HTML Introduction

HTML is the standard markup language for creating Static Web pages.

What is HTML?

- HTML stands for Hyper Text Markup Language
- HTML is the standard markup language for creating Web pages
- HTML describes the structure of a Web page
- HTML consists of a series of elements
- HTML elements tell the browser how to display the content
- HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

HTML Syntax

```
<!DOCTYPE html>
<html>
 <head>
  <title>Page Title</title>
 </head>
 <body>

 //any code

 </body>
</html>
```

Syntax Explained

- The `<!DOCTYPE html>` declaration defines that this document is an HTML5 document
- The `<html>` element is the root element of an HTML page
- The `<head>` element contains meta information about the HTML page
- The `<title>` element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The `<body>` element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.

Web Browsers

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.
A browser does not display the HTML tags, but uses them to determine how to display the document:

HTML History

Since the early days of the World Wide Web, there have been many versions of HTML:

Year	Version
1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

We follow the latest HTML5 standard.

STEPS TO CREATE HTML FILE:

Step 1: Open Any text editor(notepad, notepad++, any IDE...)
Step 2: Write Some HTML code
Step 3: Save the HTML Page(.HTML OR .HTM)
Step 4: View the HTML Page in Your Browser

The <!DOCTYPE> Declaration

The `<!DOCTYPE>` declaration represents the document type, and helps browsers to display web pages correctly.
It must only appear once, at the top of the page (before any HTML tags).
The `<!DOCTYPE>` declaration is not case sensitive.
The `<!DOCTYPE>` declaration for HTML5 is: `<!DOCTYPE html>`

HTML Headings

HTML headings are defined with the `<h1>` to `<h6>` tags.
`<h1>` defines the most important heading. `<h6>` defines the least important heading:

Example

```
<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
```

HTML Paragraphs

HTML paragraphs are defined with the `<p>` tag:

Example

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

HTML Links

HTML links are defined with the `<a>` tag:

Example

```
<a href="https://www.flipkart.com">This is a link</a>
```

The link's destination is specified in the `href` attribute.
Attributes are used to provide additional information about HTML elements.
You will learn more about attributes in a later chapter.

HTML Images

HTML images are defined with the `<img>` tag.
The source file (`src`), alternative text (`alt`), `width`, and `height` are provided as attributes:

Example

```
<img src="rose.jpg" alt="rose image to be
displayed" width="104" height="142">
```

How to View HTML Source?

Right-click in an HTML page and select "View Page Source" (in Chrome) or "View Source" (in Edge), or similar in other browsers. This will open a window containing the HTML source code of the page.

HTML Lists

HTML lists allow web developers to group a set of related items in lists.

An unordered HTML list:

- Item
- Item
- Item
- Item

An ordered HTML list:

1. First item
2. Second item
3. Third item
4. Fourth item

Unordered HTML List

An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag.

The list items will be marked with bullets (small black circles) by default:

Example

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

Unordered HTML List - Choose List Item Marker

The CSS `list-style-type` property is used to define the style of the list item marker. It can have one of the following values:

Value	Description
Disc	Sets the list item marker to a bullet (default)
Circle	Sets the list item marker to a circle
Square	Sets the list item marker to a square
None	The list items will not be marked

Example - Disc

```
<ul style="list-style-type:disc;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Circle

```
<ul style="list-style-type:circle;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - Square

```
<ul style="list-style-type:square;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Example - None

```
<ul style="list-style-type:none;">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

Ordered HTML List

An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag.
The list items will be marked with numbers by default:

Example

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

Ordered HTML List - The Type Attribute

The `type` attribute of the `<ol>` tag, defines the type of the list item marker:

Type	Description
type="1"	The list items will be numbered with numbers (default)
type="A"	The list items will be numbered with uppercase letters
type="a"	The list items will be numbered with lowercase letters
type="I"	The list items will be numbered with uppercase roman numbers
type="i"	The list items will be numbered with lowercase roman numbers

Numbers:

```
<ol type="1">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Letters:

```
<ol type="A">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Letters:

```
<ol type="a">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Uppercase Roman Numbers:

```
<ol type="I">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Lowercase Roman Numbers:

```
<ol type="i">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

Control List Counting

By default, an ordered list will start counting from 1. If you want to start counting from a specified number, you can use the `start` attribute:

Example

```
<ol start="50">
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

HTML Description Lists

A description list is a list of terms, with a description of each term.
The `<dl>` tag defines the description list, the `<dt>` tag defines the term (name), and the `<dd>` tag describes each term:

Example

```
<dl>
  <dt>Coffee</dt>
  <dd>- black hot drink</dd>
  <dt>Milk</dt>
  <dd>- white cold drink</dd>
</dl>
```

Summary

- Use the HTML `<dl>` element to define a description list
- Use the HTML `<dt>` element to define the description term
- Use the HTML `<dd>` element to describe the term in a description list

HTML Tables

HTML tables allow web developers to arrange data into rows and columns

Define an HTML Table

The `<table>` tag defines an HTML table.
Each table row is defined with a `<tr>` tag. Each table header is defined with a `<th>` tag. Each table data/cell is defined with a `<td>` tag.
By default, the text in `<th>` elements are bold and centered.
By default, the text in `<td>` elements are regular and left-aligned.

Example

A simple HTML table:

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>CONTACT</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Note: The `<td>` elements are the data containers of the table.
They can contain all sorts of HTML elements; text, images, lists, other tables, etc.

HTML Table - Add a Border

To add a border to a table, use the CSS `border` property:

Example

```
table, th, td {
  border: 1px solid black;
}
```

Remember to define borders for both the table and the table cells.

HTML Table - Collapsed Borders

To let the borders collapse into one border, add the CSS `border-collapse` property:

Example

```
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
}
```

HTML Table - Add Cell Padding

Cell padding specifies the space between the cell content and its borders. If you do not specify a padding, the table cells will be displayed without padding.

To set the padding, use the CSS `padding` property:

Example

```
th, td {
  padding: 15px;
}
```

HTML Table - Left-align Headings

By default, table headings are bold and centered.

