SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

School of Computing

Department of Information Technology

UNIT - I

Internet Programming – SIT1302

I. MARKUP LANGUAGE

Introduction to HTML

Brief History of HTML

In the late 1980's , A physicist, Tim Berners-Lee who was a contractor at CERN, proposed a system for CERN researchers. In 1989, he wrote a memo proposing an internet based hypertext system. **Tim Berners-Lee** is known as father of HTML. The first available description of HTML was a document called "HTML Tags" proposed by Tim in late 1991.

Features of HTML

1) It is a very **easy and simple** language. It can be easily understood and modified.

2) It is very easy to make **effective presentation** with HTML because it has a lot of *formatting tags*.

3) It is a **markup language** so it provides a flexible way to design web pages along with the text.

4) It facilitates programmers to add **link** on the web pages (by *html anchor tag*) , so it enhances the interest of browsing of the user.

5) It is **platform-independent** because it can be displayed on any platform like Windows, Linux and Macintosh etc.

6) It facilitates the programmer to add **Graphics, Videos, and Sound** to the web pages which makes it more attractive and interactive.

- HTML stands for Hyper Text Markup Language.
- HTML is used to create web pages.
- HTML is widely used language on the web.
- We can create static website by HTML only.
- HTML is an acronym which stands for Hyper Text Markup Language. Let's see what is Hyper Text and what is Markup Language?
- **Hyper Text:** Hyper Text simply means "Text within Text". A text has a link within it, is a hypertext. Every time when you click on a word which brings you to a new webpage, you have clicked on a hypertext.
- **Markup language:** A markup language is a programming language that is used make text more interactive and dynamic. It can turn a text into images, tables, links etc.

An HTML document is made of many HTML tags and each HTML tag contains different content.

Structure of HTML

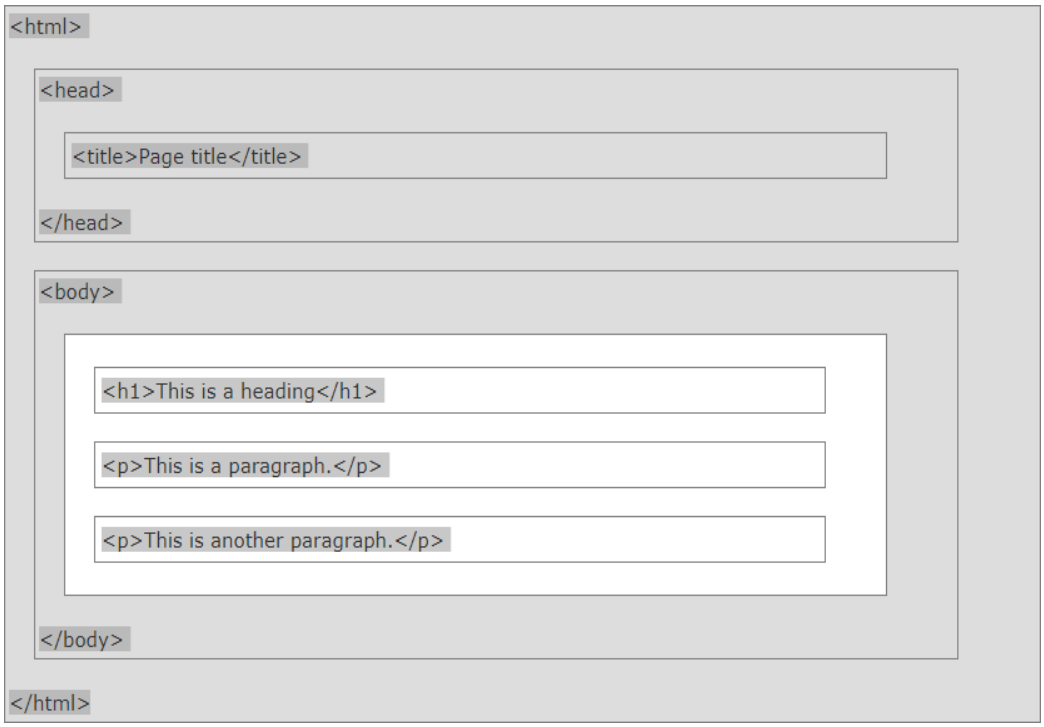


Fig 1.1: Structure of HTML

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.

Let's see a simple example of HTML.

```
<!DOCTYPE>
<html>
<body>

<h1>Write Your First Heading</h1>

<p>Write Your First Paragraph.</p>

</body>
</html>
```

Description of HTML example:

DOCTYPE: It defines the document type.

html : Text between html tag describes the web document.

body : Text between body tag describes the body content of the page that is visible to the end user.

h1 : Text between h1 tag describes the heading of the webpage.

p : Text between p tag describes the paragraph of the webpage.

HTML Elements:

HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags.When a web browser reads an HTML document, browser reads it from top to bottom and left to right. HTML tags are used to create HTML documents and render their properties. Each HTML tags have different properties.

Syntax of HTML Elements:

<tagname> content </tagname>

Note: HTML Tags are always written in lowercase letters. The basic HTML tags are given below:

HTML Formatting Tags:

Formatting is a process of formatting text for better look and feel. There are many formatting tags in HTML. These tags are used to make text bold, italicized, or underlined. There are almost 15 options available that how text appears in HTML.

1) Bold Text

If you write anything within ..... element, is shown in bold letters.

See this example:

** **Write Your First Paragraph in bold text.** **

Output:

Write a First Paragraph in bold text.

2) Italic Text

If you write anything within ..... element, is shown in italic letters.

example:

** **Write Your First Paragraph in italic text.** **

Output:

Write a First Paragraph in italic text.

3) Marked formatting

If you want to mark or highlight a text, you should write the content within <mark> ..... </mark>.

example:

<h2> I want to put a **<mark>** Mark**</mark>** on your face**</h2>**

Output:

I want to put a Mark on your face

4) Underlined Text

If you write anything within <u> ..... </u> element, is shown in underlined text.

See this example:
** <u>**Write Your First Paragraph in underlined text.**</u> **

Output:
Write Your First Paragraph in underlined text.

5) Strike Text

Anything written within <strike>.....</strike> element is displayed with strikethrough. It is a thin line which crosses the statement.

Example:

** <strike>**Write Your First Paragraph with strikethrough**</strike>**.** **

Output:

~~Write Your First Paragraph with strikethrough.~~

6) Monospaced Font

If you want that each letter has the same width then you should write the content within <tt>.....</tt> element.

Note: We know that most of the fonts are known as variable-width fonts because different letters have different width. (for example: 'w' is wider than 'i'). Monospaced Font provides similar space among every letter.

See this example:

```
<p>Hello <tt>Write Your First Paragraph in monospaced font.</tt></p>
```

Output:

Hello Write Your First Paragraph in monospaced font.

7) Superscript Text

If you put the content within ^{.....} element, is shown in superscript ; means it is displayed half a character's height above the other characters.

Example:

```
<p>Hello <sup>Write Your First Paragraph in superscript.</sup></p>
```

Output:

Hello Write Your First Paragraph in superscript.

8) Subscript Text

If you put the content within _{.....} element, is shown in subscript; means it is displayed half a character's height below the other characters.

Example:

```
<p>Hello <sub>Write Your First Paragraph in subscript.</sub></p>
```

Output:

Hello Write Your First Paragraph in subscript.

9) Deleted Text

Anything that puts within <del> .....</del> is displayed as deleted text.

example:

```
<p>Hello <del>Delete your first paragraph.</del></p>
```

Output:

Hello

10) Inserted Text

Anything that puts within <ins> ..... </ins> is displayed as inserted text.

Example:

```
<p> <del>Delete your first paragraph.</del><ins>Write another paragraph.</ins></ p>
```

Output:

Write another paragraph.

11) Larger Text

If you want to put your font size larger than the rest of the text then put the content within <big> .....</big>. It increase one font size larger than the previous one.

example:

```
<p>Hello <big>Write the paragraph in larger font.</big></p>
```

Output:

Hello Write the paragraph in larger font.

12) Smaller Text

If you want to put your font size smaller than the rest of the text then put the content within <small>.....</small>tag. It reduces one font size than the previous one.

example:

** **Hello **<small>**Write the paragraph in smaller font.**</small> **

Output:

Hello Write the paragraph in smaller font.

13) Paragraph

If you want to display the text content in paragraph format then use ..... tag.

Example:

This tag is used to display the contents in paragraph format. i.e., the sentences are printed continuously without any line breaks.

Output:

This tag is used to display the contents in paragraph format. i.e., the sentences are printed continuously without any line breaks.

Space inside HTML Paragraph

If you put a lot of spaces inside the HTML p tag, browser removes extra spaces and extra line while displaying the page. The browser counts number of spaces and lines as a single one.

14) Ordered & Unordered Lists:

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain:

<ul> - An unordered list. This will list items using plain bullets.

<ol> - An ordered list. This will use different schemes of numbers to list your items.

