

Unit-I

HTML COMMON TAGS

What is an HTML File?

- HTML stands for Hyper Text Markup Language
- An HTML file is a text file containing small markup tags
- The markup tags tell the Web browser how to display the page
- An HTML file must have an htm or html file extension
- An HTML file can be created using a simple text editor

→Type in the following text in “NOTEPAD”:

```
<html>
<head>
<title>Title of page</title>
</head>
<body>
Hello World! This is my first homepage. <b>This text is bold</b>
</body>
</html>
```

→Save the file as "mypage.html".

Start your Internet browser. Select "Open" (or "Open Page") in the File menu of your browser. A dialog box will appear. Select "Browse" (or "Choose File") and locate the HTML file you just created - "mypage.html" - select it and click "Open". Now you should see an address in the dialog box, for example "C:\MyDocuments\webdesign\mypage.html". Click OK, and the browser will display the page.

Example Explained

The first tag in your HTML document is <html>. This tag tells your browser that this is the start of an HTML document. The last tag in your document is </html>. This tag tells your browser that this is the end of the HTML document.

The text between the <head> tag and the </head> tag is header information. Header information is not displayed in the browser window. The text between the <title> tags is the title of your document. The title is displayed in your browser's caption. The text between the <body> tags is the text that will be displayed in your browser. The text between the and tags will be displayed in a bold font.

HTML ELEMENTS

HTML documents are text files made up of HTML elements. HTML elements are defined using HTML tags.

HTML Tags

- HTML tags are used to mark-up HTML elements
- HTML tags are surrounded by the two characters < and >
- The surrounding characters are called angle brackets
- HTML tags normally come in pairs like and
- The first tag in a pair is the start tag, the second tag is the end tag
- The text between the start and end tags is the element content
- HTML tags are not case sensitive, means the same as

```
<b>This text is bold</b>
```

The purpose of the tag is to define an HTML element that should be displayed as bold.

Tag Attributes

Tags can have attributes. Attributes can provide additional information about the HTML elements on your page.

This tag defines the body element of your HTML page: <body>. With an added bgcolor attribute, you can tell the browser that the background color of your page should be red, like this:

```
<body bgcolor="red">
```

This tag defines an HTML table: <table>. With an added border attribute, you can tell the browser that the table should have no borders: <table border="0">. Attributes always come in name/value pairs like this: name="value". Attributes are always added to the start tag of an HTML element.

1. HEADINGS

Headings are defined with the <h1> to <h6> tags. <h1> defines the largest heading. <h6> defines the smallest heading.

```
<h1>This is a heading</h1>  
<h2>This is a heading</h2>  
<h3>This is a heading</h3>  
<h4>This is a heading</h4>  
<h5>This is a heading</h5>  
<h6>This is a heading</h6>
```

HTML automatically adds an extra blank line before and after a heading.

2. PARAGRAPHS

Paragraphs are defined with the <p> tag.

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

HTML automatically adds an extra blank line before and after a paragraph.

3. LINE BREAKS

The
 tag is used when you want to end a line, but don't want to start a new paragraph. The
 tag forces a line break wherever you place it.

```
<p>This <br> is a para<br>graph with line breaks</p>
```

The
 tag is an empty tag. It has no closing tag.

4. Comments in HTML

The comment tag is used to insert a comment in the HTML source code. A comment will be ignored by the browser. You can use comments to explain your code, which can help you when you edit the source code at a later date.

```
<!-- This is a comment -->
```

Note that you need an exclamation point after the opening bracket, but not before the closing bracket.

HTML TEXT FORMATTING

HTML defines a lot of elements for formatting output, like bold or italic text. How to View HTML Source? To find out, click the VIEW option in your browser's toolbar and select SOURCE or PAGE SOURCE. This will open a window that shows you the HTML code of the page.

1. Text Formatting Tags

Tag	Description
<i>	Defines italic text. Recommend using
	Defines bold text. Recommend using
	Defines emphasized text. Renders as italic text.
	Defines strong text. Renders as bold text.

Example

Source	Output
<code><i>Italic text</i>
</code>	<i>Italic text</i>
<code>Bold text
</code>	Bold text
<code>Emphasized text
</code>	<i>Emphasized text</i>
<code>Strong text
</code>	Strong text

What is the difference between `<i>` & `` and `` & ``, `` and `<i>` will both become deprecated tags. Using `` and `` are the tags that will be cross-browser compatible as browsers move forward to embrace the new standards in HTML (e.g., XHTML)

2. HTML Links

HTML uses a hyperlink to link to another document on the Web. HTML uses the `<a>` (anchor) tag to create a link to another document. An anchor can point to any resource on the Web: an HTML page, an image, a sound file, a movie, etc.

The syntax of creating an anchor:

```
<a href="url">Text to be displayed</a>
```

The `<a>` tag is used to create an anchor to link from, the href attribute is used to address the document to link to, and the words between the open and close of the anchor tag will be displayed as a hyperlink.

This anchor defines a link to W3Schools:

```
<a href="http://www.google.co.in/">Google</a>
```

The line above will look like this in a browser:

[Google](http://www.google.co.in/)

The Target Attribute

With the target attribute, you can define where the linked document will be opened.

The line below will open the document in a new browser window:

```
<a href="http://www.google.co.in/" target="_blank">Google</a>
```

The Anchor Tag and the Name Attribute

The name attribute is used to create a named anchor. When using named anchors we can create links that can jump directly into a specific section on a page, instead of letting the user scroll around to find what he/she is looking for.

Below is the syntax of a named anchor:

```
<a name="label">Text to be displayed</a>
```

The name attribute is used to create a named anchor. The name of the anchor can be any text you care to use.