To left-align the table headings, use the CSS `text-align` property:

Example

```
th {
  text-align: left;
}
```

HTML Table - Add Border Spacing

Border spacing specifies the space between the cells.

To set the border spacing for a table, use the CSS `border-spacing` property:

Example

```
table {
  border-spacing: 5px;
}
```

Note: If the table has collapsed borders, `border-spacing` has no effect.

HTML Table - Cell that Span Many Columns

To make a cell span more than one column, use the `colspan` attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name</th>
    <th colspan="2">Telephone</th>
  </tr>
  <tr>
    <td>Bill Gates</td>
    <td>55577854</td>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Cell that Span Many Rows

To make a cell span more than one row, use the `rowspan` attribute:

Example

```
<table style="width:100%">
  <tr>
    <th>Name:</th>
    <td>Bill Gates</td>
  </tr>
  <tr>
    <th rowspan="2">Telephone:</th>
    <td>55577854</td>
  </tr>
  <tr>
    <td>55577855</td>
  </tr>
</table>
```

HTML Table - Add a Caption

To add a caption to a table, use the `<caption>` tag:

Example

```
<table style="width:100%">
  <caption>Monthly savings</caption>
  <tr>
    <th>Month</th>
    <th>Savings</th>
  </tr>
  <tr>
    <td>January</td>
    <td>$100</td>
```

```
    </tr>
    <tr>
      <td>February</td>
      <td>$50</td>
    </tr>
  </table>
```

Note: The `<caption>` tag must be inserted immediately after the `<table>` tag.

Summary

- Use the HTML `<table>` element to define a table
- Use the HTML `<tr>` element to define a table row
- Use the HTML `<td>` element to define a table data
- Use the HTML `<th>` element to define a table heading
- Use the HTML `<caption>` element to define a table caption
- Use the CSS `border` property to define a border
- Use the CSS `border-collapse` property to collapse cell borders
- Use the CSS `padding` property to add padding to cells
- Use the CSS `text-align` property to align cell text
- Use the CSS `border-spacing` property to set the spacing between cells
- Use the `colspan` attribute to make a cell span many columns
- Use the `rowspan` attribute to make a cell span many rows
- Use the `id` attribute to uniquely define one table

HTML Images

Images can improve the design and the appearance of a web page.

HTML Images Syntax

The HTML `<img>` tag is used to embed an image in a web page.
Images are not technically inserted into a web page; images are linked to web pages. The `<img>` tag creates a holding space for the referenced image.
The `<img>` tag is empty, it contains attributes only, and does not have a closing tag.
The `<img>` tag has two required attributes:

- src - Specifies the path to the image
- alt - Specifies an alternate text for the image

Syntax

```
<img src="url" alt="alternatetext">
```

The src Attribute

The required `src` attribute specifies the path (URL) to the image.

Example

```
<img src="rose.jpg" alt="Rose Flower to be displayed">
```

The alt Attribute

The required `alt` attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).
The value of the `alt` attribute should describe the image:

Example

```
<img src="rose.jpg" alt=" Rose Flower to be displayed ">
```

Image Size - Width and Height

You can use the `style` attribute to specify the width and height of an image.

Example

```
<img src="rose.jpg" alt=" Rose Flower to be displayed
" style="width:500px;height:600px;">
```

Alternatively, you can use the `width` and `height` attributes:

Example

```
<img src="rose.jpg" alt=" Rose Flower to be displayed
" width="500" height="600">
```

Images in Another Folder

If you have your images in a sub-folder, you must include the folder name in the `src` attribute:

Example

```
<img src=" F:\web technologies\unit-2\practicals\nature.jpg" alt="HTML5
 Icon" style="width:128px;height:128px;">
```

Images on Another Server/Website

Some web sites points to an external image on another server.
To point to an image on another server, you must specify an absolute (full) URL in the `src` attribute:

Example

```
<img src="https://www.w3schools.com/images/w3schools_green.jpg" alt="W
3Schools.com">
```

Common Image Formats

Here are the most common image file types, which are supported in all browsers (Chrome, Edge, Firefox, Safari, Opera):

Abbreviation	File Format	File Extension
APNG	Animated Portable Network Graphics	.apng
GIF	Graphics Interchange Format	.gif
ICO	Microsoft Icon	.ico, .cur
JPEG	Joint Photographic Expert Group image	.jpg, .jpeg, .jfif, .pjpeg, .pjp
PNG	Portable Network Graphics	.png
SVG	Scalable Vector Graphics	.svg

Summary

- Use the HTML `<img>` element to define an image
- Use the HTML `src` attribute to define the URL of the image
- Use the HTML `alt` attribute to define an alternate text for an image, if it cannot be displayed
- Use the HTML `width` and `height` attributes or the CSS `width` and `height` properties to define the size of the image
- Use the CSS `float` property to let the image float to the left or to the right

Note: Loading large images takes time, and can slow down your web page. Use images carefully.

HTML Forms

An HTML form is used to collect user input. The user input is most often sent to a server for processing.

The <form> Element

The HTML `<form>` element is used to create an HTML form for user input:

```
<form>
.
form elements
.
</form>
```

The `<form>` element is a container for different types of input elements, such as: text fields, checkboxes, radio buttons, submit buttons, etc.

The <input> Element

The HTML `<input>` element is the most used form element.
An `<input>` element can be displayed in many ways, depending on the `type` attribute.
Here are some examples:

Type	Description
<input type="text">	Displays a single-line text input field
<input type="radio">	Displays a radio button (for selecting one of many choices)
<input type="checkbox">	Displays a checkbox (for selecting zero or more of many choices)
<input type="submit">	Displays a submit button (for submitting the form)
<input type="button">	Displays a clickable button

```
<form>
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname">
</form>
```

The <label> Element

Notice the use of the `<label>` element in the example above.
The `<label>` tag defines a label for many form elements.

