. A white space separated list of attribute-sets containing attributes to be added to the element

xsl:attribute Element:

- The <xsl:attribute> element is used to add attributes to elements.
- Note: The <xsl:attribute> element replaces existing attributes with equivalent names.
- Syntax:
```
<xsl:attribute name="attributename" namespace="uri">
    <!-- Content:template -->
</xsl:attribute>
```
- Attributes

Attribute	Value	Description
name	attributename	Required. Specifies the name of the attribute
namespace	URI	Optional. Defines the namespace URI for the attribute

- **Example 1: Add a source attribute to the picture element:**
```
<picture>
    <xsl:attribute name="source"/>
</picture>
```

xsl:attribute-set Element:

- The <xsl:attribute-set> element creates a named set of attributes. The attribute-set can be applied as whole to the output document.
- **Note:** Must be child of <xsl:stylesheet> or <xsl:transform>.
- Syntax:
```
<xsl:attribute-set name="name" use-attribute-sets="name-list">
    <!-- Content:xsl:attribute* -->
</xsl:attribute-set>
```
- Attributes

Attribute	Value	Description
name	name	Required. Specifies the name of the attribute-set
use-attribute-sets	name-list	Optional. A white space separated list of other attribute-sets to use in the attribute-set

xsl:value-of Element:

- The XSLT <xsl:value-of> element is used to extract the value of selected node. It puts the value of selected node as per XPath expression, as text.
- Syntax: **<xsl:value-of** select = Expression disable-output-escaping = "yes" | "no"**>**
  </xsl:value-of>
- Attributes

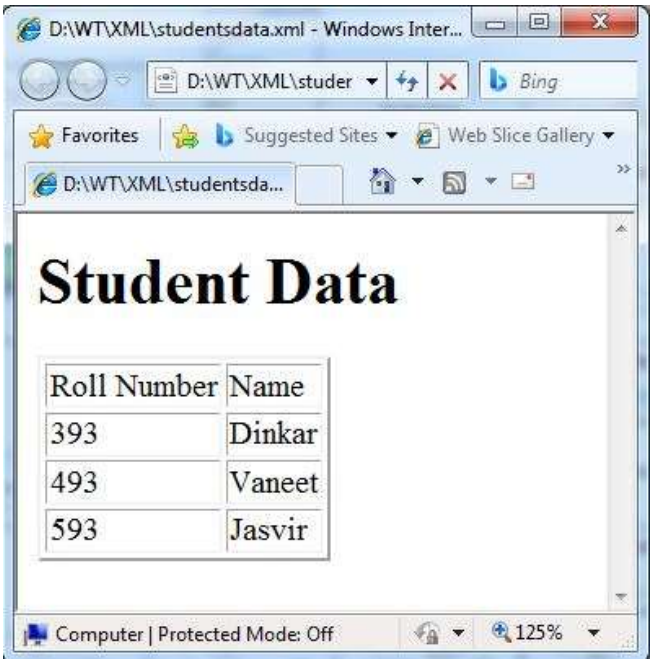
Index	Name	Description
1)	select	It specifies the XPpath expression to be evaluated in current context.
2)	disable-output-escaping	Default-"no". If "yes", output text will not escape XML characters from text.

Example: Create a XSL file that transforms an XML document into HTML document.

Studentdata.xml:

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "transform.xsl"?>
<class>
  <student rollno = "393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno = "493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
  </student>
  <student rollno = "593">
    <firstname>Jasvir</firstname>
    <lastname>Singh</lastname>
    <nickname>Jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

Output:



Transform.xsl:

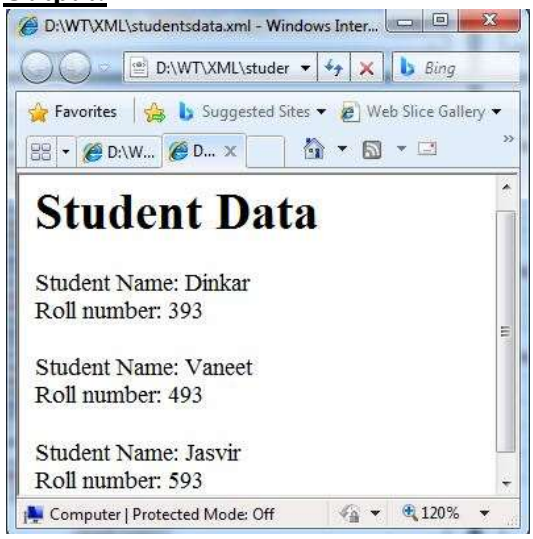
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template   match="/">
        <html>
                <body>
                        <table border="2">
                                <tr>
                                        <td>Roll Number</td>
                                        <td>Name</td>
                                </tr>
                                <xsl:for-each select="class/student">
                                <tr>
                                        <td><xsl:value-of select="@rollno"/></td>
                                        <td><xsl:value-of select="firstname"/></td>
                                </tr>
                                </xsl:for-each>
                        </table>
                </body>
        </html>
</xsl:template>
</xsl:stylesheet>
```

Example: Create a XSL file that transforms an XML document into HTML document using xsl:template and xsl:apply-tempates

Studentdata.xml:

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "apply.xsl"?>
<class>
  <student rollno = "393">
    <firstname>Dinkar</firstname>
    <lastname>Kad</lastname>
    <nickname>Dinkar</nickname>
    <marks>85</marks>
  </student>
  <student rollno = "493">
    <firstname>Vaneet</firstname>
    <lastname>Gupta</lastname>
    <nickname>Vinni</nickname>
    <marks>95</marks>
  </student>
  <student rollno = "593">
    <firstname>Jasvir</firstname>
    <lastname>Singh</lastname>
    <nickname>Jazz</nickname>
    <marks>90</marks>
  </student>
</class>
```

Output:



apply.xsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
        <html>
                <body>
                        <h1>Student Data</h1>
                        <xsl:apply-templates/>
                </body>
        </html>
        </xsl:template>
        <xsl:template match="class/student">
        <p>
                <xsl:apply-templates select="firstname"/>
                <xsl:apply-templates select="@rollno"/>
        </p>
        </xsl:template>
        <xsl:template match="firstname">
                Student Name:
                <xsl:value-of select="."/><br />
        </xsl:template>
        <xsl:template match="@rollno">
                Roll number:
                <xsl:value-of select="."/><br />
        </xsl:template>
</xsl:stylesheet>
```

AJAX: Overview of AJAX

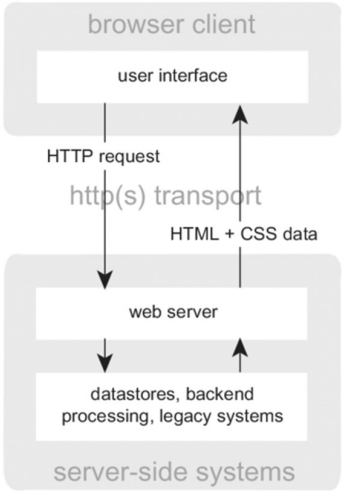
- AJAX (**A**synchronous **J**avaScript **A**nd **X**ML) is a collection of web technologies to develop more interactive web applications or web pages.
- AJAX features are
  - ✓ Update a web page without reloading the page
  - ✓ Request data from a server - after the page has loaded
  - ✓ Receive data from a server - after the page has loaded
  - ✓ Send data to a server - in the background

Different Web Technologies:

- Common Gate Way Interface (CGI)
- Applet
- JavaScript
- Servlet
- Java Server Pages (JSP)
- Active Server Page (ASP)
- Hypertext Preprocessor (PHP)
- DHTML
- XML

Problem with Classical Technologies:

- The entire classical web application model adopts **start-stop-start-stop** approach to establish communication between client and server.
- In traditional model, the browser responds to the user action by discarding the current HTML page.
- Then the request is sent back to the web server and when the server completes the processing of request, it returns the response page to the web browser.
- Finally the browser refreshes the screen and displays the new page.
- The user cannot perform any task during the entire cycle, until the entire process is completed.



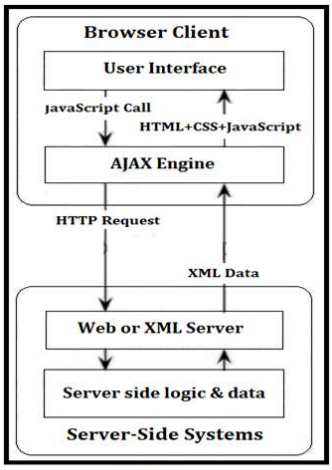
classic
web application model

Exploring AJAX:

- AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and JavaScript.
- AJAX is not a programming language but way to use existing standards such as JavaScript and XML.
- AJAX is based on some internet standards, are
  - ✓ **XMLHttpRequest:** Refers to an object that is used to exchange data asynchronously with a server.
  - ✓ **JavaScript and DOM:** Display and interact with the information.
  - ✓ **CSS:** provides styles to display the data.
  - ✓ **XML:** provides a format to transfer the data from server to a client.

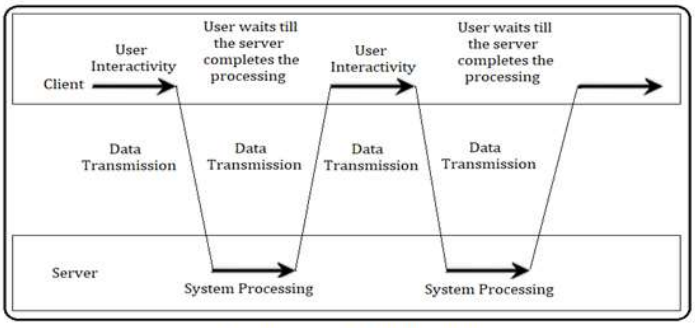
AJAX Web Application Model:

- The AJAX we application eliminates the start-stop-start-stop nature.
- At beginning it loads an ASP engine. An Asp engine code written in JavaScript and it establish the communication between a user interface and the AJAX engine on the client side.



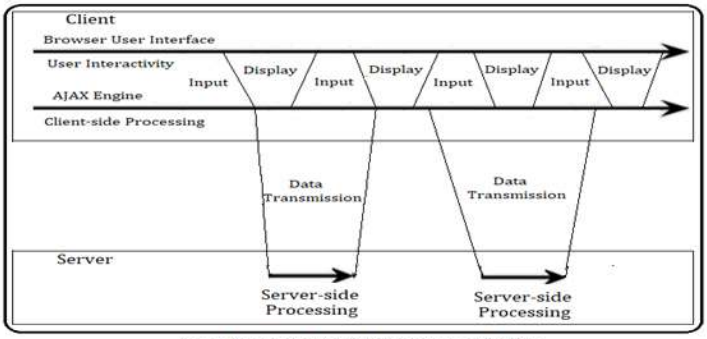
AJAX Web Application Model

- A web page sends its request to the server using JavaScript function.
- The server response contains data and not style. The style on the data was implemented by using markup language.
- Most of the page does not change only some parts of the page that need to change are updated.
- JavaScript dynamically updates the Webpage, without reloading the entire Web page.
- This prevents the user from waiting for the server to complete its processing.
- The AJAX engine takes care of displaying the user interface and interacts with the server on the user's behalf.
- In traditional web application, a synchronous mode of communication exists between the client and the server as shown below.



Synchronous Mode of Communication

- In AJAX, an asynchronous mode of communication exists between the client and the server as shown below.



Asynchronous Mode of Communication

- It is seen that in the synchronous mode of interaction, there is no scope for user to wait until the server side processing gets over.

How AJAX works:

- Following steps are involved in the working of AJAX:

Step 1:

- Creating an instance of XMLHttpRequest object to send HttpRequest from client to server.
- Syntax: **variable = new XMLHttpRequest();**
- Example: **req = new XMLHttpRequest();**

Step 2:

- Creating a request to the server by using the open( ) method of the XMLHttpRequest object.
- Syntax: open**(method, URL, async);**
- Example: **req.open(GET, "file.asp", true);**
- Here, **method** specifies the type of request (GET, POST),
  URL specifies the location of file on the server,
  async specifies that the request is handled or not. The true indicate asynchronous and false indicates synchronous.

Step 3:

- Sending a request to the server by using send( ) method of the XMLHttpRequest object.
- It having two metods send( ) and send(string) for sending request.
- If request types is GET then use send( ) constructor.
- Syntax: open**(method, URL, async);**
  send( )
- Example: **req.open(GET, "file.asp", true);**
  send( );
- If request type is POST the use send(string) constructor.
- Syntax: open**(method, URL, async);**
  send(string);
- Example: **req.open(GET, "file.asp", true);**
  send(uname=appe&password=12345);

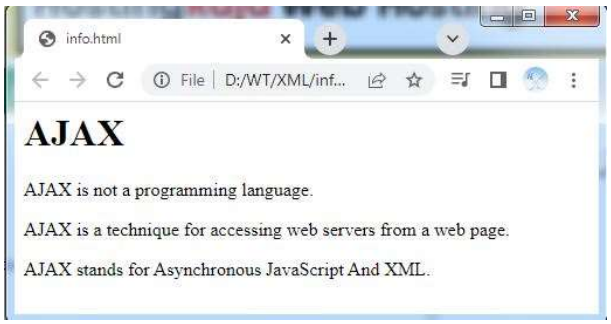
Example: Create a HTML file that demonstrates the AJAX.