<dl> - A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTML Unordered Lists:

An unordered list is a collection of related items that have no special order or sequence. This list is

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
<ul>
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ul>
</body>
</html>
```

created by using HTML **<ul>** tag. Each item in the list is marked with a bullet.

Example:

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

Type Attribute:

You can use **type** attribute for <ul> tag to specify the type of bullet you like. By default it is a disc. Following are the possible options:

```
<ul type="square">


<ul type="disc">
```

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
   <ul type="square">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li
   >
   <li>Radish</li>
   </ul>
</body>
</html>
```

Example:

Following is an example where we used <ul type="square">

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
   <ul type="disc">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ul>
</body>
</html>
```

Following is an example where we used <ul type="disc"> :

This will produce following result:

- Beetroot
- Ginger
- Potato

- Radish

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
   <ul type="circle">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ul>
</body>
</html>
```

Example:Following is an example where we used <ul type="circle">

This will produce following result:

- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists:

If you are required to put your items in a numbered list instead of bulleted then HTML ordered list will be used. This list is created by using **<ol>** tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with

Example

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Unordered List</title>
</head>
<body>
   <ol type="circle">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ol>
</body>
```

This will produce following result:

1. Beetroot
2. Ginger
3. Potato
4. Radish

The type Attribute:

You can use **type** attribute for <ol> tag to specify the type of numbering you like. By default it is a number. Following are the possible options:

```
<ol type="1"> - Default-Case Numerals.
<ol type="I"> - Upper-Case Numerals.
<ol type="i"> - Lower-Case Numerals.
<ol type="a"> - Lower-Case Letters.
<ol type="A"> - Upper-Case Letters.
```

Example:

Following is an example where we used <ol type="1">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol type="1">
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ol>
```

This will produce following result:

1. Beetroot
2. Ginger
3. Potato
4. Radish

Example

Following is an example where we used <ol type="I">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
   <ol type="I">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ol>
</body>
</html>
```

This will produce following result:

I. Beetroot
II. Ginger
III. Potato
IV. Radish

Example

Following is an example where we used <ol type="i">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol type="i">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
    </ol> </body></html>
```

This will produce following result:

i. Beetroot
ii. Ginger
iii. Potato
iv. Radish

Following is an example where we used <ol type="A">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
   <ol type="A">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ol>
</body>
</html>
```

This will produce following result:

A. Beetroot
B. Ginger
C. Potato
D. Radish

Following is an example where we used <ol type="a">

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
<ol type="a">
<li>Beetroot</li>
<li>Ginger</li>
<li>Potato</li>
<li>Radish</li>
</ol>
</body>
</html>
```

This will produce following result:

a. Beetroot
b. Ginger
c. Potato
d. Radish

The start Attribute

You can use **start** attribute for <ol> tag to specify the starting point of numbering you need.

```
<ol type="1"          - Numerals starts with
<ol type="I"          - Numerals starts with
<ol type="i"          - Numerals starts with
<ol type="a"          - Letters starts with
<ol type="A"          - Letters starts with
```

Following are the possible options:

Following is an example where we used <ol type="i" start="4" >

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Ordered List</title>
</head>
<body>
   <ol type="i" start="4">
   <li>Beetroot</li>
   <li>Ginger</li>
   <li>Potato</li>
   <li>Radish</li>
   </ol>
</body>
</html>
```

This will produce following result:

iv. Beetroot
v. Ginger
vi. Potato
vii. Radish

15) HTMLDefinition Lists:

HTML and XHTML support a list style which is called **definition lists** where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

<dl> - Defines the start of the list

<dt> - A term

<dd> - Term definition

</dl> - Defines the end of the list Example:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Definition List</title>
</head>
<body>
<dl>
<dt><b>HTML</b></dt>
<dd>This stands for Hyper Text Markup Language</dd>
<dt><b>HTTP</b></dt>
<dd>This stands for Hyper Text Transfer Protocol</dd>
</dl>
</body>
</html>
```

This will produce following result:

HTML

This stands for Hyper Text Markup Language

HTTP

This stands for Hyper Text Transfer Protocol

16) Marquee HTML:

The Marquee HTML tag is a non-standard HTML element which is used to scroll a image or text horizontally or vertically. In simple words, you can say that it scrolls the image or text up, down, left or right automatically. Marquee tag was first introduced in early versions of Microsoft's Internet Explorer. It is compared with Netscape's blink element.

Example:

<marquee>This is an example of html marquee </marquee>

HTML Marquee Attributes:

Marquee's element contains several attributes that are used to control and adjust the appearance of the marquee.

Attribute	Description
Behavior	It facilitates user to set the behavior of the marquee to one of the three different types: scroll, slide and alternate.
Direction	defines direction for scrolling content. It may be left, right, up and down.
Width	defines width of marquee in pixels or %.
Height	defines height of marquee in pixels or %.
Hspace	defines horizontal space in pixels around the marquee.
Vspace	defines vertical space in pixels around the marquee.
Scrolldelay	defines scroll delay in seconds.
scrollamount	defines scroll amount in number.
Loop	defines loop for marquee content in number.
Bgcolor	defines background color. It is now deprecated.

HTML Scroll Marquee:

It is a by default property. It is used to scroll the text from right to left, and restarts at the right side of the marquee when it is reached to the end of left side. After the completion of loop text disappears.

<marquee width="100%" behavior="scroll" bgcolor="pink">

This is an example of a scroll marquee...

</marquee>

HTML Slide Marquee:

In slide marquee, all the contents to be scrolled will slide the entire length of marquee but stops at the end to display the content permanently.

<marquee width="100%" behavior="slide" bgcolor="pink">

This is an example of a slide marquee...

</marquee>

HTML Alternate Marquee:

It scrolls the text from right to left and goes back left to right.

<marquee width="100%" behavior="alternate" bgcolor="pink">

This is an example of a alternate marquee...

</marquee>

Direction in HTML marquee:

This is used to change the direction of scrolling text. Let's take an example of marquee scrolling to the right. The direction can be left, right, up and down.

<marquee width="100%" direction="right"> This is an example of a right direction marquee...
</marquee>

Disadvantages HTML marquee:

1) Marquee may be distracting because human eyes are attracted towards movement and marquee text constantly.
2) Since Marquee text moves, so it is more difficult to click static text, depending on the scrolling speed.
3) It is a non-standard HTML element.
4) It draws user's attention needlessly and makes the text harder to read.

** Element**

** ** is used for defining a section of your document. With the div tag, you can group large sections of HTML elements together and format them with CSS. The difference between the div tag and the span tag is that the div tag is used with block-level elements whilst the span tag is used with inline elements

Example:

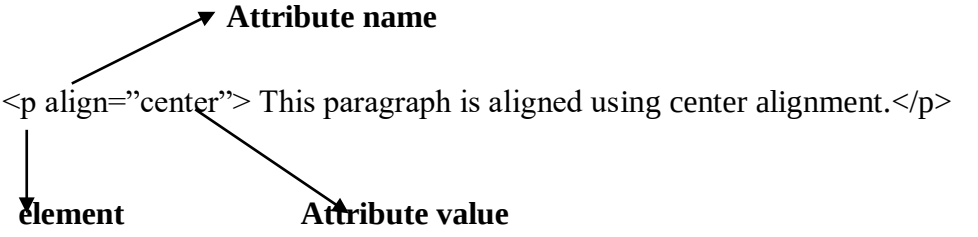
```
<html>
<head> <title>HTML div Tag</title>
</head>
<body>
<div id="contentinfo">
<p>Welcome to our page. We provide HTML notes.</p>
</div>
</body>
</html>
```

HTML Attributes

Attributes provide additional information about HTML elements.

- HTML elements can have attributes
- Attributes provide additional information about an element
- Attributes are always specified in the start tag
- Attributes come in name/value pairs like: name="value"

Example:



HTML Styling:

Every HTML element has a default style (background color is white, text color is black, text-size is 12px ...)

Changing the default style of an HTML element, can be done with the style attribute. This example changes the default background color from white to light grey.

Example:

```
<body style="background-color:lightgrey">
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
```

HTML Style Attribute:

The HTML style attribute has the following syntax: style="property:value"

The property is a CSS property. The value is a CSS value.

HTML Text Color:

The color property defines the text color to be used for an HTML element:

Example:

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

HTML Text Fonts:

The font-family property defines the font to be used for an HTML element: Example:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="font-family:verdana">This is a heading</h1>
<p style="font-family:courier">This is a paragraph.</p>
</body>
</html>
```

HTML Text Size:

The font-size property defines the text size to be used for an HTML element: Example:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="font-size:300%">This is a heading</h1>
<p style="font-size:160%">This is a paragraph.</p>
</body>
</html>
```

HTML Text Alignment:

The text-align property defines the horizontal text alignment for an HTML element: Example:

```
<!DOCTYPE html>
<html>
<body>
<h1 style="text-align:center">Centered Heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Markup Tags for inserting Images, URL, Tables and Frames:

a. Images:

HTML images are defined with the <img> tag. Example:

```
<img src="Sathyabama.jpg" alt="www.sathyabamauniversity.ac.in" width="104" height="142">
```

- The source file (src), alternative text (alt), and size (width and height) are attributes.
- The alt attribute specifies an alternative text to be used, when an HTML element cannot be displayed.

b. URL:

The **HTML anchor tag** defines *a hyperlink that links one page to another page* i.e., an element in an electronic document that links to another place in the same document or to an entirely different document.

The "href" attribute is the most important attribute of the HTML a tag.

href attribute of HTML anchor tag:

The href attribute is used to define the address of the file to be linked. In other words, it points out the destination page.

The syntax of HTML anchor tag is given below.
 Link Text

Let's see an example of HTML anchor tag.

Click for Second Page

Visit Google India

- The href attribute specifies the destination address (https://www.google.co.in) of the link.
- The link text is the visible part (Visit Google India).
- Clicking on the link text will send you to the specified address.

The target Attribute

- The target attribute specifies where to open the linked document.
- The target attribute can have one of the following values:

  - _blank - Opens the linked document in a new window or tab
  - _self - Opens the linked document in the same window/tab as it was clicked (this is default)
  - _parent - Opens the linked document in the parent frame
  - _top - Opens the linked document in the full body of the window
  - framename - Opens the linked document in a named frame

Example

Visit Google India This example will open the linked document in a new browser window/tab.

Appearance of HTML anchor tag:

- An unvisited link is displayed **underlined** and **blue**.
- A visited link displayed **underlined** and **purple**.
- An active link is **underlined** and **red**.

c. Table:

HTML table tag is used to display data in tabular form (row * column). There can be many columns in a row.

HTML tables are used to manage the layout of the page e.g. header section, navigation bar, body content, footer section etc. But it is recommended to use div tag over table to manage the layout of the page.

Tag	Description
<table>	It defines a table.
<tr>	It defines a row in a table.
<th>	It defines a header cell in a table.
<td>	It defines a cell in a table.
<caption>	It defines the table caption.
<colgroup>	It specifies a group of one or more columns in a table for formatting.
<col>	It is used with <colgroup> element to specify column properties for each column.
<tbody>	It is used to group the body content in a table.
<thead>	It is used to group the header content in a table.
<tfooter>	It is used to group the footer content in a table.

Example:

```
<html>
<body>
<table border="1">
<tr>
<th>Table Header</th>
<th>Table Header</th>
</tr>
<tr>
<td>Table cell 1</td>
<td>Table cell 2</td>
</tr>
<tr>
<td>Table cell 3</td>
<td>Table cell 4</td>
</tr>
</table>
</body>
</html>
```

Output:

Table Header	Table Header
Table cell 1	Table cell 2
Table cell 3	Table cell 4

Colspan and Rowspan Attributes:

You will use **colspan** attribute if you want to merge two or more columns into a single column.

Similar way you will use **rowspan** if you want to merge two or more rows.

Example

```
<!DOCTYPE html>

<html>
<head>
<title>HTML Table Colspan/Rowspan</title>

</head>
<body>

 <table border="1">
 <tr>
 <th>Column 1</th>
 <th>Column 2</th>
 <th>Column 3</th>
 </tr>
 <tr><td rowspan="2">Row 1 Cell 1</td><td>Row 1 Cell 2</td><td>Row 1 Cell 3</td></tr>
 <tr><td>Row 2 Cell 2</td><td>Row 2 Cell 3</td></tr>
 <tr><td colspan="3">Row 3 Cell 1</td></tr>
 </table>
```

This will produce following result:

Column 1	**Column 2**	**Column 3**
Row1Cell1	Row1Cell2	Row1Cell3
	Row2Cell2	Row2Cell3
Row 3 Cell 1		

Table Header, Body, and Footer:

Tables can be divided into three portions: a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are:

- **<thead> -** to create a separate table header.
- **<tbody> -** to indicate the main body of the table.
- **<tfoot> -** to create a separate table footer.

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Table</title>
</head>
<body>
<table border="1" width="100%">
<thead>
<tr>
<td colspan="4">This is the head of the table</td>
</tr>
</thead>
<tfoot>
<tr>
<td colspan="4">This is the foot of the table</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
<td>Cell 3</td>
<td>Cell 4</td>
</tr>
</tbody>
</table>
</body>
```

<Frameset> element andtargetattribute:

One of the most popular uses of frames is to place navigation bars in one frame and then load main pages into a separate frame.