The line below defines a named anchor:

```
<a name="tips">Useful Tips Section</a>
```

You should notice that a named anchor is not displayed in a special way. To link directly to the "tips" section, add a # sign and the name of the anchor to the end of a URL, like this:

```
<a href="http://www.amyschan.com/mypage.html#tips">  
Jump to the Useful Tips Section</a>
```

HTML LISTS

HTML supports ordered, unordered and definition lists.

1. Unordered Lists

An unordered list is a list of items. The list items are marked with bullets (typically small black circles).

An unordered list starts with the tag. Each list item starts with the tag.

```
<ul>  
<li>Coffee</li>  
<li>Milk</li>  
</ul>
```

Here is how it looks in a browser:

- Coffee
- Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

2. Ordered Lists

→ An ordered list is also a list of items. The list items are marked with numbers.

→ An ordered list starts with the tag. Each list item starts with the tag.

```
<ol>  
<li>Coffee</li>  
<li>Milk</li>  
</ol>
```

Here is how it looks in a browser:

1. Coffee
2. Milk

Inside a list item you can put paragraphs, line breaks, images, links, other lists, etc.

HTML TABLE

Tables are defined with the <table> tag.

A table is divided into rows (with the <tr> tag), and each row is divided into data cells (with the <td> tag). td stands for "table data," and holds the content of a data cell. A <td> tag can contain text, links, images, lists, forms, other tables, etc.

Table Example

```
<table border="1">
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How the HTML code above looks in a browser:

row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

HTML Tables and the Border Attribute

If you do not specify a border attribute, the table will be displayed without borders. Sometimes this can be useful, but most of the time, we want the borders to show.

To display a table with borders, specify the border attribute:

```
<table border="1">
<tr>
<td>Row 1, cell 1</td>
<td>Row 1, cell 2</td>
</tr>
</table>
```

HTML Table Headers

Header information in a table are defined with the <th> tag. All major browsers display the text in the <th> element as bold and centered.

```
<table border="1">
<tr>
<th>Header 1</th>
<th>Header 2</th>
</tr>
<tr>
<td>row 1, cell 1</td>
<td>row 1, cell 2</td>
</tr>
<tr>
<td>row 2, cell 1</td>
<td>row 2, cell 2</td>
</tr>
</table>
```

How the HTML code above looks in your browser:

Header 1	Header 2
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

HTML Table Tags

Tag	Description
<table>	Defines a table
<th>	Defines a header cell in a table
<tr>	Defines a row in a table
<td>	Defines a cell in a table
<caption>	Defines a table caption
<colgroup>	Specifies a group of one or more columns in a table for formatting
<col>	Specifies column properties for each column within a <colgroup> element
<thead>	Groups the header content in a table
<tbody>	Groups the body content in a table
<tfoot>	Groups the footer content in a table

HTML <table> Tag

Definition and Usage

The <table> tag defines an HTML table. An HTML table consists of the <table> element and one or more <tr>, <th>, and <td> elements. The <tr> element defines a table row, the <th> element defines a table header, and the <td> element defines a table cell. A more complex HTML table may also include <caption>, <col>, <colgroup>, <thead>, <tfoot>, and <tbody> elements.

Attributes

Attribute	Value	Description
align	left center right	Not supported in HTML5. Deprecated in HTML 4.01. Specifies the alignment of a table according to surrounding text
bgcolor	<i>rgb(x,x,x)</i> <i>#xxxxxx</i> <i>colorname</i>	Not supported in HTML5. Deprecated in HTML 4.01. Specifies the background color for a table
border	"1"	Specifies whether the table cells should have borders or not
Cellpadding	<i>pixels</i>	Not supported in HTML5. Specifies the space between the cell wall and the cell content
Cellspacing	<i>pixels</i>	Not supported in HTML5. Specifies the space between cells
frame	void above below hsides lhs rhs vsides box border	Not supported in HTML5. Specifies which parts of the outside borders that should be visible
rules	none groups rows cols all	Not supported in HTML5. Specifies which parts of the inside borders that should be visible
summary	<i>text</i>	Not supported in HTML5. Specifies a summary of the content of a table
width	<i>pixels</i> <i>%</i>	Not supported in HTML5. Specifies the width of a table

Example

```
<table border="1">
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
</table>
```



```
<td>January</td>
<td>$100</td>
</tr>
</table>
```

HTML IMAGES

With HTML you can display images in a document.

The Image Tag and the src Attribute

In HTML, images are defined with the tag. To display an image on a page, you need to use the src attribute. Src stands for "source". The value of the src attribute is the URL of the image you want to display on your page.

```
The syntax of defining an image: 
```

The URL points to the location where the image is stored. An image named "boat.gif" located in the directory "images" on "www.w3schools.com" has the URL: <http://www.w3schools.com/images/boat.gif>. The browser puts the image where the image tag occurs in the document. If you put an image tag between two paragraphs, the browser shows the first paragraph, then the image, and then the second paragraph.

The Alt Attribute

The alt attribute is used to define an "alternate text" for an image. The value of the alt attribute is an author-defined text:

```

```

The "alt" attribute tells the reader what he or she is missing on a page if the browser can't load images. The browser will then display the alternate text instead of the image. It is a good practice to include the "alt" attribute for each image on a page, to improve the display and usefulness of your document for people who have text-only browsers.

HTML BACKGROUNDS

A good background can make a Web site look really great.

Backgrounds

The <body> tag has two attributes where you can specify backgrounds. The background can be a color or an image.

1. Bgcolor

The bgcolor attribute specifies a background-color for an HTML page. The value of this attribute can be a hexadecimal number, an RGB value, or a color name:

```
<body bgcolor="#000000">
<body bgcolor="rgb(0,0,0)">
<body bgcolor="black">
```

The lines above all set the background-color to black.

2. Background

The background attribute specifies a background-image for an HTML page. The value of this attribute is the URL of the image you want to use. If the image is smaller than the browser window, the image will repeat itself until it fills the entire browser window.