```
<!DOCTYPE html>
<html>
<head>
	<title>Ajax Page</title>
	<script type="text/javascript">
		function loadDoc() {
			xhttp = new XMLHttpRequest();
			xhttp.onreadystatechange = function()
			{
				if(this.readyState == 4 && this.Status == 200)
				{
					document.getElementById("demo").innerHTML = this.responseText;
				}
			};
			xhttp.open("GET", "info.html", true);
			xhttp.send(null);
		}
	</script>
	</head>
	<body>
		<div id="demo">
			<h1>The XMLHttpRequest Object</h1>
			<button type="button" onclick="loadDoc()">Change Content</button>
		</div>
	</body>
</html>
```

Info.html:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

Output:



Asynchronous Data Transfer with XMLHttpRequest

- An object of XMLHttpRequest is used for asynchronous communication between client and server.
- It performs following operations:
  1. Sends data from the client in the background
  2. Receives the data from the server
  3. Updates the webpage without reloading it.

Creating the XMLHttpRequest beject:

- The XMLHttpRequest object creation varies from browser to browser.
- Most of the browser uses the following syntax:

  var httpreq = new XMLHttpRequest( );
- If web browser is IE5 or IE6, then the following syntax is used

  var httpreq = new ActiveXobject("Microsoft.XMLHTTP");
- The synchronous request of XMLHttpRequest object works in the following pattern
  1. Creating the XMLHttpRequest object
  2. Creating a request
  3. Sends the request
  4. Holds the processing until the response is received.

  Example:

```
var xhttp = new XMLHttpRequest();
xhttp.open("GET", " info.html", false);
xhttp.send(null);
var res = xhttp.responseXML;
```

- The asynchronous request of XMLHttpRequest object works in the following pattern
  1. Creating the XMLHttpRequest object
  2. Set a readystatechange event to trigger a specific function.
  3. Checks the readyState property to see if the data is ready. If data is not ready, then recheck it after a particular interval.
  4. Open the request
  5. Sends the request
  6. Continues the processing until the response is received.

  Example:

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = asyncHeader;
xhttp.open("GET", " info.html", true);
xhttp.send(null);
function asyncHeader() {
        if(xhttp.readyState ==4 && xhttp.status == 200)
                var res = xhttp.responseText;
}
```

XMLHttpRequest Object Methods

Method	Description
new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
open(*method,url,async,user,psw*)	Specifies the request *method*: the request type GET or POST *url*: the file location *async*: true (asynchronous) or false (synchronous) *user*: optional user name *psw*: optional password
send()	Sends the request to the server Used for GET requests
send(*string*)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent
getResponseHeader()	Returns specific header information
getAllResponseHeaders()	Returns header information

XMLHttpRequest Object Properties:

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" etc.
statusText	Returns the status-text (e.g. "OK" or "Not Found")