<frameset> tag defines a frameset.

- Each <frameset> holds one or more <frame> elements. Each <frame> element can hold a separate document.
- <frameset> element specifies HOW MANY columns or rows there will be in the frameset, and HOW MUCH percentage/pixels of space will occupy each of them.

Example:

A simple two-framed (row frames) page:

sampleframe.html:

```
<html>
<frameset rows="25%,*">
  <frame src="frame1.html">
  <frame src="frame2.html">
</frameset>
</html>
```

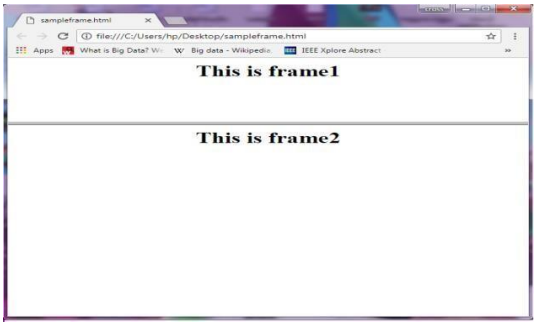
frame1.html:

```
<html>
<body>
<center><h1>This is frame1</h1></center>
</body>
</html>
```

frame2.html:

```
<html>
<body>
<center><h1>This is frame2</h1></center>
</body>
</html>
```

Output:



A simple two-framed (column frames) page:

sampleframe1.html:

```
<html>
<frameset cols="25%,*">
  <frame src="frame1.htm">
  <frame src="frame2.htm">
</frameset>
</html>
```

Output:

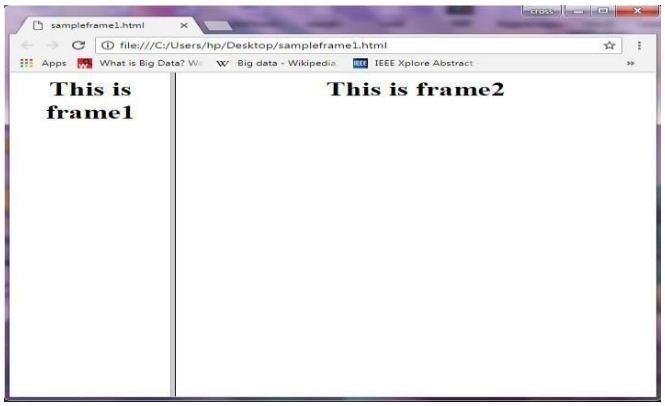


Image Mapping:

In Web page development, an image map is a graphic image defined so that a user can click on different areas of the image and be linked to different destinations. You make an image map by defining each of the sensitive areas in terms of their x and y coordinates (that is, a certain horizontal distance and a certain vertical distance from the left-hand corner of the image). With each set of coordinates, you specify a Uniform Resource Locator or Web address that will be linked to when the user clicks on that area.

Originally, the map file had to be sent to the server. Now the creator can place the map information either at the server or at the client (a "client-side map").

Client side image Mapping:

The X and Y coordinates are expressed in pixels in the same HTML file that contains the link to the image map.

clientmap.html:

```
<html>
<head>
<title>MAPPING</title>
</head>
<body>
<img src="color.jpg" usemap="#shapes">
<map name="shapes">
<area shape="circle" coords=101,114,100 href="circle.html">
<area shape="rect" coords=261,64,343,142 href="rect.html">
<area shape="poly" coords=223,154,171,204,171,256,221,284,271,256,273,204 href="poly.html">
</map>
</bodY>
</html>
```

Circle.html*:*

```
<html>
<head>
<title> circle</title>
</head>
<body>
<h3> This is a circle
</h3>
</body>
</html>
```

Rectangle.html:

```
<html>
<head>
<title> rectangle</title>
</head>
<body>
<h3> This is a rectangle</h3>
</body>
</html>
```

Polygon.html:

```
<html>
<head>
<title> polygon</title>
</head>
<body>
<h3> This is a polygon</h3>
<a href="image.bmp" target="third">image</a>
</body>
</html>
```

Server side image Mapping:

The X and Y coordinates are expressed in pixels in a different file which has **.map** extension that contains the link to the image map.

servermap.asp:

```
<html>
<head><title>Server-side Image map Example</title></head>
<body>
 <h1 align="center">Server-side Image map Test</h1><hr/>
 <a href="mapper.map">
 <img src="paradise.jpg" ismap alt="Shapes Map" border="0" width="400" height="200"></a>
</body>
</html>
```

Mapper.map

```
default a.jpg
poly b.jpg 16,358,216,378,206,556,66,574,14,358
circle c.jpg 200,200,100
rect Sunset.jpg 250,50,350,150
```

Cascading Style Sheets:

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet:

An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing just one file.

Each page must include a link to the style sheet with the <link> tag. The <link> tag goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet file must be saved with a **.css** extension. An example of a style sheet file called **"myStyle.css"**, is shown below:

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

Internal Style Sheet:

An internal style sheet should be used when a single document has a unique style. You define internal styles in the head section of an HTML page, inside the
<style> tag, like this:

Example:

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

}
</style>
</head>
```

Inline Style Sheet:

An inline style loses many of the advantages of a style sheet (by mixing content with presentation). To use inline styles, add the style attribute to the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a h1 element:

Example:

```
<h1 style="color: blue; margin-left: 30px;">This is a heading.</h1>
```

HTML Forms:

A form is simply an area that can contain form fields.

Form fields are objects that allow the visitor to enter information - for example text boxes, drop- down menus or radio buttons.

<Form> </Form>

To let the browser know where to send the content we add these attributes to the <form> tag:

- action=address

  The **address** is the url of script the content should be sent to.

- method=post or method=get

  The **post** and **get** methods are simply two different methods for submitting data to the script

POST offers better security because the submitted data is not visible in the page address.

If the form submission is passive (like a search engine query), and without sensitive information.

When you use **GET**, the form data will be visible in the page address:

enctype	Specifies the encoding of the submitted data (default: is url-encoded)

enctype="application/x-www-form-urlencoded"

.

These fields can be added to your forms:

- **Text field**

  Text fields are one line areas that allow the user to input text.

  <input type="text" size="25" value="Enter your name here!">

The **size** option defines the width of the field. That is how many visible characters it can contain. The **maxlength** option defines the maximum length of the field. That is how many characters can be entered in the field. If you do not specify a maxlength, the visitor can easily enter more characters than are visible in the field at one time. The **name** setting adds an internal name to the field so the program that handles the form can identify the fields. The **value** setting defines what will appear in the box as the default value.

- **Password field**

Password fields are similar to text fields. The difference is that what is entered into a password field shows up as dots on the screen. This is, of course, to prevent others from reading the password on the screen.

```
<input type="password" size="25">
```

- **Hidden field**

Hidden fields are similar to text fields, with one very important difference!

The difference is that the hidden field does not show on the page

```
<input type="hidden" name="Language" value="English">
```

- **Text area**

Text areas are text fields that can span several lines.

```
<textarea cols="40" rows="5" name="myname">
```

- **Check box**

Check boxes are used when you want to let the visitor select one or more options from a set of alternatives. If only one option is to be selected at a time you should use radio buttons instead.

```
<input type="checkbox" name="option1" value="Milk"> Milk<br>
<input type="checkbox" name="option2" value="Butter" checked>
<input type="checkbox" name="option3" value="Cheese">Cheese<br>
```

- **Radio button**

Radio buttons are used when you want to let the visitor select one - and just one - option from a set of alternatives. If more options are to be allowed at the same time you should use check boxes instead.

```
<input type="radio" name="group1" value="Milk"> Milk<br>
<input type="radio" name="group1" value="Butter" checked>Butter<br>
<input type="radio name=”group1” value="Cheese"> Cheese
```

- **Drop-down menu**

Drop-down menus are probably the most flexible objects you can add to your forms. Depending on your settings, drop-down menus can serve the same purpose as radio buttons (one selection only) or check boxes (multiple selections allowed). The advantage of a drop-down menu, compared to radio buttons or check boxes, is that it takes up less space. But that is also a disadvantage, because people can't see all options in the menu right away.

HTML	EXPLANATION
Select name= size= multiple= option selected value=	Drop-down menu Name of the field. Visible items in list. Allows multiple choices if yes. Individual items in the menu. Default select the item. Value to send if selected.

Drop-down menus combine <select> and <option>. Both tags have an opening and a closing tag.

A typical example of the syntax would be:

```
<select>
  <option>Milk</option>
  <option>Coffee</option>
  <option>Tea</option>
 </select>
```

<option value="Milk" selected>Milk</option>

- **Submit button**

When a visitor clicks a submit button, the form is sent to the address specified in the action of the <form> tag.

<input type="submit" value="Send me your name!">

- **Reset button**

When a visitor clicks a reset button, the entries are reset to the default values.

<input type="reset" value="Reset!">

- **Image button**

Image buttons have the same effect as submit buttons. When a visitor clicks an image button the form is sent to the address specified in the action setting of the <form> tag

```
<input type="image" src="rainbow.gif" name="image" width="60" height="60">
```

The HTML button Element represents a clickable button.