```
<body background="images/cloudswirl.jpg">
<body background="http://www.amyschan.com/images/cloudswirl.jpg">
```

FORMS

Forms can be used to collect information. Sophisticated forms may be linked to a database to store the information, but this is beyond the scope of the article. This article will show you how to create a form that emails you the information.

Here is a very basic email form:

```
<form action="mailto:support@pixelmill.com" method="post">
Name:<br>
<input type="text" name="name" size="20"><br>
Email:<br>
<input type="text" name="email" size="20"><br>
Comment:<br>
<textarea cols="20" rows="5"></textarea><br>
<input type="submit" value="Submit"> <input type="reset" value="Reset">
</form>
```

And here's what it looks like:

Name:

Email:

Comment:

Notice that the "input" tag was used for the text fields as well as the buttons! The "input" tag is the tag you will use most often in forms. Here are different forms of the input tag:

- type="text"
- type="password" (asterisks when you type)
- type="checkbox"
- type="radio"
- type="submit" (You can change the text on the button.)

- type="reset" (You can change the text on the button.)
- type="hidden" (You can't see this because it's hidden! This allows you to send additional variables that your user can't see.)
- type="button" (You can change the text on the button.)

Below you will find a table of the various form elements and how you can use them.

FORM – attributes

action This attribute usually has a link to a web page that contains code which processes the form. In our example above, we use "mailto" to tell the form to send the information as an email.
 ="[url]"

method This attribute specifies how the information is sent. With the "get" method, all the information is sent in one long URL string. With the "post" method, all the information is sent but is "invisible" to the user.
 ="get"
 ="post"

name When creating complex forms and using scripting, you may need to specify the name of name of the form. The name should be unique - that is, you shouldn't have other forms on the page with the same form name.
 ="[name form]"

INPUT – attributes

type The "type" attribute allows you to specify what type of form element you want to use. As you can see, the "input" element can have many different forms - see above for examples.
 ="text"
 ="password"
 ="checkbox"
 ="radio"
 ="submit"
 ="reset"
 ="hidden"
 ="button"

checked This attribute does not have any values. When you put this attribute into the tag for "radio" or "checkbox," they will be selected.

src Use this attribute when you have an "image" to specify the location of the image.
 ="[url]"

TEXTAREA - attributes

This form element is the multi-line text box that you often see for "comments." The opening and closing tag surround the text that will be initially displayed.

name This attribute must be set to a unique name so you know how to refer to the form element.
 ="[referring name]"

rows This attribute allows you to set the number of rows for the text area.
 ="[number]"

cols This attribute allows you to set the number of columns for the text area.
 ="[number]"

SELECT - attributes

The SELECT tag allows you to create "select boxes" or "drop down menus." It is the "outer" element, used with the OPTION tag. (See the example link below)

name This attribute must be set to a unique name so you know how to refer to the form element.
="[referring name]"

size This attribute allows you to set how many rows are visible.
="[number]"

multiple This attribute does not have any "values." When you set this attribute, it allows the user to select more than one option.

OPTION - attributes

The OPTION tag is used with the SELECT tag to define the different options in the select box or dropdown menu. The opening and closing tag surround the text that will be displayed.

value This attribute must be set to a unique name so you know how to refer to the form element. It will probably be an abbreviated version of the text that is displayed.
="[value]"

selected This attribute does not have any "values." When you set this attribute, it marks this option as being "preselected."

Putting it all together...

Here's a sample page using this code below:

```
<form>
Enter your Name: <input type="text" width="20" size="20">
<p>Favorite Color:<br>
<input type="checkbox" checked value="red" name="colors"> Red<br>
<input type="checkbox" value="blue" name="colors"> Blue</p>
<p>Gender:<br>
<input type="radio" checked value="male" name="gender">Male<br>
<input type="radio" value="female" name="gender">Female</p>
<p>Address:<br>
<textarea rows="5" cols="20">Please enter your permanent address
here.</textarea></p>
Country:<br>
<select name="country">
<option value="usa" selected>USA</option>
<option value="uk">United Kingdom</option>
<option value="canada">Canada</option>
</select>
```

CSS

Introduction to CSS:

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media. CSS saves a lot of work. It can control the layout of multiple web pages all at once. External style sheets are stored in “.css” files. CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

Types of CSS:

There are three types of CSS available. They are:

- 1) Inline CSS
- 2) Internal CSS
- 3) External CSS

1. Inline CSS:

If the styles are mentioned along with the tag then this type of CSS is known as inline CSS.

Example:

```
<p style="text-align: justify; background-color: red">Hi this is the Inline CSS. </p>
```

2. Internal CSS:

For internal CSS, we write all the desired “styles” for the “selectors” along with the properties and values in the “**head**” section. And in the **body** section then newly defined selector tags are used with the actual contents.

Example:

```
<html>
<head>
<style type= "text/css">
p
{
    text-align: justify;
    background-color: red;
}
</style>
</head>
<body>
<p>This is Internal CSS.</p>
</body>
</html>
```

3. External CSS:

Sometimes we need to apply particular style to more than one web page; in such cases external CSS can be used. The main idea in this type of CSS is that the desired styles can be written in one “.css” file. And this file can be called in our web pages to apply the styles.

Example:

```
<html>
<head>
<link rel = "stylesheet" type = "text/css" href = "external.css">
</head>
<body>
<p>This is Internal CSS.</p>
</body>
</html>
external.css:
p
{
    text-align: justify;
    background-color: red;
}
```

CSS Selectors:

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more.

The element Selector:

The element selector selects elements based on the element name. You can select all <p> elements on a page like this (in this case, all <p> elements will be center-aligned, with a red text color):

Example:

```
p {
    text-align: center;
    color: red;
}
```

The id Selector:

The id selector uses the id attribute of an HTML element to select a specific element. The id of an element should be unique within a page, so the id selector is used to select one unique element! To select an element with a specific id, write a hash (#) character, followed by the id of the element. The style rule below will be applied to the HTML element with id="para1":

Example:

```
#para1 {
    text-align: center;
    color: red;
}
```

The class Selector:

The class selector selects elements with a specific class attribute. To select elements with a specific class, write a period (.) character, followed by the name of the class. In the example below, all HTML elements with class="center" will be red and center-aligned:

Example:

```
.center {
  text-align: center;
  color: red;
}
```

You can also specify that only specific HTML elements should be affected by a class. In the example below, only <p> elements with class="center" will be center-aligned:

Example:

```
p.center {
  text-align: center;
  color: red;
}
```

HTML elements can also refer to more than one class. In the example below, the <p> element will be styled according to class="center" and to class="large":

Example:

```
<p class="center large">This paragraph refers to two classes.</p>
```

Controlling Page Layout:

By default, HTML <div> sections will fill 100% of the web page width, and stack up one on top of the other. It is highly unlikely that this is how we want our page elements to be arranged; chances are we want some sections arranged side by side, and using various widths. For example we might want to have a main content area with a menu down the left-hand side. Thankfully CSS makes this a simple task. The first step is to understand the float property.

The CSS float Property:

The float property tells HTML elements how to arrange themselves in relation to the other divs around them. We can specify one of two values for the float property - left or right. If we tell a <div> to float to the left, then it will shift itself as far left on the line as it can go before bumping into another <div> section. If we tell an element to float to the right it will shift as far right as it can? Let's take a look at a working example; this will give us a good visualization of how floating works. We'll begin by defining three areas for our web page:

```
<div id="header">This is the page header</div>
<div id="menu">Menu links goes in here</div>
<div id="content">And finally the page content in here</div>
```

Now, what we want to do is to have the header take up the whole of the top row, with the menu and content underneath, arranged side by side. To do this we would use the following CSS (the header section doesn't require any CSS):

```
#menu {
  float: left;
}
```

```
#content {  
    float: left;  
}
```

Using CSS to set HTML element widths:

One problem with arranging our HTML in this way is that the <div> sections will size themselves automatically to fit the text contained within them. This means that for a standard menu bar, where we have lots of text that we want to contain in a narrow area, we will end up with a very wide bar which will either squeeze the content section into a small space, or ever push it onto another line. We can fix this by specifying widths for our sections:

Example:

```
#menu {  
    float: left;  
    width: 10em;  
}
```

Widths can be specified in pixels, ems or as a percentage of the total page width.

The CSS clear property

We have seen how we can position HTML elements next to each other using CSS, but sometimes we might want an element to appear beneath other elements. We can easily force an element to sit below another using the clear property. When using clear, we can tell an element to either clear those elements which are floated left, those which are floated right, or both:

Example:

```
#divname {  
    clear: left / right / both;  
}
```

Changing Background:

The CSS background properties are used to define the background effects for elements.

CSS background properties:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

Background Color

The background-color property specifies the background color of an element. The background color of a page is set like this:

Example

```
body {  
    background-color: lightblue;  
}
```

With CSS, a color is most often specified by:

- A valid color name - like "red"
- A HEX value - like "#ff0000"
- An RGB value - like "rgb(255,0,0)"

Background Image:

The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element. The background image for a page can be set like this:

Example:

```
body {
  background-image: url("paper.gif");
}
```

Background Image - Repeat Horizontally or Vertically:

By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

Example:

```
body {
  background-image: url("gradient_bg.png");
}
```

If the image above is repeated only horizontally (`background-repeat: repeat-x;`), the background will look better:

```
body {
  background-image: url("gradient_bg.png");
  background-repeat: repeat-x;
}
```

Tip: To repeat an image vertically, set `background-repeat: repeat-y;`

Background Image - Set position and no-repeat:

Showing the background image only once is also specified by the background-repeat property:

Example:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
}
```

In the example above, the background image is shown in the same place as the text. We want to change the position of the image, so that it does not disturb the text too much. The position of the image is specified by the background-position property:

Example:

```
body {
  background-image: url("img_tree.png");
  background-repeat: no-repeat;
  background-position: right top;
}
```

Background Image - Fixed position

To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property:

Example:

```
body {  
    background-image: url("img_tree.png");  
    background-repeat: no-repeat;  
    background-position: right top;  
    background-attachment: fixed;  
}
```

Controlling borders of the HTML elements using CSS:

The CSS border properties allow you to specify the style, width, and color of an element's border. The `border-style` property specifies what kind of border to display.