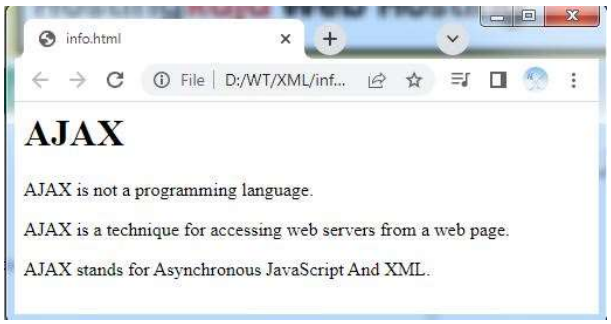
Example: Create a HTML file that demonstrates the AJAX.

```
<!DOCTYPE html>
<html>
<head>
	<title>Ajax Page</title>
	<script type="text/javascript">
		function loadDoc() {
			xhttp = new XMLHttpRequest();
			xhttp.onreadystatechange = function()
			{
				if(this.readyState == 4 && this.Status == 200)
				{
					document.getElementById("demo").innerHTML = this.responseText;
				}
			};
			xhttp.open("GET", "info.html", true);
			xhttp.send(null);
		}
	</script>
	</head>
	<body>
		<div id="demo">
			<h1>The XMLHttpRequest Object</h1>
			<button type="button" onclick="loadDoc()">Change Content</button>
		</div>
	</body>
</html>
```

Info.html:

```
<h1>AJAX</h1>
<p>AJAX is not a programming language.</p>
<p>AJAX is a technique for accessing web servers from a web page.</p>
<p>AJAX stands for Asynchronous JavaScript And XML.</p>
```

Output:



Working with jQuery

- jQuery is a lightweight, "write less, do more", JavaScript library.
- The purpose of jQuery is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.
- jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
- The jQuery library contains the following features:
  - HTML/DOM manipulation
  - CSS manipulation
  - HTML event methods
  - Effects and animations
  - AJAX
  - Utilities

Loading and Using JQuery:

- **Download and include the jQuery library**
- The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag.

```
<head>
    <script type=”text/javascript” src="jquery-3.6.0.min.js"></script>
</head>
```

- If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).
- Google is an example of someone who host jQuery:

```
<head>
        <script type=”text/javascript”
        src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
</head>
```

- **The Document Ready Event**
- Now, jQuery reads or manipulates a DOM document by using the **ready event** of the document:
- Syntax:

```
$(document) . ready ( function(){
        // jQuery methods go here...
});
```

- This is to prevent any jQuery code from running before the document is finished loading (is ready).
- It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.
- The jQuery team has also created an even shorter method for the document ready event:
- Syntax:

```
$( function(){
        // jQuery methods go here...
});
```

Example: **Create an HTML file that demonstrate the jQuery.**

```
<html>
<head>
        <title>Ajax Page</title>
        <script type="text/javascript" src="jquery-3.6.0.min.js"></script>
        <script type="text/javascript">
                $(document).ready(function(){
                        $("h1").css("color","green");
                });
        </script>
        </head>
        <body>
                <div id="demo">
                        <h1>jQuery Example</h1>
                </div>
        </body>
</html>
```

Output:

jQuery Selectors:

- jQuery selectors allow you to select and manipulate HTML element(s).
- jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more.
- All selectors in jQuery start with the dollar sign and parentheses: $().

Syntax	Description
$("*")	Selects all elements
$(“p”)	Selects all <p> elements
$(“#test”)	Selects the HTML element that match with id=”test”
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first element of the first
$("ul li:first-child")	Selects the first element of every
$("[href]")	Selects all elements with an href attribute
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"
$(":submit")	Selects all <button> elements and <input> elements of type="submit"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

jQuery Event:

- The jQuery library provides methods to handle DOM events.
- Most jQuery methods correspond to native DOM events.

Event Handling

- To handle DOM events using jQuery methods, first get the reference of DOM element(s) using jQuery selector and invoke appropriate jQuery event method.
- Example: Handle Button Click Event

```
$('button') . click ( function ( ) {
    alert(' Save button clicked' );
});
```

- The following table lists all jQuery methods and corresponding DOM events.