The `<label>` element is useful for screen-reader users, because the screen-reader will read out loud the label when the user focus on the input element.
The `<label>` element also help users who have difficulty clicking on very small regions (such as radio buttons or checkboxes) - because when the user clicks the text within the `<label>` element, it toggles the radio button/checkbox.
The `for` attribute of the `<label>` tag should be equal to the `id` attribute of the `<input>` element to bind them together.

Radio Buttons

The `<input type="radio">` defines a radio button.
Radio buttons let a user select ONE of a limited number of choices.

Example

A form with radio buttons:

```
<form>
  <input type="radio" id="male" name="gender" value="male">
  <label for="male">Male</label><br>
  <input type="radio" id="female" name="gender" value="female">
  <label for="female">Female</label><br>
  <input type="radio" id="other" name="gender" value="other">
  <label for="other">Other</label>
</form>
```

This is how the HTML code above will be displayed in a browser:

Male

Female

Other

Checkboxes

The `<input type="checkbox">` defines a **checkbox**.
Checkboxes let a user select ZERO or MORE options of a limited number of choices.

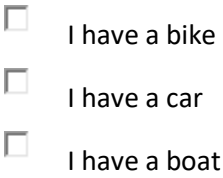
Example

A form with checkboxes:

```
<form>
  <input type="checkbox" id="vehicle1" name="vehicle1" value="Bike">
  <label for="vehicle1"> I have a bike</label><br>
  <input type="checkbox" id="vehicle2" name="vehicle2" value="Car">
  <label for="vehicle2"> I have a car</label><br>
  <input type="checkbox" id="vehicle3" name="vehicle3" value="Boat">
```

```
  <label for="vehicle3"> I have a boat</label>
</form>
```

This is how the HTML code above will be displayed in a browser:



The Submit Button

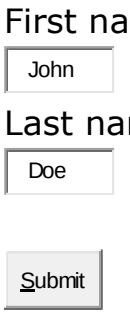
The `<input type="submit">` defines a button for submitting the form data to a form-handler.
The form-handler is typically a file on the server with a script for processing input data.
The form-handler is specified in the form's `action` attribute.

Example

A form with a submit button:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

This is how the HTML code above will be displayed in a browser:



The Name Attribute for <input>

Notice that each input field must have a `name` attribute to be submitted.
If the `name` attribute is omitted, the value of the input field will not be sent at all.

Example

This example will not submit the value of the "First name" input field:

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" value="John"><br><br>
```

```
  <input type="submit" value="Submit">
</form>
```

HTML Form Attributes

The Action Attribute

The `action` attribute defines the action to be performed when the form is submitted.
Usually, the form data is sent to a file on the server when the user clicks on the submit button.
In the example below, the form data is sent to a file called "action_page.php". This file contains a server-side script that handles the form data:

Example

On submit, send form data to "action_page.php":

```
<form action="/action_page.php">
  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>
  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>
  <input type="submit" value="Submit">
</form>
```

Tip: If the `action` attribute is omitted, the action is set to the current page.

The Target Attribute

The `target` attribute specifies where to display the response that is received after submitting the form.
The `target` attribute can have one of the following values:

Value	Description
_blank	The response is displayed in a new window or tab
_self	The response is displayed in the current window
_parent	The response is displayed in the parent frame
_top	The response is displayed in the full body of the window
Framename	The response is displayed in a named iframe

The default value is `_self` which means that the response will open in the current window.

Example

Here, the submitted result will open in a new browser tab:

```
<form action="/action_page.php" target="_blank">
```

The Method Attribute

The `method` attribute specifies the HTTP method to use used when submitting the form data.
The form-data can be sent as URL variables (with `method="get"`) or as HTTP post transaction (with `method="post"`).
The default HTTP method when submitting form data is GET.

Example

This example uses the GET method when submitting the form data:

```
<form action="/action_page.php" method="get">
```

Example

This example uses the POST method when submitting the form data:

```
<form action="/action_page.php" method="post">
```

Notes on GET:

- Appends the form data to the URL, in name/value pairs
- NEVER use GET to send sensitive data! (the submitted form data is visible in the URL!)
- The length of a URL is limited (2048 characters)
- Useful for form submissions where a user wants to bookmark the result
- GET is good for non-secure data, like query strings in Google

Notes on POST:

- Appends the form data inside the body of the HTTP request (the submitted form data is not shown in the URL)
- POST has no size limitations, and can be used to send large amounts of data.
- Form submissions with POST cannot be bookmarked

Tip: Always use POST if the form data contains sensitive or personal information!

The Autocomplete Attribute

The `autocomplete` attribute specifies whether a form should have autocomplete on or off.
When autocomplete is on, the browser automatically complete values based on values that the user has entered before.

Example

A form with autocomplete on:

```
<form action="/action_page.php" autocomplete="on">
```

HTML Form Elements

Tag	Description
<form>	Defines an HTML form for user input
<input>	Defines an input control
<textarea>	Defines a multiline input control (text area)
<label>	Defines a label for an <input> element
<fieldset>	Groups related elements in a form
<legend>	Defines a caption for a <fieldset> element
<select>	Defines a drop-down list
<optgroup>	Defines a group of related options in a drop-down list
<option>	Defines an option in a drop-down list
<button>	Defines a clickable button
<datalist>	Specifies a list of pre-defined options for input controls
<output>	Defines the result of a calculation

HTML Iframes

An HTML iframe is used to display a web page within a web page.

HTML Iframe Syntax

The HTML `<iframe>` tag specifies an inline frame.
An inline frame is used to embed another document within the current HTML document.

Syntax

```
<iframe src="url" title="description">
```

Tip: It is a good practice to always include a `title` attribute for the `<iframe>`. This is used by screen readers to read out what the content of the iframe is.

Iframe - Set Height and Width

Use the `height` and `width` attributes to specify the size of the iframe.
The height and width are specified in pixels by default:

Example

```
<iframe src="demo_iframe.htm" height="200" width="300" title="Iframe Example"></iframe>
```

Or you can add the `style` attribute and use the CSS `height` and `width` properties:

Example