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border

The `border-style` property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example:

```
p.dotted {border-style: dotted;}  
p.dashed {border-style: dashed;}  
p.solid {border-style: solid;}  
p.double {border-style: double;}  
p.groove {border-style: groove;}  
p.ridge {border-style: ridge;}  
p.inset {border-style: inset;}  
p.outset {border-style: outset;}  
p.none {border-style: none;}  
p.hidden {border-style: hidden;}  
p.mix {border-style: dotted dashed solid double;}
```

The `border-width` property specifies the width of the four borders. The width can be set as a specific size (in px, pt, cm, em, etc) or by using one of the three pre-defined values: thin, medium, or thick. The `border-width` property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example

```
p.one {
  border-style: solid;
  border-width: 5px;
}

p.two {
  border-style: solid;
  border-width: medium;
}

p.three {
  border-style: solid;
  border-width: 2px 10px 4px 20px;
}
```

The `border-color` property is used to set the color of the four borders. The color can be set by:

- name - specify a color name, like "red"
- Hex - specify a hex value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- transparent

The `border-color` property can have from one to four values (for the top border, right border, bottom border, and the left border). If `border-color` is not set, it inherits the color of the element.

Example

```
p.one {
  border-style: solid;
  border-color: red;
}

p.two {
  border-style: solid;
  border-color: green;
}

p.three {
  border-style: solid;
  border-color: red green blue yellow;
}
```

XML

WHAT IS XML?

XML stands for extensible markup language. XML is a markup language much like HTML. XML was designed to carry data, not to display data. XML tags are not predefined. You must define your own tags. XML is designed to be self-descriptive. XML is a W3C recommendation.

THE DIFFERENCE BETWEEN XML AND HTML:

XML is not a replacement for HTML. XML and HTML were designed with different goals. XML was designed to transport and store data, with focus on what data is. HTML was designed to display data, with focus on how data looks. HTML is about displaying information, while XML is about carrying information.

XML does not do anything: May be it is a little hard to understand, but XML does not do anything. XML was created to structure, store, and transport information. The following example is a note to chythu, from ravi, stored as XML:

```
<note>
<to> chythu </to>
<from> ravi </from>
<heading>reminder</heading>
<body>don't forget me this weekend!</body>
</note>
```

The note above is quite self-descriptive. It has sender and receiver information, it also has a heading and a message body. But still, this XML document does not do anything. It is just information wrapped in tags. Someone must write a piece of software to send, receive or display it.

With XML you invent your own tags: The tags in the example above (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document. That is because the XML language has no predefined tags. The tags used in HTML are predefined. HTML documents can only use tags defined in the HTML standard (like <p>, <h1>, etc.). XML allows the author to define his/her own tags and his/her own document structure. XML is not a replacement for HTML; XML is a complement to HTML. It is important to understand that XML is not a replacement for HTML. In most web applications, XML is used to transport data, while HTML is used to format and display the data.

My best description of XML is this: XML is a software- and hardware-independent tool for carrying information. XML is a W3C recommendation. XML became a W3C recommendation February 10, 1998. XML is everywhere. XML is now as important for the web as HTML was to the foundation of the web. XML is the most common tool for data transmissions between all sorts of applications.

XML separates data from HTML: If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML. With a few lines of JavaScript code, you can read an external XML file and update the data content of your web page.

Unique Features of XML:

- a) **Sharing of data:** In the real world, computer systems and databases contain data in incompatible formats. XML data is stored in plain text format. This provides a software-

and hardware-independent way of storing data. This makes it much easier to create data that can be shared by different applications.

- b) **Transporting data:** One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
- c) **Change of platform:** Upgrading to new systems (hardware or software platforms), is always time consuming. Large amounts of data must be converted and incompatible data is often lost. XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.
- d) **Availability of data:** Different applications can access your data, not only in HTML pages, but also from XML data sources. With XML, your data can be available to all kinds of "reading machines" (handheld computers, voice machines, news feeds, etc), and make it more available for blind people, or people with other disabilities.

Structure of XML:

XML documents form a tree structure that starts at "the root" and branches to "the leaves". An example of XML document is shown below. XML documents use a self-describing and simple syntax:

```
<?XML version="1.0" encoding="iso-8859-1"?>
<note>
  <to> Chythu </to>
  <from> Ravi </from>
<heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = latin-1/west European character set). The next line describes the root element of the document (like saying: "This document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

You can assume, from this example, that the XML document contains a note to Tove from Jani. Don't you agree that XML is pretty self-descriptive? XML documents form a tree structure XML documents must contain a root element. This element is "the parent" of all other elements. The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree. All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
```

```
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters). All elements can have text content and attributes (just like in html).

Example:

```
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>giada de laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">harry potter</title>
    <author>j k. rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">learning XML</title>
    <author>erik t. ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is `<bookstore>`. All `<book>` elements in the document are contained within `<bookstore>`. The `<book>` element has 4 children: `<title>`, `<author>`, `<year>`, and `<price>`.

XML syntax rules:

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use. All XML elements must have a closing tag. In HTML, Elements do not have to have a closing tag:

```
<p>this is a paragraph
<p>this is another paragraph
```

In XML, it is illegal to omit the closing tag. All elements must have a closing tag:

```
<p>this is a paragraph</p>
<p>this is another paragraph</p>
```

Note: You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.

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

```
<Message>This is incorrect</MESSAGE>
<Message>This is correct</Message>
```

Note: "Opening and closing tags" are often referred to as "start and end tags". Use whatever you prefer. It is exactly the same thing.

XML elements must be properly nested: In html, you might see improperly nested elements:

```
<b><i>this text is bold and italic</b></i> <!--This is wrong-->
```

In XML, all elements must be properly nested within each other:

```
<b><i>this text is bold and italic</i></b>
```

In the example above, "properly nested" simply means that since the <i> element is opened inside the element, it must be closed inside the element.