```
<button name="button" value="OK" type="button">Click Me</button>
```

Grouping Form Data with <fieldset>:

The <fieldset> element groups related data in a form. The <legend> element defines a caption for the <fieldset> element.

Example:

```
<html>
<body>
<form action="action_page.php">
<fieldset>
<legend>Personal information:</legend> First name:<br>
<input type="text" name="firstname" value="Mickey"><br> Last name:<br>
<input type="text" name="lastname" value="Mouse"><br><br>
<input type="submit" value="Submit"></fieldset>
</form>
</body>
</html>
```

HTML Audio Tag:

HTML audio tag is used to define sounds such as music and other audio clips. Currently there are three supported file format for HTML 5 audio tag.

1. mp3
2. wav
3. ogg

HTML5 supports <video> and <audio> controls. The Flash, Silverlight and similar technologies are used to play the multimedia items. This table defines that which web browser supports which audio file format.

Example:

Let's see the code to play mp3 file using HTML audio tag.

```
<audio controls>
<source src="koyal.mp3" type="audio/mpeg"> Your browser does not support the html audio tag.
</audio>
```

Example:

Let's see the example to play ogg file using HTML audio tag.

```
<audio controls>
<source src="koyal.ogg" type="audio/ogg"> Your browser does not support the html audio tag.
</audio>
```

Attributes of HTML Audio Tag:

There is given a list of HTML audio tag.

Attribute	Description
controls	It defines the audio controls which is displayed with play/pause buttons.
autoplay	It specifies that the audio will start playing as soon as it is ready.
loop	It specifies that the audio file will start over again, every time when it is completed.
muted	It is used to mute the audio output.
preload	It specifies the author view to upload audio file when the page loads.
src	It specifies the source URL of the audio file.

HTML Audio Tag Attribute

Example

Here we are going to use controls, autoplay, loop and src attributes of HTML audio tag.

```
<audio controls autoplay loop>
<source src="koyal.mp3" type="audio/mpeg">
</audio>
```

MIME Types for HTML Audio format:

The available MIME type HTML audio tag is given below.

Audio Format	MIME Type
mp3	audio/mpeg
Ogg	audio/ogg
Wav	audio/wav

HTML Video Tag:

HTML 5 supports <video> tag also. The HTML video tag is used for streaming video files such as a movie clip, song clip on the web page.

Currently, there are three video formats supported for HTML video tag:

1. mp4
2. webM
3. ogg Example:

Let's see the code to play mp4 file using HTML video tag.

```
<video controls>
<source src="movie.mp4" type="video/mp4"> Your browser does not support the html video tag.
</video>
```

Let's see the example to play ogg file using HTML video tag.

```
<video controls>
<source src="movie.ogg" type="video/ogg"> Your browser does not support the html video tag.
</video>
```

Attributes of HTML Video Tag:

Let's see the list of HTML 5 video tag attributes.

Attribute	Description
controls	It defines the video controls which is displayed with play/pause buttons.
height	It is used to set the height of the video player.
width	It is used to set the width of the video player.
poster	It specifies the image which is displayed on the screen when the video is not played.
autoplay	It specifies that the video will start playing as soon as it is ready.
loop	It specifies that the video file will start over again, every time when it is completed.
muted	It is used to mute the video output.
preload	It specifies the author view to upload video file when the page loads.
src	It specifies the source URL of the video file.

Example:

Let's see the example of video tag in HTML where are using height, width, autoplay, controls and loop attributes.

```
<video width="320" height="240" controls autoplay loop>
<source src="movie.mp4" type="video/mp4"> Your browser does not support the html video tag.
</video>
```

MIME Types for HTML Video format:

The available MIME type HTML video tag is given below.

Video Format	MIME Type
mp4	video/mp4
Ogg	video/ogg
webM	video/webM

HTML Canvas Tag:

The HTML canvas element provides HTML a bitmapped surface to work with. It is used to draw graphics on the web page. The HTML 5 <canvas> tag is used to draw graphics using scripting language like JavaScript. The <canvas> element is only a container for graphics, you must need a scripting language to draw the graphics. The <canvas> element allows for dynamic and scriptable rendering of 2D shapes and bitmap images. It is a low level, procedural model that updates a bitmap and does not have a built-in scene. There are several methods in canvas to draw paths, boxes, circles, text and add images.

DHTML

Stands for Dynamic HTML, it is totally different from HTML. The browsers which support the dynamic HTML are some of the versions of Netscape Navigator and Internet Explorer of version higher than 4.0. The DHTML is based on the properties of the HTML, JavaScript, CSS, and DOM (Document Object Model which is used to access individual elements of a document) which helps in making dynamic content. It is the combination of HTML, CSS, JS, and DOM. The DHTML make use of Dynamic object model to make changes in settings and also in properties and methods. It also makes uses of Scripting and it is also part of earlier computing trends. DHTML allows different scripting languages in a web page to change their variables, which enhance the effects, looks and many others functions after the whole page have been fully loaded or under a view process, or otherwise static HTML pages on the same. But in true ways, there is noting that as dynamic in DHTML, there is only the enclosing of different technologies like CSS, HTML, JS, DOM, and different sets of static languages which make it as dynamic. DHTML is used to create interactive and animated web pages that are generated in real-time, also known as dynamic web pages so that when such a page is accessed, the code within the page is analyzed on the web server and the resulting HTML is sent to the client's web browser.

HTML: HTML stands for Hypertext Markup Language and it is a client-side Markup language. It is used to build the block of web pages.

Javascript: It is a Client-side Scripting language. JavaScript is supported by most of the browser, also have cookies collection to determine the user needs.

CSS: The abbreviation of CSS is Cascading Style Sheet. It helps in the styling of the web pages and helps in designing of the pages. The CSS rules for DHTML will be modified at different levels using JS with event handlers which adds a significant amount of dynamism with very little code.

DOM: It is known as a Document Object Model which acts as the weakest links in it. The only defect in it is that most of the browser does not support DOM. It is a way to manipulate the static contents.

Key Features:

Following are the some major key features of DHTML:

1. Tags and their properties can be changed using DHTML.
2. It is used for real-time positioning.
3. Dynamic fonts can be generated using DHTML.
4. It is also used for data binding.
5. It makes a webpage dynamic and be used to create animations, games, applications along with providing new ways of navigating through websites.
6. The functionality of a webpage is enhanced due to the usage of low-bandwidth effect by DHTML.
7. DHTML also facilitates the use of methods, events, properties, and codes.

Why DHTML?

DHTML makes a webpage dynamic but JavaScript also does, the question arises that what different does DHTML do? So the answer is that DHTML has the ability to change a WebPages look, content and style once the document has loaded on our demand without changing or deleting everything already existing on the browser's webpage.

DHTML can change the content of a webpage on demand without the browser having to erase everything else, i.e. being able to alter changes on a webpage even after the document has completely loaded.

Advantages:

- Sizes of the files are compact in compared to other interactional media like Flash or Shockwave, and it downloads faster.
- It is supported by big browser manufacturers like Microsoft and Netscape.
- Highly flexible and easy to make changes.
- Viewer requires no extra plug-ins for browsing through the webpage that uses DHTML, they do not need any extra requirements or special software to view it.
- User time is saved by sending less number of requests to the server. As it is possible to modify and replace elements even after a page is loaded, it is not required to create separate pages for changing styles which in turn saves time in building pages and also reduces the number of requests that are sent to the server.
- It has more advanced functionality than a static HTML. it is capable of holding more content on the web page at the same time.

Disadvantages:

- It is not supported by all the browsers. It is supported only by recent browsers such as Netscape 6, IE 5.5, and Opera 5 like browsers.
- Learning of DHTML requires a lot of pre-requisites languages such as HTML, CSS, JS, etc should be known to the designer before starting with DHTML which is a long and time-consuming in itself.
- Implementation of different browsers are different. So if it worked in one browser, it might not necessarily work the same way in another browser.
- Even after being great with functionality, DHTML requires a few tools and utilities that are some expensive. For example, the DHTML text editor, Dreamweaver. Along with it the improvement cost of transferring from HTML to DHTML makes cost rise much higher.

Difference between HTML and DHTML:

- HTML is a markup language while DHTML is a collection of technologies.
- HTML is used to create static webpages while DHTML is capable of creating dynamic webpages.
- DHTML is used to create animations and dynamic menus but HTML not used.
- HTML sites are slow upon client-side technologies whereas DHTML sites are comparatively faster.
- Web pages created using HTML are rather simple and have no styling as it uses only one language whereas DHTML uses HTML, CSS, and JavaScript which results in a much better and way more presentable webpage.
- HTML cannot be used as server side code but DHTML used as server side code.
- DHTML needs database connectivity but not in case of HTML.
- Files in HTML are stored using .htm or .html extension while DHTML uses .dhtm extension.
- HTML requires no processing from the browser but DHTML does.

School of Computing

Department of Information Technology

UNIT - II

Internet Programming – SIT1302

UNIT II

JAVA SCRIPT

Introduction to JavaScript

Javascript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

Advantages

- **Less server interaction** − You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors** − They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity** − You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces** − You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Data Types

JavaScript allows you to work with three **primitive data types**

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.

Trivial data types

null and **undefined,** each of which defines only a single value.

Composite data type

object

Variables

Variables are declared with the **var** keyword as follows. JavaScript is **untyped** language. This means that a JavaScript variable can hold a value of any data type.

```
<script type="text/javascript">
   <!--
      var money;
      var name;
   //-->
</script>
```

You can also declare multiple variables with the same **var** keyword as follows −

```
<script type="text/javascript">
   <!--
      var money, name;
   //-->
</script>
```

Storing a value in a variable is called **variable initialization**.