```
<iframe src="demo_iframe.htm" style="height:200px;width:300px;" title="Iframe Example"></iframe>
```

Iframe - Remove the Border

By default, an iframe has a border around it.
To remove the border, add the `style` attribute and use the CSS `border` property:

Example

```
<iframe src="demo_iframe.htm" style="border:none;" title="Iframe Example"></iframe>
```

With CSS, you can also change the size, style and color of the iframe's border:

Example

```
<iframe src="demo_iframe.htm" style="border:2px solid
red;" title="Iframe Example"></iframe>
```

Iframe - Target for a Link

An iframe can be used as the target frame for a link.
The `target` attribute of the link must refer to the `name` attribute of the iframe:

Example

```
<iframe src="demo_iframe.htm" name="iframe_a" title="Iframe
Example"></iframe>

<p><a href="https://www.tkrec.cin" target="iframe_a">college
website</a></p>
```

Chapter Summary

- The HTML `<iframe>` tag specifies an inline frame
- The `src` attribute defines the URL of the page to embed
- Always include a `title` attribute (for screen readers)
- The `height` and `width` attributes specifies the size of the iframe
- Use `border:none;` to remove the border around the iframe

CSS Introduction

What is CSS?

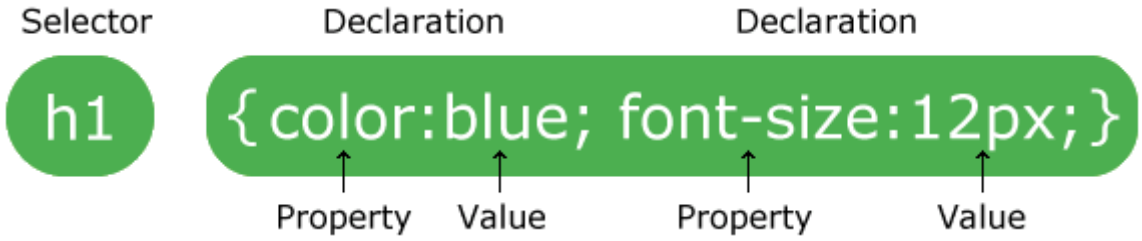
- **CSS** stands for **C**ascading **S**tyle **S**heets
- CSS describes **how HTML elements are to be displayed on screen, paper, or in other media**
- CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
- External stylesheets are stored in **CSS files**

Why Use CSS?

- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

CSS Syntax

- A CSS rule-set consists of a selector and a declaration block:



- The selector points to the HTML tag you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

Example

- In this example all <p> elements will be center-aligned, with a red text color:

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

Example Explained

- `p` is a **selector** in CSS (it points to the HTML element you want to style: <p>).
- `color` is a property, and `red` is the property value
- `text-align` is a property, and `center` is the property value

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style

The CSS element Selector

The element selector selects HTML elements based on the element name.

Example

Here, all <p> elements on the page will be center-aligned, with a red text color:

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

The CSS id Selector

The id selector uses the id attribute of an HTML element to select a specific element.
The id of an element is 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.

Example

The CSS rule below will be applied to the HTML element with id="para1":

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

Note: **An id name cannot start with a number!**

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.
To select elements with a specific class, write a period (.) character, followed by the class name.

Example

In this example all HTML elements with class="center" will be red and center-aligned:

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

You can also specify that only specific HTML elements should be affected by a class.

Example
In this example only <p> elements with class="center" will be center-aligned:

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

The CSS Universal Selector
The universal selector (*) selects all HTML elements on the page.

Example
The CSS rule below will affect every HTML element on the page:

```
* {
  text-align: center;
  color: blue;
}
```

How To Add CSS

Three Ways to Insert CSS
There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

External CSS
With an external style sheet, you can change the look of an entire website by changing just one file!
Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

Example
External styles are defined within the <link> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension.
The external .css file should not contain any HTML tags.
Here is how the "mystyle.css" file looks like:

"mystyle.css"

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

h1 {
  color: navy;
  margin-left: 20px;
}
```

Note: **Do not add a space between the property value and the unit (such as `margin-left: 20 px;`). The correct way is: `margin-left: 20px;`**

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.
The internal style is defined inside the <style> element, inside the head section.

Example

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
  background-color: linen;
}

h1 {
  color: maroon;
  margin-left: 40px;
}
</style>
</head>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Inline CSS

An inline style may be used to apply a unique style for a single element. To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

Example

Inline styles are defined within the "style" attribute of the relevant element:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="color:blue;text-align:center;">This is a heading</h1>
<p style="color:red;">This is a paragraph.</p>
</body>
</html>
```

Cascading Order

What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
<style>
h1 {
  color: orange;
}
</style>
</head>
<body>
<h1>This is a heading</h1>
<p>The style of this document is a combination of an external
stylesheet, and internal style</p>
</body>
</html>
```

CSS Colors

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

CSS Color Names

In CSS, a color can be specified by using a color name:

CSS Background Color

You can set the background color for HTML elements:

Example

```
<h1 style="background-color:DodgerBlue;">Hello World</h1>
<p style="background-color:Tomato;">Lorem ipsum...</p>
```

CSS Text Color

You can set the color of text:

Example

```
<h1 style="color:Tomato;">Hello World</h1>
<p style="color:DodgerBlue;">Lorem ipsum...</p>
<p style="color:MediumSeaGreen;">Ut wisi enim...</p>
```

CSS Color Values

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:
Same as color name "Tomato":

Example

```
<h1 style="background-color:rgb(255, 99, 71);">...</h1>
<h1 style="background-color:#ff6347;">...</h1>
<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>
<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>
```

CSS Backgrounds

CSS background-color

The `background-color` property specifies the background color of an element.

Example

The background color of a page is set like this:

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

CSS 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.