Category	jQuery Event	Description
Form events	blur	Specifies an event that occurs when an element looses focus
	change	Specifies an event that occurs when an elemnt changes.
	focus	Specifies an event that occurs when an element gets focused
	focusin	Specifies an event that occurs when an element is focused.
	select	Specifies an event that occurs when a text is selected.
	submit	Specifies an event that occurs when a form is submitted.
Keyboard events	keydown	Specifies an event that occurs when a key pressed.
	keypress	Specifies an event that occurs when a key pressed and key released.
	keyup	Specifies an event that occurs when a key released.
	focusout	Specifies an event that occurs when an element is focus out or blur.
Mouse events	click	Specifies an event that occurs when a mouse button click.
	dblclick	Specifies an event that occurs when a mouse button double clicked.
	focusout	Specifies an event that occurs when an element is focus out or blur.
	hover	Specifies an event that occurs when a mouse entered and released.
	mousedown	Specifies an event that occurs when mouse button pressed.
	mouseup	Specifies an event that occurs when mouse button released.
	mouseenter	Specifies an event that occurs when mouse is entered into element region.
	mouseleave	Specifies an event that occurs when mouse is leaves into element region.
	mousemove	Specifies an event that occurs when mouse pointer move.
	mouseover	Specifies an event that occurs when mouse pointer move over of element.
	mouseout	Specifies an event that occurs when mouse pointer move out of an element.
	toggle	Specifies an event that occurs toggle between hide( ) and show( ) methods.
Browser events	error	Specifies an event that occurs when browser raise an error.
	resize	Specifies an event that occurs when a resize the browser.
	scroll	Specifies an event that occurs when scroll the window.
Document loading	load	Specifies an event that occurs when documents are loaded
	ready	Specifies an event that occurs when documents are ready
	unload	Specifies an event that occurs when documents are unloaded.

Exploring jQuery methods:

- jQuery methods are categorized into following ways:

1. Methods for Access the HTML attributes:

- jQuery enable to read, add, modify, or remove an attribute from an HTML.

Method	Description
attr(properties)	Sets or returns attributes/values of selected elements
attr(key, fn)	Sets a single property to a computed value for all matched elements.
removeAttr(name)	Removes one or more attributes from selected elements
addClass(classes)	Adds one or more class names to selected elements
removeClass(class)	Removes one or more classes from selected elements
html()	Sets or returns the content of selected elements
text()	Sets or returns the text of the selected elements
val()	Sets or returns the value attribute of the selected elements

2. Methods for Manipulate CSS properties:

- jQuery provides various methods to modify the CSS properties.

Method	Description
css()	Sets or returns one or more style properties for selected elements
height()	Sets or returns the height of selected elements
position()	Returns the position (relative to the parent element) of an element
scrollLeft()	Sets or returns the horizontal scrollbar position of selected elements
scrollTop()	Sets or returns the vertical scrollbar position of selected elements
width()	Sets or returns the width of selected elements

3. Methods for Manipulate HTML elements:

- jQuery provides various methods to modify or manipulate the content of HTML elements and attribute.

Method	Description
append()	Inserts content at the end of selected elements
prepend()	Inserts content at the beginning of selected elements
after()	Inserts content after selected elements
before()	Inserts content before selected elements
clone()	Makes a copy of selected elements
detach()	Removes selected elements (keeps data and events)
empty()	Removes all child nodes and content from selected elements
hasClass()	Checks if any of the selected elements have a specified class name
insertAfter()	Inserts HTML elements after selected elements
insertBefore()	Inserts HTML elements before selected elements
remove()	Removes the selected elements (including data and events)
replaceAll()	Replaces selected elements with new HTML elements
replaceWith()	Replaces selected elements with new content

4. Traversing methods:

- jQuery is provides a variety of DOM traversal methods to help to select elements in an HTML or XML document randomly as well as in sequential method.
- Elements in the DOM are organized into a tree-like data structure that can be traversed to navigate, locate the content within an HTML or XML document.