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

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

XML attribute values must be quoted: XML elements can have attributes in name/value pairs just like in html. In XML, the attribute values must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

- ```
<note date=12/11/2007>
 <to>tove</to>
 <from>jani</from>
</note>
```
- ```
<note date="12/11/2007">
  <to>tove</to>
  <from>jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

Entity References: Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element. This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an entity reference:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Note: only the characters "<" and "&" are strictly illegal in XML. the greater than character is legal, but it is a good habit to replace it.

Comments in XML: The syntax for writing comments in XML is similar to that of html.

```
<!-- this is a comment -->
```

white-space is preserved in XML

HTML truncates multiple white-space characters to one single white-space:

HTML: hello tove

Output: hello tove

With XML, the white-space in a document is not truncated.

XML stores new line as lf in windows applications, a new line is normally stored as a pair of characters: carriage return (cr) and line feed (lf). In UNIX applications, a new line is normally stored as an lf character. Macintosh applications also use an lf to store a new line. XML stores a new line as lf.

Basic Building Blocks of XML Documents:

All XML documents are made up the following four building blocks:

- i. Elements/Tags
- ii. Attributes
- iii. Entities
- iv. Character Data
 - a. Parsed Character Data (PCDATA)
 - b. Unparsed Character Data (CDATA)

i) XML Elements/Tags:

An XML element is everything from (including) the element's start tag to (including) the element's end tag. An element can contain: other elements text, attributes or a mix of all of the above.

```
<bookstore>
  <book category="children">
    <title>harry potter</title>
    <author>j k. rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>learning XML</title>
    <author>erik t. ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

In the example above, <bookstore> and <book> have element contents, because they contain other elements. <book> also has an attribute (category="children"). <title>, <author>, <year>, and <price> have text content because they contain text.

XML Naming Rules:

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names cannot start with a number or punctuation character
- Names cannot start with the letters XML (or XML, or XML, etc)
- Names cannot contain spaces
- Any name can be used, no words are reserved.

Best Naming Practices:

- Make names descriptive. Names with an underscore separator are nice: <first_name>, <last_name>.
- Names should be short and simple, like this: <book_title> not like this: <the_title_of_the_book>.
- Avoid "-" characters. If you name something "first-name," some software may think you want to subtract name from first.
- Avoid "." characters. If you name something "first.name," some software may think that "name" is a property of the object "first."
- Avoid ":" characters. Colons are reserved to be used for something called namespaces (more later).
- XML documents often have a corresponding database. a good practice is to use the naming rules of your database for the elements in the XML documents.
- Non-english letters like èóá are perfectly legal in XML, but watch out for problems if your software vendor doesn't support them.

XML Elements are Extensible:

XML elements can be extended to carry more information. look at the following XML example:

```
<note>
  <to>Chythu</to>
  <from>Ravi</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Let's imagine that we created an application that extracted the <to>, <from>, and <body> elements from the XML document to produce this output:

Message

to: Chythu

from: Ravi

Don't forget me this weekend!

Imagine that the author of the XML document added some extra information to it:

```
<note>
  <date>2008-01-10</date>
  <to>Chythu</to>
  <from>Ravi</from>
  <heading>reminder</heading>
  <body>don't forget me this weekend!</body>
</note>
```

Should the application break or crash?

No, the application should still be able to find the <to>, <from>, and <body> elements in the XML document and produce the same output. One of the beauties of XML is that it can be extended without breaking applications.

ii) XML Attributes:

XML elements can have attributes, just like html. Attributes provide additional information about an element. In html, attributes provide additional information about elements:

```

  <a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but can be important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

XML attributes must be quoted: Attribute values must always be quoted. Either single or double quotes can be used. For a person's sex, the person element can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='george "shotgun" ziegler'>
```

or you can use character entities:

```
<gangster name="george &quot;shotgun&quot; ziegler">
```

XML Elements v/s Attributes:

Take a look at these examples:

- ```
<person gender="male">
 <firstname>Ravi</firstname>
 <lastname>Chythanya</lastname>
</person>
```
- ```
<person>
  <gender>male</gender>
  <firstname>Ravi</firstname>
  <lastname>Chythanya</lastname>
</person>
```

In the first example gender is an attribute. In the next, gender is an element. Both examples provide the same information. There are no rules about when to use attributes or when to use elements. Attributes are handy in html. In XML my advice is to avoid them. Use elements instead.

My favorite way:

The following three XML documents contain exactly the same information:

- 1) A date attribute is used in the first example:

```
<note date="10/01/2008">
  <to>tove</to>
  <from>jani</from>
  <heading>reminder</heading>
  <body>don't forget me this weekend!</body>
</note>
```

- 2) A date element is used in the second example:

```
<note>
  <date>10/01/2008</date>
  <to>tove</to>
  <from>jani</from>
```

```

    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
  </note>

```

3) An expanded date element is used in the third: (this is my favorite):

```

  <note>
    <date>
      <day>10</day>
      <month>01</month>
      <year>2008</year>
    </date>
    <to>tove</to>
    <from>jani</from>
    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
  </note>

```

Avoid XML attributes?

Some of the problems with using attributes are:

- Attributes cannot contain multiple values (elements can)
- Attributes cannot contain tree structures (elements can)
- Attributes are not easily expandable (for future changes)
- Attributes are difficult to read and maintain. use elements for data. use attributes for information that is not relevant to the data.
- Don't end up like this:

```

<note day="10" month="01" year="2008" to="tove" from="jani" heading="reminder"
body="don't forget me this weekend!"> </note>

```

XML attributes for metadata:

Sometimes ID references are assigned to elements. These IDs can be used to identify XML elements in much the same way as the id attribute in html. This example demonstrates this:

```

<messages>
  <note id="501">
    <to>tove</to>
    <from>jani</from>
    <heading>reminder</heading>
    <body>don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>jani</to>
    <from>tove</from>
    <heading>re: reminder</heading>
    <body>i will not</body>
  </note>
</messages>

```

The ID attributes above are for identifying the different notes. It is not a part of the note itself. What I'm trying to say here is that metadata (data about data) should be stored as attributes, and the data itself should be stored as elements.

iii) XML Entities:

An entity is the symbolic representation of some information. It references the data that look like an abbreviation or can be found at an external location. Entities reduce the redundant information and also allow for easier editing. The entities can be internal or external. For example, the entity '**&**' will replace as '**&**', where ever it is found in the XML document.

iv) PCDATA:

PCDATA stands for Parsed Character Data. The text enclosed between starting and ending tag is known as Character Data. The parser parses the PCDATA and identifies the entities as well as markup. Markups include the tags inside the text and entities are expanded. In the case of Parsed Character Data, the symbols '&', '<', '>' are represented using entities.

v) CDATA:

CDATA stands for Character Data and it is not parsed by the parser. The tags inside the text are not considered as markup and the entities are not expanded.

Validation of XML:

There are two levels of correctness of an XML document:

- 1) XML with correct syntax is **Well Formed XML**.
- 2) XML validated against a DTD is **Valid XML**.

A **Well Formed XML** document is a document that conforms to the XML syntax rules like:

- XML documents must have a root element.
- XML elements must have a closing tag.
- XML tags are case sensitive.
- XML elements must be properly nested.
- XML attribute values must be quoted

Example:

```
<?XML version="1.0" encoding="iso-8859-1"?>
<note>
  <to>tove</to>
  <from>jani</from>
  <heading>reminder</heading>
  <body>don't forget me this weekend!</body>
</note>
```

A **Valid XML** document is a "well formed" XML document, which conforms to the rules of a document type definition (DTD).

Document Type Definition (DTD):

The purpose of a DTD is to define the structure of an XML document. It defines the structure with a list of legal elements:

DTD Syntax:

```
<!DOCTYPE DOCUMENT [
  <!ELEMENT ELEMENT_NAME1 (Attribute Names) Appearance of attributes>
  <!ELEMENT ELEMENT_NAME2>
```

```

<!ELEMENT ELEMENT_NAME3>
.
.
.
<!ELEMENT ELEMENT_NAMEn>
<!ATTLIST Element_name Attribute_name Attribute_type Default_value>
]>

```

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

Internal DTD Declaration:

If the DTD is declared inside the XML file, it should be wrapped in DOCTYPE definition as in the following example:

```

<!DOCTYPE note[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#CDATA)>
]>
<note>
  <to>Ravi</to>
  <from>Chythanya</from>
  <heading>Message</heading>
  <body>Welcome to the XML with DTD</body>
</note>

```

The DTD above is interpreted like this:

- !DOCTYPE note defines that the root element of this document is note.
- !ELEMENT note defines that the note element contains four elements “to, from, heading, body”.
- !ELEMENT to defines that to element to be of the type #PCDATA.
- !ELEMENT from defines that from element to be of the type #PCDATA.
- !ELEMENT heading defines that heading element to be of the type #PCDATA.
- !ELEMENT body defines that body element to be of the type #CDATA.

External DTD Declaration:

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

```
<!DOCTYPE DOCUMENT SYSTEM “File_Name.dtd”>
```

Example: emailDTD.xml:

```

<?xml version=”1.0”?>
<!DOCTYPE note SYSTEM “emailDTD.dtd”>
<note>
  <to>Ravi</to>
  <from>Chythanya</from>
  <heading>Message</heading>

```

```
<body>Welcome to the XML with DTD</body>
</note>
```

emailDTD.dtd:

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

XML Schema:

W3C supports an XML-based alternative to DTD, called XML schema:

- 1) The XML schema is used to represent the structure of XML document. The goal or purpose of XML schema is to define the building blocks of an XML document. These can be used as an alternative to XML DTD. The XML schema language is called as XML Schema Definition (XSD) language. The XML schema became the World Wide Web Consortium (W3C) recommendation in 2001.
- 2) XML schema defines elements, attributes, elements having child elements, order of child elements. It also defines fixed and default values of elements and attributes.
- 3) XML schema also allows the developers to use data types.
- 4) File extension of XML schema is “.xsd” i.e., Filename.xsd

Ex:**StudentSchema.xsd:**

```
<?XML version="1.0"?>
<xs:schema XMLNs:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="std" type="xs:string"/>
        <xs:element name="marks" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Document (myschema.XML):

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<student XMLNs:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "studentschema.xsd">
  <name>anand</name>
  <address>knr</address>
  <std>second</std>
  <marks>70percent</marks>
</student>
```

Data Types in XML:

Various data types are that can be used to specify the data types of an elements are:

- String
- Date
- Time
- Numeric
- Boolean

1. String Data Type:

The string data type can be used to define the elements containing the characters lines, tabs, white spaces, or special characters.

Example:

XML Schema [stringType.xsd]:

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

XML Document[stringTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<StudentName xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="stringType.xsd">
  Ravi Chythanya
</StudentName>
```

2. Date Data Type:

For specifying the date we use date data type. The format of date is yyyy-mm-dd, where yyyy denotes the Year, mm denotes the month and dd specifies the day.