Example

The background image for a page can be set like this:

```
body {
  background-image: url("rose.jpg");
}
```

Introduction to XML

XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

The Difference Between XML and HTML

XML and HTML were designed with different goals:

- XML was designed to carry data - with focus on what data is
- HTML was designed to display data - with focus on how data looks
- XML tags are not predefined like HTML tags are

XML Tree Structure

- XML documents are formed as **element trees**.
- An XML tree starts at a **root element** and branches from the root to **child elements**.
- 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.
- Parents have children. Children have parents. Siblings are children on the same level (brothers and sisters).

XML Does Not Use Predefined Tags

- The XML language has no predefined tags.
- The tags in the example BELOW (like <to> and <from>) are not defined in any XML standard. These tags are "invented" by the author of the XML document.
- HTML works with predefined tags like <p>, <h1>, <table>, etc.
- With XML, the author must define both the tags and the document structure.

XML is Extensible

- Most XML applications will work as expected even if new data is added (or removed).

XML Syntax Rules

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>STUDENTS</to>
  <from>FACULTY</from>
  <heading>Reminder</heading>
  <message>Don't forget to register for EXMAS</message>
</note>
```

1.XML Documents Must Have a Root Element

XML documents must contain one **root** element that is the **parent** of all other elements: In ABOVE example **<note>** is the root element

2.The XML Prolog

- This line is called the XML **prolog**:
  ```
  <?xml version="1.0" encoding="UTF-8"?>
  ```
- The XML prolog is optional. If it exists, it must come first in the document.
- XML documents can contain international characters, like Norwegian øæå or French êèé.
- To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.
- UTF-8 is the default character encoding for XML documents.

3.All XML Elements Must Have a Closing Tag

- In XML, it is illegal to omit the closing tag. All elements **must** have a closing tag:
  ```
  <p>This is a paragraph.</p>
  <br />
  ```

  Note: The XML prolog does not have a closing tag! This is not an error. The prolog is not a part of the XML document.

4.XML Tags are Case Sensitive

- 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 correct</message>
  ```
- "Opening and closing tags" are often referred to as "Start and end tags". Use whatever you prefer. It is exactly the same thing.

5.XML Elements Must be Properly Nested

- In HTML, you might see improperly nested elements:
  ```
  <message><b><i>this is message</b></i></message>
  ```
- 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 <b> element, it must be closed inside the <b> element.

6.XML Attribute Values Must Always be Quoted

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

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>mahesh</from>
</note>
```

Avoid XML Attributes?

Some things to consider when 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)

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>
```

7.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>salary < 1000</message>
```

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

```
<message>salary &lt; 10000</message>
```

- There are 5 pre-defined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	Apostrophe
&quot;	"	quotation mark

- Only < and & are strictly illegal in XML, but it is a good habit to replace > with > as well

8.Comments in XML

- The syntax for writing comments in XML is similar to that of HTML:
- `<!-- This is a comment -->`
- Two dashes in the middle of a comment are not allowed:
- `<!-- This is an invalid -- comment -->`

9.White-space is Preserved in XML

- XML does not truncate multiple white-spaces (HTML truncates multiple white-spaces to one single white-space):

XML:	mahesh	kumar
HTML:	Mahesh kumar	

10.XML Stores New Line as LF

- Windows applications store a new line as: carriage return and line feed (CR+LF).
- Unix and Mac OSX use LF.
- Old Mac systems use CR.
- XML stores a new line as LF.

IMPORTANT: Well Formed XML

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Document Type Definition(DTD)

What is a DTD?

A DTD is a Document Type Definition.
A DTD defines the structure and the legal elements and attributes of an XML document.

Why Use a DTD?

With a DTD, independent groups of people can agree on a standard DTD for interchanging data.
An application can use a DTD to verify that XML data is valid.

An Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the <!DOCTYPE> definition:

```
<?xml version="1.0" encoding=”UTF-8”?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT message (#PCDATA)>
]>
<note>
  <to>STUDENTS</to>
  <from>FACULTY</from>
  <heading>Reminder</heading>
  <message>Don't forget to register for EXAMS</message>
</note>
```

In the XML file, select "view source" to view the DTD.

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 must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT message** defines the body element to be of type "#PCDATA"

An External DTD Declaration

- If the DTD is declared in an external file, the <!DOCTYPE> definition must contain a reference to the DTD file:

XML document with a reference to an external DTD

```
<?xml version="1.0" encoding=”UTF-8”?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>STUDENTS</to>
  <from>FACULTY</from>
  <heading>Reminder</heading>
  <message>Don't forget to register for EXMAS</message>
</note>
```

And here is the file "**note.dtd**", which contains the DTD:

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

The Building Blocks of XML Documents

Seen from a DTD point of view, all XML documents are made up by the following building blocks:

- Elements
- Attributes
- Entities
- PCDATA
- CDATA

Elements

Elements are the **main building blocks** of both XML and HTML documents. Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Examples:

```
<body>some text</body>

<message>some text</message>
```

Attributes

Attributes provide **extra information about elements**.

Attributes are always placed inside the opening tag of an element. Attributes always come in name/value pairs. The following "img" element has additional information about a source file:

```
<img src="computer.gif" />
```

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

Entities

Some characters have a special meaning in XML, like the less than sign (<) that defines the start of an XML tag.
Most of you know the HTML entity: " ". This "no-breaking-space" entity is used in HTML to insert an extra space in a document. Entities are expanded when a document is parsed by an XML parser.
The following entities are predefined in XML:

Entity References	Character
<	<
>	>
&	&
"	"
'	'

PCDATA

PCDATA means parsed character data.
Think of character data as the text found between the start tag and the end tag of an XML element.
PCDATA is text that WILL be parsed by a parser. **The text will be examined by the parser for entities and markup**.
Tags inside the text will be treated as markup and entities will be expanded.
However, parsed character data should not contain any &, <, or > characters; these need to be represented by the < and > entities, respectively.

CDATA

CDATA means character data.
CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

XML Schema

An XML Schema describes the structure of an XML document.
The XML Schema language is also referred to as XML Schema Definition (XSD).

XSD Example

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

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The purpose of an XML Schema is to define the legal building blocks of an XML document:

- the elements and attributes that can appear in a document
- the number of (and order of) child elements
- data types for elements and attributes
- default and fixed values for elements and attributes

Why Learn XML Schema?

In the XML world, hundreds of standardized XML formats are in daily use.
Many of these XML standards are defined by XML Schemas.
XML Schema is an XML-based (and more powerful) alternative to DTD.

XML Schemas Support Data Types

One of the greatest strength of XML Schemas is the support for data types.

- It is easier to describe allowable document content
- It is easier to validate the correctness of data
- It is easier to define data facets (restrictions on data)
- It is easier to define data patterns (data formats)
- It is easier to convert data between different data types

XML Schemas use XML Syntax

Another great strength about XML Schemas is that they are written in XML.

- You don't have to learn a new language
- You can use your XML editor to edit your Schema files
- You can use your XML parser to parse your Schema files
- You can manipulate your Schema with the XML DOM
- You can transform your Schema with XSLT

XML Schemas are extensible, because they are written in XML.

With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML Schemas Secure Data Communication

- When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content.
- With XML Schemas, the sender can describe the data in a way that the receiver will understand.
- A date like: "03-11-2004" will, in some countries, be interpreted as 3.November and in other countries as 11.March.
- However, an XML element with a data type like this:
  - <date type="date">2004-03-11</date>
- ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

Well-Formed is Not Enough

A well-formed XML document is a document that conforms to the XML syntax rules, like:

- it must begin with the XML declaration
- it must have one unique root element
- start-tags must have matching end-tags
- elements are case sensitive
- all elements must be closed
- all elements must be properly nested
- all attribute values must be quoted
- entities must be used for special characters

Even if documents are well-formed they can still contain errors, and those errors can have serious consequences.

Think of the following situation: you order 5 gross of laser printers, instead of 5 laser printers. With XML Schemas, most of these errors can be caught by your validating software.

XML documents can have a reference to an XML Schema.

A Simple XML Document

Look at this simple XML document called "note.xml":

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>STUDENTS</to>
  <from>FACULTY</from>
  <heading>Reminder</heading>
  <message>Don't forget to register for EXMAS</message>
</note>
```

An XML Schema

The following example is an XML Schema file called "note.xsd" that defines the elements of the XML document above ("note.xml"):

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

The note element is a **complex type** because it contains other elements. The other elements (to, from, heading, body) are **simple types** because they do not contain other elements. You will learn more about simple and complex types in the following chapters.

A Reference to an XML Schema

This XML document has a reference to an XML Schema:

```
<?xml version="1.0"?>
<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml      note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't      forget      me      this      weekend!</body>
</note>
```

The <schema> element is the root element of every XML Schema.

The <schema> Element

The <schema> element is the root element of every XML Schema:

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

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

The <schema> element may contain some attributes. A schema declaration often looks something like this:

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

The below fragment indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace. It also specifies that the elements and data types that come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with **xs:**

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

The below fragment indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "https://www.w3schools.com" namespace.

```
targetNamespace="https://www.w3schools.com"
```

The below fragment indicates that the default namespace is "https://www.w3schools.com".

```
xmlns="https://www.w3schools.com"
```

The below fragment indicates that any elements used by the XML instance document which were declared in this schema must be namespace qualified.

```
elementFormDefault="qualified"
```

What is a Simple Element?

- A simple element is an XML element that can contain only text. It cannot contain any other elements or attributes.
- However, the "only text" restriction is quite misleading. The text can be of many different types. It can be one of the types included in the XML Schema definition (boolean, string, date, etc.), or it can be a custom type that you can define yourself.
- You can also add restrictions (facets) to a data type in order to limit its content, or you can require the data to match a specific pattern.

Defining a Simple Element

The syntax for defining a simple element is:

```
<xs:element name="xxx" type="yyy"/>
```

where xxx is the name of the element and yyy is the data type of the element.

XML Schema has a lot of built-in data types. The most common types are:

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

Example

Here are some XML elements:

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

And here are the corresponding simple element definitions:

```
<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.

In the following 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.

XSD Restrictions/Facets

Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

Restrictions on Values

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 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 "password" with a restriction. There must be exactly eight characters in a row and those characters must be lowercase or uppercase letters from a to z, or a number from 0 to 9:

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z0-9]{8}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions for Datatypes

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
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than 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
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
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

What is a Complex Element?

A complex element is an XML element that contains other elements and/or attributes.

There are four kinds of complex elements:

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

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex XML element, "product", which is empty:

```
<product pid="1345"/>
```

A complex XML element, "employee", which contains only other elements:

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

A complex XML element, "food", which contains only text:

```
<food type="dessert">Ice cream</food>
```

A complex XML element, "description", which contains both elements and text:

```
<description>
It     happened     on <date lang="norwegian">03.03.99</date> ....
</description>
```

How to Define a Complex Element

Look at this complex XML element, "employee", which contains only other elements:

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

We can define a complex element in an XML Schema two different ways:

The "employee" element can be declared directly by naming the element, like this:

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

If you use the method described above, only the "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

Document Object Model (DOM)

What is the DOM?

- The DOM defines a standard for accessing and manipulating documents:
- *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- The HTML DOM defines a standard way for accessing and manipulating HTML documents. It presents an HTML document as a tree-structure.
- The XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure.
- Understanding the DOM is a must for anyone working with HTML or XML.

The HTML DOM

All HTML elements can be accessed through the HTML DOM.
This example changes the value of an HTML element with id="demo":

Example

```
<!DOCTYPE html>
<html>
<body>
<h1 id="demo">This is a Heading</h1>
<p>This is a paragraph.</p>
      <script>
      document.getElementById("demo").innerHTML = "Hello World!";
      </script>
</body>
</html>
```

This example changes the value of the first <h1> element in an HTML document:

```
<!DOCTYPE html>
<html>
<body>
<h1>This is a Heading</h1>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<script>
document.getElementsByTagName("h1")[0].innerHTML = "Hello World!";
</script>
</body>
</html>
```

Note: Even if the HTML document contains only ONE <h1> element you still have to specify the array index [0], because the getElementsByTagName() method always returns an array.

The XML DOM

All XML elements can be accessed through the XML DOM.
The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

Get the Value of an XML Element

This code retrieves the text value of the first <title> element in an XML document:

Example

```
txt = xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
```

Loading an XML File

The XML file used in the examples below is books.xml.
This example reads "books.xml" into xmlDoc and retrieves the text value of the first <title> element in books.xml:

Example

Books.xml:

```
<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">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price> </book>
```

```
  <book category="web" cover="paperback">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price> </book>
</bookstore>
```

Book.html:

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>

<script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
    myFunction(this);
    }
};
xhttp.open("GET", "books.xml", true);
xhttp.send();
function myFunction(xml) {
    var xmlDoc = xml.responseXML;
    document.getElementById("demo").innerHTML =

xmlDoc.getElementsByTagName("title")[0].childNodes[0].nodeValue;
}
</script>

</body>
</html>
```

Example Explained

- **xmlDoc** - the XML DOM object created by the parser.
- **getElementsByTagName("title")[0]** - get the first <title> element
- **childNodes[0]** - the first child of the <title> element (the text node)
- **nodeValue** - the value of the node (the text itself)

XML DOM Properties

These are some typical DOM properties:

- x.nodeName - the name of x
- x.nodeValue - the value of x
- x.parentNode - the parent node of x
- x.childNodes - the child nodes of x
- x.attributes - the attributes nodes of x

Note: In the list above, x is a node object.

XML DOM Methods

- x.getElementsByTagName(*name*) - get all elements with a specified tag name
- x.appendChild(*node*) - insert a child node to x
- x.removeChild(*node*) - remove a child node from x

Note: In the list above, x is a node object.

XHTML | Introduction

XHTML stands for **EXtensible HyperText Markup** Language. It is the next step to evolution of internet. The XHTML was developed by World Wide Web Consortium (W3C). It helps web developers to make the transition from HTML to XML. Using XHTML, developers can enter the XML world with all the features of it, and they can still remain confident about the backward and future compatibility of the content. The XHTML 1.0 is the first document type in the XHTML family and it is Recommended by W3C in 26 January 2000. The XHTML 1.1 is Recommended by W3c in 31 May 2001. The XHTML5 is a standard and is used to develop an XML adaptation of the HTML5 specification. The XHTML documents contains three parts, which are discussed below:

- **DOCTYPE:** It is used to declare a DTD
- **head:** The head section is used to declare the title and other attributes.
- **body:** The body tag contains the content of web pages. It consists many tags.

Creating a XHTML web page, it is necessary to include DTD (Document Type Definition) declaration. There are three types of DTD which are discussed below:

- Transitional DTD
- Strict DTD
- Frameset DTD

Transitional DTD: It is supported by the older browsers which does not have inbuilt cascading style sheets supports. There are several attributes enclosing the body tag which are not allowed in strict DTD.

Syntax:

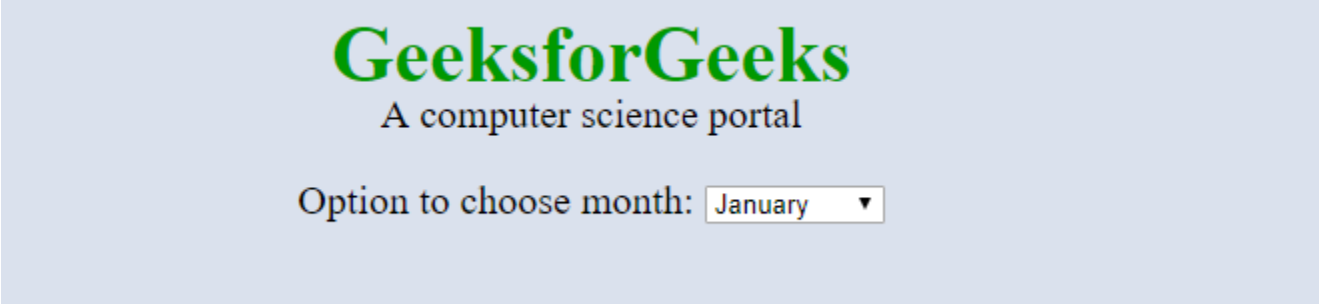
```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Transitional DTD XHTML</title>
    </head>
    <body bgcolor="#dae1ed">
        <div style="color:#090;font-size:40px;
                font-weight:bold;text-align:center;
                margin-bottom:-25px;">GeeksforGeeks</div>
        <p style="text-align:center;font-size:20px;">
                A computer science portal</p>
```

```
        <p style="text-align:center;font-size:20px;">
        Option to choose month:
        <select name="month">
            <option selected="selected">January</option>
            <option>February</option>
            <option>March</option>
            <option>April</option>
            <option>May</option>
            <option>June</option>
            <option>July</option>
            <option>Augusy</option>
            <option>September</option>
            <option>October</option>
            <option>November</option>
            <option>December</option>
        </select>
        </p>
    </body>
</html>
```

Output:



2) Strict DTD: Strict DTD is used when XHTML page contains only markup language. Strict DTD is used together with cascading style sheets, because this attribute does not allow CSS property in body tag.

Syntax:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

3) Frameset DTD: The frameset DTD is used when XHTML page contains frames.

Syntax:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"DTD/xhtml1-frameset.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Why use XHTML?

- XHTML documents are validated with standard XML tools.
- It is easily to maintain, convert, edit document in the long run.
- It is used to define the quality standard of web pages.
- XHTML is an official standard of the W3C, your website becomes more compatible and accurate with many browsers.

Benefits of XHTML:

- All XHTML tags must have closing tags and are nested correctly. This generates cleaner code.
- XHTML documents are lean which means they use less bandwidth. This reduces cost particularly if your web site has 1000s of pages.
- XHTML documents are well formated well–formed and can easily be transported to wireless devices, Braille readers and other specialized web environments.
- All new developments will be in XML (of which XHTML is an application).
- XHTML works in association with CSS to create web pages that can easily be updated.