Method	Description
eq(index)	Returns an element with a specific index number of the selected elements
filter(selector)	Reduce the set of matched elements to those that match the selector or pass the function's test
is(selector)	Checks the set of matched elements against a selector/element/jQuery object, and return true if at least one of these elements matches the given arguments
not(selector)	Returns elements that do not match a certain criteria
add(selector)	Adds elements to the set of matched elements
children([selector])	Returns all direct children of the selected element
contents()	Returns all direct children of the selected element (including text and comment nodes)
find(selector)	Returns descendant elements of the selected element
next([selector])	Returns the next sibling element of the selected element
nextAll([selector])	Returns all next sibling elements of the selected element
parent([selector])	Returns the direct parent element of the selected element
prev([selector])	Returns the previous sibling element of the selected element
prevAll([selector])	Returns all previous sibling elements of the selected element
siblings([selector])	Returns all sibling elements of the selected element

jQuery Effect Methods

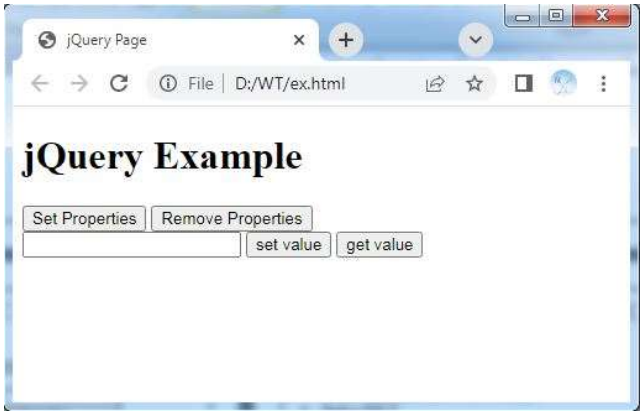
- The following table lists all the jQuery methods for creating animation effects.

Method	Description
animate( )	Runs a custom animation on the selected elements
fadeIn( )	Fades in the selected elements
fadeOut( )	Fades out the selected elements
fadeTo( )	Fades in/out the selected elements to a given opacity
hide( )	Hides the selected elements
show( )	Shows the selected elements
stop( )	Stops the currently running animation for the selected elements
delay( )	Sets a delay for all queued functions on the selected elements
toggle( )	Toggles between the hide() and show() methods

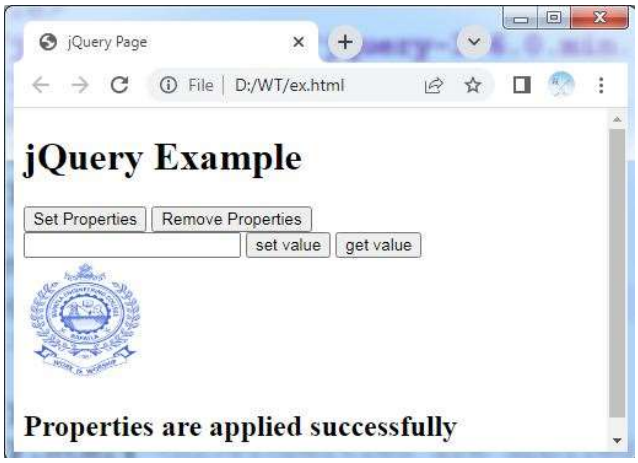
Example: Write a jQuery to demonstrate the **Methods for Access the HTML attributes.**

```
<!DOCTYPE html>
<html>
<head>
	<title>jQuery Page</title>
		<script type="text/javascript" src="jquery-3.6.0.min.js"></script>
		<script type="text/javascript">
			$(document).ready(function(){
				$("#set").click(function(){
					$("img").attr({
						src:"images/beclogo.png",
						width:"200px",
						height:"200px",
						alt:"College Logo"
					});
					$("div").addClass("message");
					$("div").html("<h2>Properties are applied successfully</2>");
				});
				$("#remove").click(function(){
					$("img").removeAttr("src");
					$("div").removeClass("message");
					$("div").text("Properties are removed successfully");
				});
				$("#setval").click(function(){
					$("input").val("apple");
				});
				$("#getval").click(function(){
					alert("value is :"+$("input").val());
				});
			});
		</script>
	</head>
	<body>
		<h1>jQuery Example</h1>
		<button id="set">Set Properties</button>
		<button id="remove">Remove Properties</button><br/>
		<input type="text" />
		<button id="setval">set value</button>
		<button id="getval">get value</button><br/>
		<img />
		<div></div>
	</body>
</html>
```

Output:



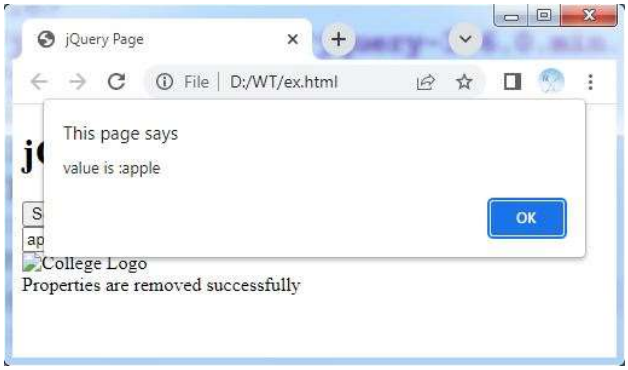
After click on "**set properties**"



After click on "**remove properties**"

After click on "**set value**"

After click on "**get value**"



jQuery AJAX Methods

- AJAX is the art of exchanging data with a server, and update parts of a web page - without reloading the whole page.
- The following table lists all the jQuery AJAX methods:

Method	Description
$.ajax()	Performs an async AJAX request
ajaxComplete()	Specifies a function to run when the AJAX request completes
ajaxError()	Specifies a function to run when the AJAX request completes with an error
ajaxSend()	Specifies a function to run before the AJAX request is sent
$.ajaxSetup()	Sets the default values for future AJAX requests
ajaxStart()	Specifies a function to run when the first AJAX request begins
ajaxStop()	Specifies a function to run when all AJAX requests have completed
ajaxSuccess()	Specifies a function to run when an AJAX request completes successfully
$.get()	Loads data from a server using an AJAX HTTP GET request
$.getJSON()	Loads JSON-encoded data from a server using a HTTP GET request
$.getScript()	Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request
load()	Loads data from a server and puts the returned data into the selected element
$.param()	Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)
$.post()	Loads data from a server using an AJAX HTTP POST request

Differences between jQuery and JavaScript:

jQuery	JavaScript
It is a javascript library.	It is a dynamic and interpreted web-development programming language.
The user only need to write the required jQuery code	The user needs to write the complete js code
It is less time-consuming.	It is more time consuming as the whole script is written.
There is no requirement for handling multi-browser compatibility issues.	Developers develop their own code for handling multi-browser compatibility.
It is required to include the URL of the jQuery library in the header of the page.	JavaScript is supportable on every browser. Any additional plugin need not to be included.
It depends on the JavaScript as it is a library of js.	jQuery is a part of javascript. Thus, the js code may or may not depend on jQuery.
It contains only a few lines of code.	The code can be complicated, as well as long.
It is quite an easy, simple, and fast approach.	It is a weakly typed programming approach.
jQuery is an optimized technique for web designing.	JavaScript is one of the popular web designing programming languages for developers that introduced jQuery.
jQuery creates DOM faster.	JavaScript is slow in creating DOM.

One mark Questions:

1. Define well-formed and valid XML document.
2. What are the features of XML?
3. Define namespace and write its syntax.
4. Define XSLT.
5. Write the difference between DTD and XSD.
6. What is XPath?
7. What is AJAX?
8. Write the features of AJAX.
9. Write about XMLHttpRequest object.
10. Write the features of jQuery.
11. Differentiate fadeIn( ) and fadeout( ) methods.

Essay Questions:

1. Write the difference between HTML and XML.
2. Define DTD. Explain internal and external DTD with an example.
3. Explain XSLT procedure with an example.
4. Write the difference between DTD and XSD.
5. Draw and explain working process of AJAX with an example.
6. List and Explain different methods in jQuery.
7. Write the differences between JavaScript and jQuery.

Programming Exercises:

1. Create a XML document to store student name,roll number, branch, and college details. Create a DTD to validate the document.
2. Create a XML document to store voterID, voter name, address, and date of birth details. Create a DTD to validate the document.
3. Create a XML document to store student name,roll number, branch, and college details. Create a XSD to validate the document.
4. Create a XML document to store voterID, voter name, address, and date of birth details. Create a XSD to validate the document.
5. Create a XSLT document to convert XML document to HTML document.
6. Create a JavaScript file that demonstrates the AJAX.
7. Create a JavaScript file that demonstrates the jQuery.