```
<script type="text/javascript">
   <!--
      var name = "Ali";
      var money;
      money = 2000.50;
   //-->
</script>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables** − A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables** − A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name.

```
<html>
 <body onload = checkscope();>
    <script type = "text/javascript">
      <!--
        var myVar = "global"; // Declare a global variable
        function checkscope( ) {
          var myVar = "local";  // Declare a local variable
          document.write(myVar);
        }
      //-->
    </script>
  </body>
</html>
```

This produces the following result −

```
local
```

Operators

What is an operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators

+ , - , / , *, %, ++, --

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var a = 33;
        var b = 10;
```

```
        var c = "Test";
        var linebreak = "<br />";

        document.write("a + b = ");
        result = a + b;
        document.write(result);
        document.write(linebreak);

        document.write("a - b = ");
        result = a - b;
        document.write(result);
        document.write(linebreak);

        document.write("a / b = ");
        result = a / b;
        document.write(result);
        document.write(linebreak);

        document.write("a % b = ");
        result = a % b;
        document.write(result);
        document.write(linebreak);

        document.write("a + b + c = ");
        result = a + b + c;
        document.write(result);
        document.write(linebreak);

        a = ++a;
        document.write("++a = ");
        result = ++a;
        document.write(result);
        document.write(linebreak);

        b = --b;
        document.write("--b = ");
        result = --b;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    Set the variables to different values and then try...
  </body>
</html>
```

Output

```
a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
++a = 35
--b = 8
Set the variables to different values and then try...
```

Comparison Operators

==,!=.>,>=,<,<=

The following code shows how to use comparison operators in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write("(a == b) => ");
        result = (a == b);
        document.write(result);
        document.write(linebreak);

        document.write("(a < b) => ");
        result = (a < b);
        document.write(result);
        document.write(linebreak);

        document.write("(a > b) => ");
        result = (a > b);
        document.write(result);
        document.write(linebreak);

        document.write("(a != b) => ");
        result = (a != b);
        document.write(result);
        document.write(linebreak);
```

```
         document.write("(a >= b) => ");
         result = (a >= b);
         document.write(result);
         document.write(linebreak);

         document.write("(a <= b) => ");
         result = (a <= b);
         document.write(result);
         document.write(linebreak);
       //-->
    </script>

    Set the variables to different values and different operators and then try...
  </body>
</html>
```

Output

```
(a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
a <= b) => true
Set the variables to different values and different operators and then try...
```

Logical Operators

&& (logical AND)

|| (logical OR)

! (logical NOT)

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
         var a = true;
         var b = false;
         var linebreak = "<br />";

         document.write("(a && b) => ");
         result = (a && b);
```

```
         document.write(result);
         document.write(linebreak);

         document.write("(a || b) => ");
         result = (a || b);
         document.write(result);
         document.write(linebreak);

         document.write("!(a && b) => ");
         result = (!(a && b));
         document.write(result);
         document.write(linebreak);
       //-->
     </script>


     <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
(a && b) => false
(a || b) => true
!(a && b) => true
Set the variables to different values and different operators and then try...
```

Bitwise Operators

& (Bitwise AND)

| (Bitwise OR)

^ (Bitwise XOR)

~ (Bitwise Not)

<< (Left Shift)

>> (Right Shift)

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
```

```
      <!--
        var a = 2; // Bit presentation 10
        var b = 3; // Bit presentation 11
        var linebreak = "<br />";

        document.write("(a & b) => ");
        result = (a & b);
        document.write(result);
        document.write(linebreak);

        document.write("(a | b) => ");
        result = (a | b);
        document.write(result);
        document.write(linebreak);

        document.write("(a ^ b) => ");
        result = (a ^ b);
        document.write(result);
        document.write(linebreak);

        document.write("(~b) => ");
        result = (~b);
        document.write(result);
        document.write(linebreak);

        document.write("(a << b) => ");
        result = (a << b);
        document.write(result);
        document.write(linebreak);

        document.write("(a >> b) => ");
        result = (a >> b);
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
 (a & b) => 2
(a | b) => 3
(a ^ b) => 1
(~b) => -4
```

```
(a << b) => 16
(a >> b) => 0
Set the variables to different values and different operators and then try...
```

Assignment Operators

JavaScript supports the following assignment operators –

+=,-=,*=,/=,%=

Try the following code to implement assignment operator in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var a = 33;
        var b = 10;
        var linebreak = "<br />";

        document.write("Value of a => (a = b) => ");
        result = (a = b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a += b) => ");
        result = (a += b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a -= b) => ");
        result = (a -= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a *= b) => ");
        result = (a *= b);
        document.write(result);
        document.write(linebreak);

        document.write("Value of a => (a /= b) => ");
        result = (a /= b);
        document.write(result);
        document.write(linebreak);
```

```
        document.write("Value of a => (a %= b) => ");
        result = (a %= b);
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0
Set the variables to different values and different operators and then try...
```

Miscellaneous Operator

There are two miscellaneous operators in JavaScript: the **conditional operator** (? :) and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Sr.No	Operator and Description
1	**? : (Conditional)** If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var a = 10;
        var b = 20;
        var linebreak = "<br />";

        document.write ("((a > b) ? 100 : 200) => ");
        result = (a > b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);

        document.write ("((a < b) ? 100 : 200) => ");
        result = (a < b) ? 100 : 200;
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100
Set the variables to different values and different operators and then try...
```

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"

String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement **typeof** operator.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var a = 10;
        var b = "String";
        var linebreak = "<br />";

        result = (typeof b == "string" ? "B is String" : "B is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);

        result = (typeof a == "string" ? "A is String" : "A is Numeric");
        document.write("Result => ");
        document.write(result);
        document.write(linebreak);
      //-->
    </script>

    <p>Set the variables to different values and different operators and then try...</p>
  </body>
</html>
```

Output

```
Result => B is String
Result => A is Numeric
Set the variables to different values and different operators and then try...
```

Control Statements

It used to control the flow of execution within a program .Two types

1. *Conditional*

Decision Making Statement	*Loop Control Statement*
1. if statement	*1.while*
2. if .. else statement	*2. do – while*
3. if – else – if statement	*3. for*
4. switch	*4. for – in*

2. *Unconditional*
    1. *break*
    2. *continue*
    3. *goto*

if statement

The **if** statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows −

```
if (expression){
   Statement(s) to be executed if expression is true
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

Try the following example to understand how the **if** statement works.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var age = 20;

        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Qualifies for driving
Set the variable to different value and then try...
```

if...else statement:

The **'if...else'** statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

```
if (expression){

  Statement(s) to be executed if expression is true

}

else{

  Statement(s) to be executed if expression is false

}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the 'if' block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var age = 15;

        if( age > 18 ){
          document.write("<b>Qualifies for driving</b>");
        }

        else{
          document.write("<b>Does not qualify for driving</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Does not qualify for driving
Set the variable to different value and then try...
```

if...else if... statement

The **if...else if...** statement is an advanced form of **if…else** that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows −

```
if (expression 1){
  Statement(s) to be executed if expression 1 is true
}
```

```
else if (expression 2){
   Statement(s) to be executed if expression 2 is true
}

else if (expression 3){
   Statement(s) to be executed if expression 3 is true
}

else{
   Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var book = "maths";
        if( book == "history" ){
          document.write("<b>History Book</b>");
        }

        else if( book == "maths" ){
          document.write("<b>Maths Book</b>");
        }

        else if( book == "economics" ){
          document.write("<b>Economics Book</b>");
        }

        else{
          document.write("<b>Unknown Book</b>");
        }
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
```

```
<html>
```

Output

Maths Book
Set the variable to different value and then try...

Switch Statement

The objective of a **switch** statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each **case** against the value of the expression until a match is found. If nothing matches, a **default** condition will be used.

```
switch (expression)
{
  case condition 1: statement(s)
  break;

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

The **break** statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain **break** statement in *Loop Control* chapter.

Example

Try the following example to implement switch-case statement.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var grade='A';
        document.write("Entering switch block<br />");
        switch (grade)
        {
          case 'A': document.write("Good job<br />");
          break;

          case 'B': document.write("Pretty good<br />");
          break;

          case 'C': document.write("Passed<br />");
          break;

          case 'D': document.write("Not so good<br />");
          break;

          case 'F': document.write("Failed<br />");
          break;

          default:  document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Entering switch block
Good job
Exiting switch block
```

```
Set the variable to different value and then try...
```

Looping Statements

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

The while Loop

The most basic loop in JavaScript is the **while** loop which would be discussed in this chapter. The purpose of a **while** loop is to execute a statement or code block repeatedly as long as an **expression** is true. Once the expression becomes **false,** the loop terminates.

Syntax

The syntax of **while loop** in JavaScript is as follows −

```
while (expression){
  Statement(s) to be executed if expression is true
}
```

Example

Try the following example to implement while loop.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var count = 0;
        document.write("Starting Loop ");

        while (count < 10){
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        document.write("Loop stopped!");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
```

```
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Current : 9
Loop stopped!
Set the variable to different value and then try...
```

The do...while Loop

The **do...while** loop is similar to the **while** loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is **false**.

Syntax

The syntax for **do-while** loop in JavaScript is as follows −

```
do{
   Statement(s) to be executed;
} while (expression);
```

Example

Try the following example to learn how to implement a **do-while** loop in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
```

```
        var count = 0;

        document.write("Starting Loop" + "<br />");
        do{
          document.write("Current Count : " + count + "<br />");
          count++;
        }

        while (count < 5);
        document.write ("Loop stopped!");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
Set the variable to different value and then try...
```

For Loop

The '**for**' loop is the most compact form of looping. It includes the following three important parts −

- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The **test statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Syntax

The syntax of for loop is JavaScript is as follows −

```
for (initialization; test condition; iteration statement){

   Statement(s) to be executed if test condition is true

}
```

Example

Try the following example to learn how a for loop works in JavaScript.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var count;
        document.write("Starting Loop" + "<br />");

        for(count = 0; count < 10; count++){
          document.write("Current Count : " + count );
          document.write("<br />");
        }

        document.write("Loop stopped!");
      //-->
    </script>

    <p>Set the variable to different value and then try...</p>
  </body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!
```

```
Set the variable to different value and then try...
```

For – in

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

```
for (variablename in object){
  statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to implement 'for-in' loop. It prints the web browser's **Navigator** object.

```
<html>
  <body>

    <script type="text/javascript">
      <!--
        var aProperty;
        document.write("Navigator Object Properties<br /> ");

        for (aProperty in navigator) {
          document.write(aProperty);
          document.write("<br />");
        }
        document.write ("Exiting from the loop!");
      //-->
    </script>

    <p>Set the variable to different object and then try...</p>
  </body>
</html>
```

Output

```
Navigator Object Properties
serviceWorker
```

```
webkitPersistentStorage
webkitTemporaryStorage
geolocation
doNotTrack
onLine
languages
language
userAgent
product
platform
appVersion
appName
appCodeName
Exiting from the loop!
Set the variable to different object and then try...
```

Functions

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here.