Difference Between HTML and XHTML:

➢ HTML	➢ XHTML
➢ HTML or HyperText Markup Language is the main markup language for creating web pages	➢ XHTML (Extensible HyperText Markup Language) is a family of XML markup languages that mirror or extend versions of the widely used Hypertext Markup Language (HTML)
➢ Flexible framework requiring lenient HTML specific parser	➢ Restrictive subset of XML which needs to be parsed with standard XML parsers
➢ Proposed by Tim Berners-Lee in 1987	➢ World Wide Web Consortium Recommendation in 2000.

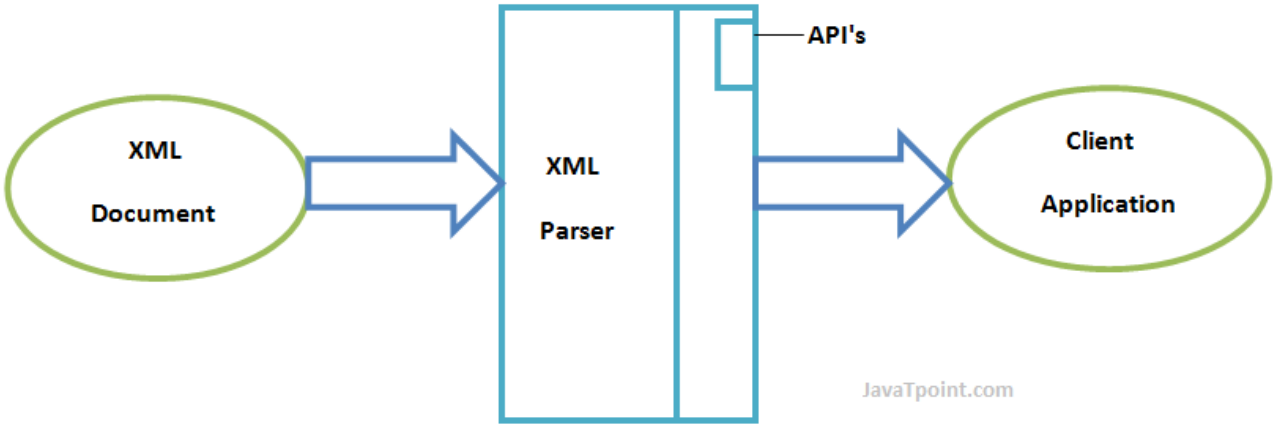
➢ HTML	➢ XHTML
➢ Application of Standard Generalized Markup Language (SGML).	➢ Application of XML
➢ Extended from SGML.	➢ Extended from XML, HTML

XML Parsers

An XML parser is a software library or package that provides interfaces for client applications to work with an XML document. The XML Parser is designed to read the XML and create a way for programs to use XML.

XML parser validates the document and check that the document is well formatted.

Let's understand the working of XML parser by the figure given below:



DOM Vs SAX Parser in Java

Difference between DOM Vs SAX Parser is very popular **java interview question** and often asked when interviewed on Java and XML. Both DOM and SAX parser are extensively used to read and parse XML file in java applications and both of them have their own set of advantages and disadvantages. In this post, I am listing down some big and easily seen differences between both parsers.

1. DOM XML Parser in Java

- **DOM parser is a tree-based API**. A tree-based API is centered around a tree structure and therefore provides interfaces on components of a tree (which is a DOM document) such as **Document** interface, **Node** interface, **NodeList** interface, **Element** interface, **Attr** interface and so on.
- A DOM parser creates a tree structure in memory from the input document and then waits for requests from client. A DOM parser always serves the client application with the **entire document no matter how much is actually needed** by the client. With DOM parser, method calls in client application have to be explicit and forms a kind of chained method calls.

2. SAX XML Parser in Java

- **SAX parser is a event-based API**. Usually an event-based API provides interfaces on handlers. There are four handler

interfaces, **ContentHandler** interface, **DTDHandler** interface, **EntityResolver** interface and **ErrorHandler** interface.

- SAX parser **does not create any internal structure**. Instead, it takes the occurrences of components of a input document as events, and tells the client what it reads as it reads through the input document. SAX parser serves the client application always **only with pieces of the document at any given time**. With SAX parser, some custom methods are called ["**callback**" methods] when some certain events occur during parsing on xml document. These methods do not have to be called explicitly by the client, though we could call them explicitly.

3. Difference between DOM and SAX XML Parser in Java

Let's list down an easy to remember short list of differences.

DOM (Document Object Model)

- Parses entire document
- Represents result as a tree
- Lets you search tree
- Lets you modify tree
- Good for reading data/configuration files

SAX

- Parses until you tell it to stop
- Fires event handlers for each:
    1. Start tag
    2. Tag body
    3. End tag
- Low level APIs
- Good for very large documents, especially if you only care about very small portions of the document.

4. How to choose between DOM and SAX Parsers?

Ideally a good parser should be fast (time efficient),space efficient, rich in functionality and easy to use . But in reality, none of the main parsers have all these features at the same time. For example, a DOM Parser is rich in functionality (because it creates a DOM tree in memory and allows you to access any part of the document repeatedly and allows you to modify the DOM tree), but it is space inefficient when the document is huge, and it takes a little bit long to learn how to work with it.

A SAX Parser, however, is much more space efficient in case of big input document (because it creates no internal structure). What's more, it runs faster and is easier to learn than DOM Parser because its API is really simple. But from the functionality point of view, it provides less functions which mean that the users themselves have to take care of more, such as creating their own data structures.
I think the answer really **depends on the characteristics of your application and your current requirements**.

5. Can SAX and DOM parsers be used at the same time?

Yes, of course, because the use of a DOM parser and a SAX parser is independent. For example, if your application needs to work on two XML documents, and does different things on each document, you could use a DOM parser on one document and a SAX parser on another, and then combine the results or make the processing cooperate with each other.