Example:

XML Schema [dateType.xsd]:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DOB" type="xs:date">
</xs:schema>
```

XML Document[dateTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<DOB xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="dateType.xsd">
  1988-03-01
</DOB>
```

3. Time Data Type:

For specifying the time we use time data type. The format of date is hh:mm:ss, where hh denotes the Hour, mm denote the minutes and ss specify the seconds.

Example:

XML Schema [timeType.xsd]:

```
<?xml version="1.0" ?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MyTime" type="xs:time">
</xs:schema>
```

XML Document[timeTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<MyTime xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "timeType.xsd">
  01:57:03
</MyTime>
```

4. Numeric Data Type:

If we want to use numeric value for some element then we can use the data types as either “decimal” or “integer”.

Example:**XML Schema [decimalType.xsd]:**

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Percentage" type="xs:decimal">
</xs:schema>
```

XML Document[decimalTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Percentage xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "decimalType.xsd">
  99.99
</Percentage>
```

XML Schema [integerType.xsd]:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Age" type="xs:integer">
</xs:schema>
```

XML Document[integerTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Age xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "integerType.xsd">
  30
</Age>
```

5. Boolean Data Type:

For specifying either **true** or **false** values we must use the Boolean data type.

Example:**XML Schema [booleanType.xsd]:**

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



```
<xs:element name= "Flag" type="xs:boolean">
</xs:element>
```

XML Document[booleanTypeDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Flag xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "booleanType.xsd">
    true
</Flag>
```

Simple Types:

XML defines the simple type which contains only text and does not contain any other element or attributes. But the **text** appears for element comes along with some **type**. The syntax is:

```
<xs:element name= "element_name" type= "data_type">
```

Here the type can be any built in data type:

- xs:string
- xs:date
- xs:time
- xs:integer
- xs:decimal
- xs:Boolean

Restrictions on Simple Types:

For an XML document, it is possible to put certain restrictions on its contents. These restrictions are called as **facets**.

XML Schema [Month.xsd]:

```
<?xml version= "1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLSchema">
    <xs:element name= "Month">
        <xs:simpleType>
            <xs:restriction base= "xs:integer">
                <xs:minInclusive value= "1">
                <xs:maxInclusive value= "12">
            </xs:restriction>
        </xs:simpleType>
    </xs:element>
</xs:schema>
```

XML Document[MonthDemo.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Age xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "Month.xsd">
    12
</Month>
```

Various facets that can be applied are:

- length – Specifies exact number of characters.

- minLength – Specifies the minimum number of characters allowed.
- maxLength – Specifies the maximum number of characters allowed.
- minInclusive – Specifies lower bound for numeric values.
- minExclusive – Specifies lower bound for numeric values.
- maxInclusive – Specifies upper bound for numeric values.
- maxExclusive – Specifies upper bound for numeric values.
- enumeration – A list of acceptable values can be defined.
- pattern – specifies exact sequence of characters that are acceptable.
- totalDigits – Specifies the exact number of digits.
- fractionDigits – Specifies maximum number of decimal places allowed.
- whiteSpace – Specifies how to handle white spaces. The white spaces can be spaces, tabs, line feed, or carriage return.

Restrictions that can be applied on the data types are:

string	date	numeric	boolean
enumeration	enumeration	enumeration	enumeration
Length	maxExclusive	fractionDigits	length
minLength	maxInclusive	maxExclusive	maxLength
maxLength	minExclusive	maxInclusive	minLength
Pattern	minInclusive	minExclusive	pattern
whiteSpaces	pattern	minInclusive	whiteSpaces
	whiteSpaces	totalDigits	
		whiteSpaces	

Complex Types:

1) Elements:

Complex elements are the elements that contain some other elements or attributes.

The complex types of elements are:

- Empty Elements
- Elements that contain text
- Elements that contain other elements
- Elements that contain other text as well other elements

i) Empty Elements:

Let us define the empty elements by the complexType.

Example [Student.xsd]:

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Student">
    <xs:complexType>
      <xs:attribute name="HTNo" type="xs:integer">
    </xs:complexType>
```

```

</xs:element>
</xs:schema>
XML Document[Student.xml]:
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceschemaLocation = "Student.xsd" HTNo="1343">
</Student>

```

ii) Elements that contain text only:

We have the elements that contain simple contents such as text and attributes. Hence we must add the type as **simpleContent** in the schema definition. This simpleContent element must have either extension or restriction.

Example [Student.xsd]:

```

<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Student">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:integer">
          <xs:attribute name="HTNo" type="xs:integer">
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

XML Document[Student.xml]:

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceschemaLocation = "Student.xsd" HTNo="1343">
  100
</Student>

```

iii) Elements that contain other elements only:

Example [Student.xsd]:

```

<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firstName" type="xs:string">
        <xs:element name="lastName" type="xs:string">
      </xs:sequence>
    </xs:complexType>
  </xs:element>

```

```
</xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:nonamespacesthemalocation = "Student.xsd">
  <firstName>Ravi</firstName>
  <lastName>Chythanya</lastName>
</Student>
```

iv) Elements that contain text as well as other elements:**Example [Student.xsd]:**

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs=" http://www.w3.org/2001/XMLSchema">
  <xs:element name="Student">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element name="Name" type="xs:string">
          <xs:element name="HTNo" type="xs:integer">
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

XML Document[Student.xml]:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Student xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:nonamespacesthemalocation = "Student.xsd">
  My Dear Student <Name>Ravi</Name>
  With Hall Ticket Number <HTNo>0536</HTNo>
</Student>
```

2) Indicators:

Indicators allow us to define the elements in the manner we want. Following are some indicators that are available in XML Schema.

- i) all
- ii) choice
- iii) sequence
- iv) maxOccurs
- v) minOccurs