```
<script type="text/javascript">
  <!--
    function functionname(parameter-list)
    {
      statements
    }
```

```
  //-->

</script>
```

Example

Try the following example. It defines a function called sayHello that takes no parameters −

```
<script type="text/javascript">

  <!--

    function sayHello()

    {

      alert("Hello there");

    }

  //-->

</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
  <head>

    <script type="text/javascript">
      function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>

  </head>
  <body>
    <p>Click the following button to call the function</p>

    <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
    </form>

    <p>Use different text in write method and then try...</p>
  </body>
```

```
</html>
```

Output

Hello there!

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our **sayHello** function here. Now it takes two parameters.

```
<html>
  <head>

    <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
    </script>

  </head>
  <body>
    <p>Click the following button to call the function</p>

    <form>
      <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
    </form>

    <p>Use different parameters inside the function and then try...</p>
  </body>
</html>
```

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
  <head>

    <script type="text/javascript">
      function concatenate(first, last)
      {
        var full;
        full = first + last;
        return full;
      }

      function secondFunction()
      {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
      }
    </script>

  </head>

  <body>
    <p>Click the following button to call the function</p>

    <form>
      <input type="button" onclick="secondFunction()" value="Call Function">
    </form>

    <p>Use different parameters inside the function and then try...</p>

  </body>
</html>
```

Objects

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

- **Encapsulation** − the capability to store related information, whether data or methods, together in an object.

- **Aggregation** − the capability to store one object inside another object.
- **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
- **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes.Attribute is considered to be property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

```
objectName.objectProperty = propertyValue;
```

For example − The following code gets the document title using the **"title"**property of the **document** object.

```
var str = document.title;
```

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

```
var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");
```

Array Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** object −

```
var fruits = new Array( "apple", "orange", "mango" );
```

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows −

```
var fruits = [ "apple", "orange", "mango" ];
```

You will use ordinal numbers to access and to set values inside an array as follows.

```
fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element
```

Array Methods

Here is a list of the methods of the Array object along with their description.

Sr.No	Method & Description
1	**concat()** Returns a new array comprised of this array joined with other array(s) and/or value(s).
2	**forEach()** Calls a function for each element in the array.
3	**indexOf()** Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found.
4	**join()** Joins all elements of an array into a string.
5	**lastIndexOf()** Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found.
6	**pop()** Removes the last element from an array and returns that element.
7	**push()** Adds one or more elements to the end of an array and returns the new length of the array.
8	**reverse()** Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
9	**shift()**

	Removes the first element from an array and returns that element.
10	**slice()** Extracts a section of an array and returns a new array.
11	**sort()** Sorts the elements of an array
12	**splice()** Adds and/or removes elements from an array.
13	**toString()** Returns a string representing the array and its elements.

String Object

The **String** object lets you work with a series of characters; As JavaScript automatically converts between string primitives and String objects .

Syntax

Use the following syntax to create a String object −

```
var val = new String(string);
```

The **String** parameter is a series of characters that has been properly encoded.

String Methods

Here is a list of the methods available in String object along with their description.

Sr.No	Method & Description
1	**charAt()** Returns the character at the specified index.

2	**charCodeAt()** Returns a number indicating the Unicode value of the character at the given index.
3	**concat()** Combines the text of two strings and returns a new string.
4	**indexOf()** Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
5	**lastIndexOf()** Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found.
6	**localeCompare()** Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order.
7	**match()** Used to match a regular expression against a string.
8	**replace()** Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring.
9	**search()** Executes the search for a match between a regular expression and a specified string.
10	**slice()** Extracts a section of a string and returns a new string.
11	**split()** Splits a String object into an array of strings by separating the string into substrings.

12	**substr()** Returns the characters in a string beginning at the specified location through the specified number of characters.
13	**substring()** Returns the characters in a string between two indexes into the string.
14	**toLocaleLowerCase()** The characters within a string are converted to lower case while respecting the current locale.
15	**toLocaleUpperCase()** The characters within a string are converted to upper case while respecting the current locale.
16	**toLowerCase()** Returns the calling string value converted to lower case.
17	**toString()** Returns a string representing the specified object.
18	**toUpperCase()** Returns the calling string value converted to uppercase.
19	**valueOf()** Returns the primitive value of the specified object.

Math Object

The **math** object provides you properties and methods for mathematical constants and functions.

All the properties and methods of **Math** are static and can be called by using Math as an object without creating it.

Syntax

The syntax to call the properties and methods of Math are as follows

```
var pi_val = Math.PI;

var sine_val = Math.sin(30);
```

Math Methods

Here is a list of the methods associated with Math object and their description

Sr.No	Method & Description
1	**abs()** Returns the absolute value of a number.
2	**acos()** Returns the arccosine (in radians) of a number.
3	**asin()** Returns the arcsine (in radians) of a number.
4	**atan()** Returns the arctangent (in radians) of a number.
5	**atan2()** Returns the arctangent of the quotient of its arguments.
6	**ceil()** Returns the smallest integer greater than or equal to a number.
7	**cos()** Returns the cosine of a number.
8	**exp()** Returns E^N, where N is the argument, and E is Euler's constant, the base of the natural logarithm.
9	**floor()** Returns the largest integer less than or equal to a number.

10	**log()** Returns the natural logarithm (base E) of a number.
11	**max()** Returns the largest of zero or more numbers.
12	**min()** Returns the smallest of zero or more numbers.
13	**pow()** Returns base to the exponent power, that is, base exponent.
14	**random()** Returns a pseudo-random number between 0 and 1.
15	**round()** Returns the value of a number rounded to the nearest integer.
16	**sin()** Returns the sine of a number.
17	**sqrt()** Returns the square root of a number.
18	**tan()** Returns the tangent of a number.
19	**toSource()** Returns the string "Math".

Boolean Object

The **Boolean** object represents two values, either "true" or "false". If *value*parameter is omitted or is 0, -0, null, false, **NaN,** undefined, or the empty string (""), the object has an initial value of false.

Syntax

Use the following syntax to create a **boolean** object.

```
var val = new Boolean(value);
```

Boolean Methods

Here is a list of the methods of Boolean object and their description.

Sr.No	Method & Description
1	**toSource()** Returns a string containing the source of the Boolean object; you can use this string to create an equivalent object.
2	**toString()** Returns a string of either "true" or "false" depending upon the value of the object.
3	**valueOf()** Returns the primitive value of the Boolean object.

Global Object

- It provides top-level properties and methods that are not part of any other object
- You cannot create an instance of the Global object
- These methods are called directly and are not prefixed with “global”
- Globally Available Methods
    1. escape()
    2. unescape()
    3. eval()
    4. isFinite()
    5. isNaN()

6. parseFloat()
7. parseInt()

escape()

- Takes a string and returns a string where all non-alphanumeric characters such as spaces, tabs, and special characters have been replaced with their hexadecimal equivalents in the form %xx

Example:

```
var aString="O'Neill & Sons";
// aString = "O'Neill & Sons"
aString = escape(aString);
// String="O%27Neill%20%26%20Sons"
```

unescape ()

- Takes a hexadecimal string value containing some characters of the form %xx and returns the ISO-Latin-1 ASCII equivalent of the passed values

Example:

```
var aString="O%27Neill%20%26%20Sons";
aString = unescape(aString);
// aString = "O'Neill & Sons"
aString = unescape("%64%56%26%23");
// aString = "dV&#"
```

eval()

- Takes a string and executes it as JavaScript code

Example:

```
var x;
var aString = "5+9";
x = aString;
// x contains the string "5+9"
x = eval(aString);
// x will contain the number 14
```

isFinite()

- Returns a Boolean indicating whether its number argument is finite or not

Example:

```
var x;
x = isFinite('56');
// x is true
x = isFinite(Infinity);
// x is false
```

isNaN()

- Returns a Boolean indicating whether its number argument is NaN

Example:

```
var x;
x = isNaN('56');
// x is False
x = isNaN(0/0);
// x is true
x = isNaN(NaN);
// x is true
```

parseInt()

- Converts the string argument to an integer and returns the value
- If the string cannot be converted, it returns NaN
- This method also should handle strings starting with numbers, but other mixed strings will not be converted

Example:

```
var x;
x = parseInt("-53");    // x is -53
x = parseInt("33.01568");       // x is 33
```

```
x = parseInt("47.6k-red-dog");          // x is 47
x = parseInt("a567.34");        // x is NaN
x = parseInt("won't work");    // x is NaN
```

parseFloat()

- Converts the string argument to a floating-point number and returns the value
- If the string cannot be converted, it returns NaN
- The method should handle strings starting with numbers, but other mixed strings will not be converted

Example:

```
var x;
x = parseFloat("33.01568");   // x is 33.01568
x = parseFloat("47.6k-red-dog");       // x is 47.6
x = parseFloat("a567.34");     // x is NaN
x = parseFloat("won't work");// x is NaN
```

Date Object

- The Date object provides a sophisticated set of methods for manipulating dates and times
- To understand the relationship between Greenwich Mean Time (GMT), Coordinated Universal Time (UTC), and local time zones
- Creating Dates using Date() constructor

Syntax:

```
var myDate=new Date();
```

Manipulating Dates

- JavaScript provides a comprehensive set of **get** and **set** methods to read and write each field of a date
- The methods are
  - ✓ **getDate()**, **setDate()**,
  - ✓ **getMonth()**, **setMonth()**,

- ✓ **getHours()**, **setHours()**,
- ✓ **getMinutes()**, **setMinutes()**,
- ✓ **getTime()**, **setTime**, and so on.

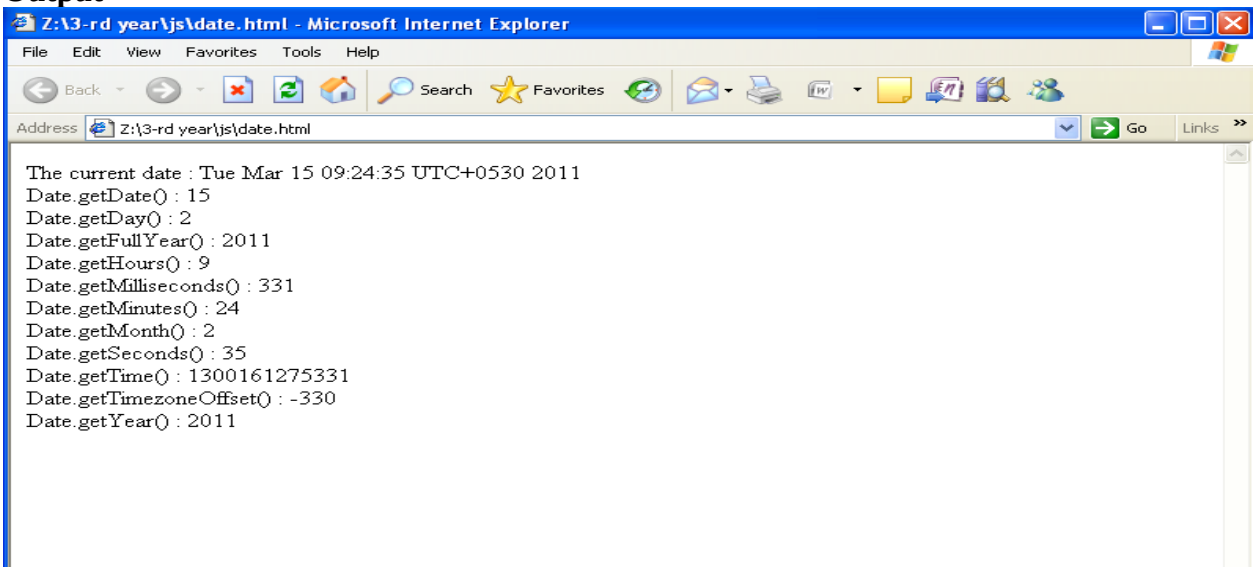
- In addition, UTC versions of all these methods are also included:
  - ✓ **getUTCMonth()**, **setUTCMonth()**
  - ✓ **getUTCHours()**, **setUTCHours()**, and so forth.
- One set of methods requires a special comment:
  - ✓ **getDay() and setDay().**
  - ✓ **These are used to manipulate the day of the week that is stored as an integer from 0 (Sunday) to 6 (Saturday).**

Example

```
<html>
  <body>
   <script type="text/javascript">
    var today = new Date();
    document.write("The current date : "+today+"<br />");
    document.write("Date.getDate() : "+today.getDate()+"<br/>");
    document.write("Date.getDay() : "+today.getDay()+"<br/>");
    document.write("Date.getFullYear() : "+today.getFullYear()+"<br/>");
    document.write("Date.getHours() : "+today.getHours()+"<br/>");
    document.write("Date.getMilliseconds() : "+today.getMilliseconds()+"<br/>");
    document.write("Date.getMinutes() : "+today.getMinutes()+"<br/>");
    document.write("Date.getMonth() : "+today.getMonth()+"<br/>");
    document.write("Date.getSeconds() : "+today.getSeconds()+"<br/>");
    document.write("Date.getTime() : "+today.getTime()+"<br/>");
```

```
        document.write("Date.getTimezoneOffset() : "+today.getTimezoneOffset()+"<br/>");
        document.write("Date.getYear() : "+today.getYear()+"<br/>");
        </script>
      </body>
    </html>
```

Output



Number Object

- **Number is the built-in object corresponding to the primitive number data type**

Syntax

The syntax for creating a **number** object is as follows −

```
var val = new Number(number);
```

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Example:

var x = new Number(); // returns x=0

var y = new Number(17.5); // y=17.5

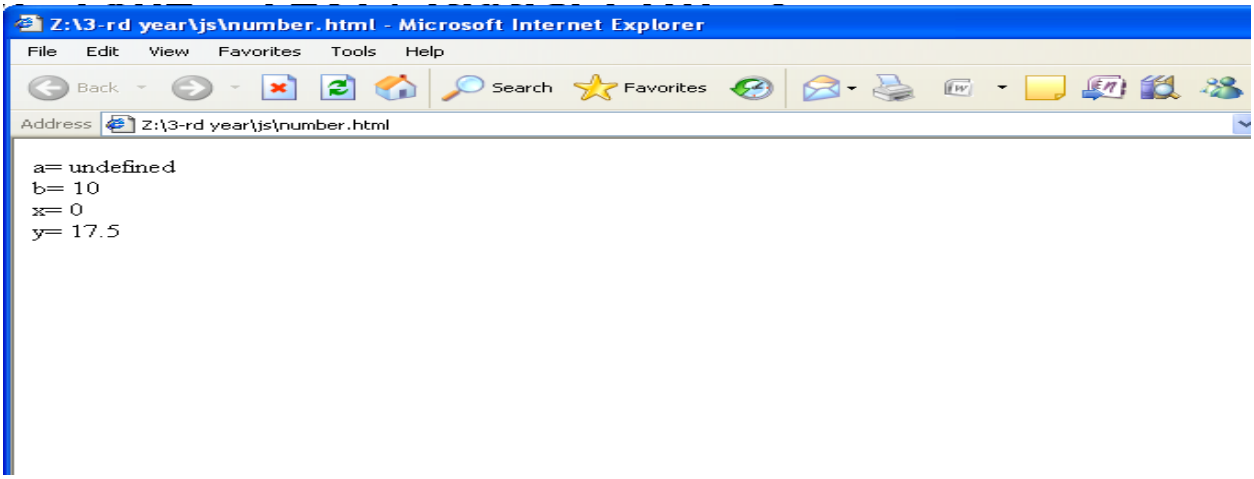
Properties of the Number Object

Property	Value
Number.MAX_VALUE	Largest magnitude representable
Number.MIN_VALUE	Smallest magnitude representable
Number.POSITIVE_INFINITY	The special value **Infinity**
Number.NEGATIVE_INFINITY	The special value **-Infinity**
Number.NaN	The special value **NaN**

Example

```
<html>
  <body>
   <script type="text/javascript">
                    var a;
                    document.write("a= "+a+"<br>");
                    b=10;
                    document.write("b= "+b+"<br>");
                    var x = new Number();
                    document.write("x= "+x+"<br>");
                    var y = new Number(17.5);
                    document.write("y= "+y);
          </script>
 </body>
</html>
```

Output



.

Windows and Frames

- Window is an object that corresponds to the window that displays a Web page
- Such a window can be created dynamically

Properties

- ✓ These properties are assigned when creating the window.
- ✓ They value *yes* or *no* according to the state enabled or not.
- ✓ **scrollbars** -*yes* if scrollbars appear and *no* otherwise.
- ✓ **statusbar** -*yes* or *no,* depending on whether one shows the status bar or not.
- ✓ **toolbar** -*yes* or *no,* depending on whether one displays the toolbar or not.
- ✓ **menubar** -presenting a menu or not.
- ✓ **resizable** -to be able to change the size or not. The lower right corner is designed accordingly.
- ✓ **directories** -including buttons for favorites.

Attributes of window

- ✓ These attributes can be read only for some, their value can be assigned otherwise.
- ✓ **frames[]**-The frames in the window. Read only.
- ✓ **Length**-Number of frames. Read only.

- ✓ **Name-**Name of the window.
- ✓ **Status-**Text of the status bar **defaultStatus-**Default text in statusbar.
- ✓ **Closed-**State closed or not.
- ✓ **Opener-**Reference on the window that opened this window. Example: x = window.opener;
- ✓ **Parent-**The window parent of a window. Example: x = mywin.parent;
- ✓ **Top-**The parent of the highest level.

Objects in window

- ✓ These objects have their own attributes and methods that are not detailed here but they have their own page.
- ✓ **document-**Refers to a page, that is contained in the window. *Document*.
- ✓ **history-**List of pages previously viewed in the same window. *History*.
- ✓ **location-**Designates the URL of a page that contains the window. *Location*.
- ✓ **screen-**The screen and its properties: width, height, availWidth, availHeight, colorDepth.

Window Creation

- The **Window** object methods **open()** and **close()** are used to create and destroy a window, respectively.
- When you open a window, you can set its URL, name, size, buttons, and other attributes, such as whether or not the window can be resized**.**

  Syntax:

  - **window.open(**url, name, features, replace**)** ;
- Where
  - *url* is a URL that indicates the document to load into the window.
  - *name* is the name for the window
  - *features* is a comma-delimited string that lists the features of the window.
  - *replace* is an optional Boolean value (**true** or **false**) that indicates if the URL specified should replace the window's contents or not.

Example

```
secondwindow = open("http://www.yahoo.com", "yahoo",
"height=300,width=200, scrollbars=yes");
```

Example Program

```
<html>
    <head>
        <script language="JavaScript">
            function openWin()
            {
                myWin = open("window.html");
            }
            function closeWin()
```

```
                {
                        myWin.close();
                }
        </script>
    </head>
    <body>
    <form>
        <input type = "button" value = "Open new Window" onClick="openWin()">
        <input type = "button" value = "Close Window" onClick="closeWin()">
    </form>
    </body>
</html>
```

Forms and Validation

```
<html>
<head><title>Form validation</title>
<script language="javascript">
var s1,s2,s3,s4,s5,s6,s7,s8;
function fun1()
{
if(form1.name.value=="")
{
alert("enter the name");
form1.name.focus();
}
else
{
var l=form1.name.value.length;
```

```
s1=new String(form1.name.value);
for(i=0;i<l;i++)
{
if(!((s1.charAt(i)>='a'||s1.charAt(i)>='A') && (s1.charAt(i)<='z'||s1.charAt(i)<='Z')))
{
alert("Please enter the correct name");
form1.name.focus();
break;
}
}
}
}
function fun2()
{
if(form1.email.value=="")
{
alert("enter the email");
form1.email.focus();
}
else
{
var c1=0;
var c2=0;
var l=form1.email.value.length;
s2=new String(form1.email.value);
for(i=0;i<=l;i++)
{
if(s2.charAt(i)=='@')
{
c1=c1+1;
var j=i;
}
}
if(c1==1)
{
for(i=j;i<=l;i++)
{
if(s2.charAt(i)=='.')
```

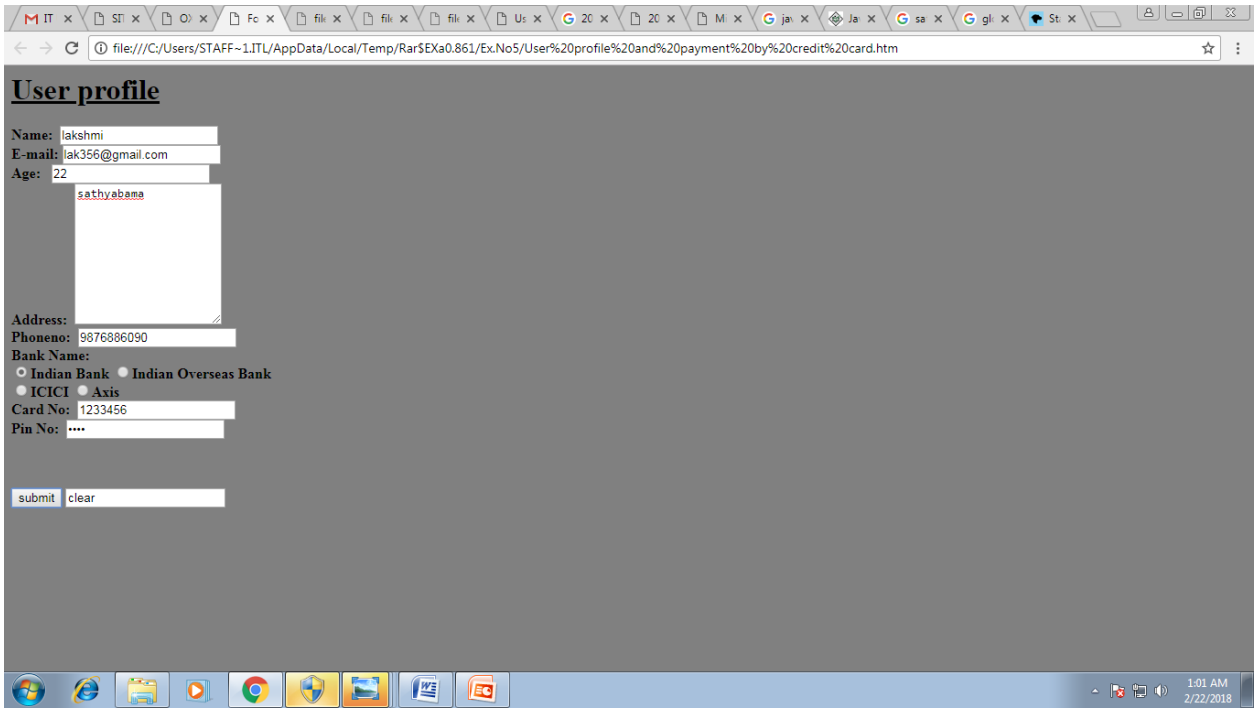
```
c2=c2+1;
}
}
if(!((c1==1) && (c2==1)))
{
alert("enter the correct e-mail id");
form1.email.focus();
}
}
}
function fun3()
{
s3=form1.age.value;
if((s3=="")||isNaN(s3)) //isNaN() function determines whether a value is an illegal number
{
alert("enter the age");
form1.age.focus();
}
}
function fun4()
{
s4=form1.addrs.value;
if(s4=="")
{
alert("enter the address");
form1.addrs.focus();
}
}
function fun5()
{
s5=form1.phno.value;
if((s5=="") || isNaN(s5))
{
alert("Invalid phone number");
form1.phno.focus();
}
}
function fun6()
```

```
{
var tag1=0;
for(var i=0;i<form1.bank.length;i++)
{
if (form1.bank[i].checked==true)
{
s6=form1.bank[i].value;
tag1=1;
}
}
if(!(tag1==1))
{
alert("select ur Bank");
}
}
function fun7()
{
s7=form1.card.value;
if((s7=="") || isNaN(s7))
{
alert("Invalid card number");
form1.card.focus();
}
}
function fun8()
{
s8=form1.pin.value;
var l=form1.pin.value.length;
if(((s8=="") || isNaN(s8))&&(l==3))
{
alert("Invalid card number");
form1.pin.focus();
}
}
function fun9()
{
w1=window.open('Order Conformation.html');
```

```
w1.document.writeln("<h1><center>");
w1.document.writeln("Order Conformation Details");
w1.document.writeln("</h1></center>");
w1.document.writeln("<br>");
w1.document.writeln("Name:",s1);
w1.document.writeln("<br>");
w1.document.writeln("Email:",s2);
w1.document.writeln("<br>");
w1.document.writeln("age:",s3);
w1.document.writeln("<br>");
w1.document.writeln("Address:",s4);
w1.document.writeln("<br>");
w1.document.writeln("Phone:",s5);
w1.document.writeln("<br>");
w1.document.writeln("Bank Name:",s6);
w1.document.writeln("<br>");
w1.document.writeln("Card Number:",s7);
w1.document.writeln("<br>");
}
</script></head>
<body bgcolor="gray" >
<h1><b><u> User profile </u></b></h1>
<form name="form1" method="get">
<b>
Name:  <input type="text" name="name" maxlen="5"><br>
E-mail: <input type="text" name="email" onfocus="fun1()"><br>
Age:   <input type="text" name="age" onfocus="fun2()"><br>
Address:  <textarea onfocus="fun3()" rows=10 cols=20
name=addrs></textarea><br>
Phoneno:  <input type="text" name="phno" onfocus="fun4() "><br>
Bank Name:<br>
<input type="radio" onfocus="fun5()" name="bank" value="Indian Bank">Indian Bank
<input type="radio" onfocus="fun5()" name="bank" value="Indian Overseas Bank">Indian
Overseas Bank <br>
<input type="radio" onfocus="fun5()" name="bank"value="ICICI">ICICI
<input type="radio" onfocus="fun5()" name="bank" value="Axis">Axis <br>
Card No:  <input type="text" name="card" onfocus="fun6() "><br>
Pin No:  <input type="password" name="pin"onfocus="fun7()
"><br><br><br><br>
```

```
<input type="button" value="submit" onfocus="fun8()" onclick="fun9()">
<input type=”reset “ value=clear>
</b>
</form>
</body>
</html>
```

Output



School of Computing

Department of Information Technology

UNIT - III

Internet Programming – SIT1302

UNIT III

XML TECHNOLOGIES

Introduction to XML

What is xml?

- ✓XML stands for e**X**tensible **M**arkup **L**anguage.
- ✓XML is designed to transport and store data.
- ✓XML is important to know, and very easy to learn.
- ✓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	HTML
XML was designed to transport and store data	HTML was designed to display data
With focus on what data is	With focus on how data looks

How Can XML be Used?

- ✓XML Separates Data from HTML
- ✓XML Simplifies Data Sharing
- ✓XML Simplifies Data Transport
- ✓XML Simplifies Platform Changes
- ✓XML Makes Your Data More Available
- ✓XML is Used to Create New Internet Languages

XML Tree

- ✓XML documents form a tree structure that starts at "the root" and branches to "the leaves".
- ✓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.

Syntax:

<root>

<child>

<subchild>.....</subchild>

</child>

</root>

Example:

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

<note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

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

Example: book.xml

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

<book category="cooking">

<title lang="english">Everyday Italian</title>

<author>Giada De Laurentiis</author> <year>2005</year>

<price>30.00</price>

</book>

<book category="children">

<title lang="english">Harry Potter</title> <author>J K. Rowling</author> <year>2005</year> <price>29.99</price>

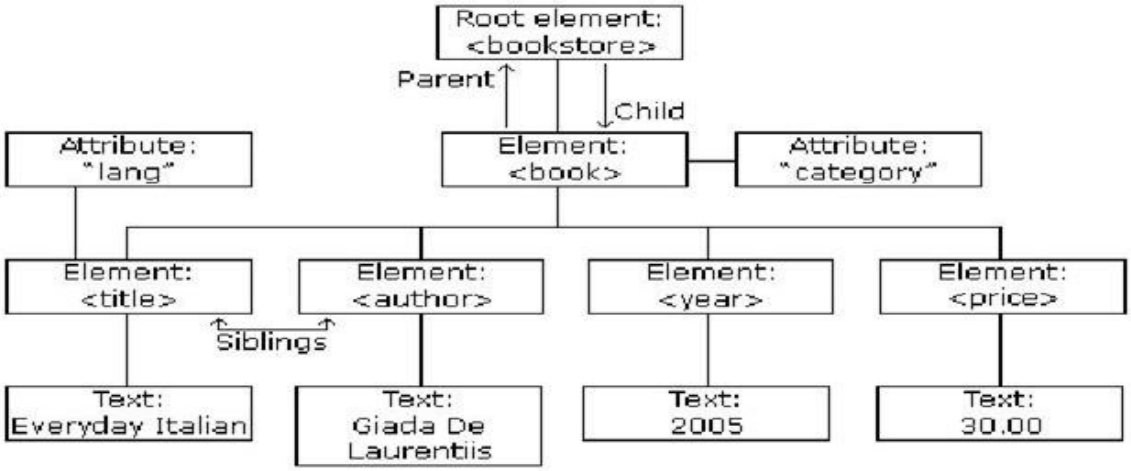
</book>

<book category="web">

<title lang="english">Learning XML</title>

<author>Erik T. Ray</author>

<year>2003</year>



<price>39.95</price>

</book>

</bookstore>

Fig 3.1 XML Structure

XML Rules

- ✓All XML Elements Must Have a Closing Tag
- ✓XML Tags are Case Sensitive
- ✓XML Elements Must be Properly Nested
- ✓XML Documents Must Have a Root Element
- ✓XML Attribute Values Must be Quoted
- ✓Character and Entity References
- ✓Comments in XML
- ✓White-space is Preserved in XML

3.2 XML Elements

- ✓ 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, simple text or a mixture of both. Elements can also have attributes

```
<bookstore>
    <book category="Prog">
        <title>JAVA</title>
        <author>James Gosling</author>
        <year>2005</year>
        <price>RS. 500.00</price>
    </book>
</bookstore>
```

- ✓ In the example above, <bookstore> and <book> have **element contents**, because they contain other elements. <author> has **text content** because it contains text.
- ✓ In the example above only <book> has an attribute (category="Prog").

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, <book_title>
- ✓ Avoid "-" characters, Avoid "." characters, Avoid ":" characters

XML Attributes

- XML elements can have attributes in the start tag.
- Attributes provide additional information about elements.
- XML Attributes Must be Quoted.

✓ Attribute values must always be enclosed in quotes, but either single or double quotes can be used.

✓ For a person's sex, the person tag can be written like this

<person sex="female"> or <person sex='female'>

Example

```
<person sex="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
```

XML Attributes vs Elements

Take a look at these examples:

```
<person sex="female">
    <firstname>Anna</firstname>
    <lastname>Smith</lastname>
</person>
<person>
 <sex>female</sex>
<firstname>Anna</firstname>
<lastname>Smith</lastname>
</person>
```

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.

Well Formed XML Documents

- XML with correct syntax is "Well Formed" XML.
- XML validated against a DTD is "Valid" XML.

- A "Well Formed" XML document has correct XML syntax
- 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

Xml namespaces

- XML Namespaces provide a method to avoid element name conflicts.

Name Conflicts

- In XML, element names are defined by the developer. This often results in a conflict
- when trying to mix XML documents from different XML applications

- This XML carries HTML table information:

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

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

```
<table>
```

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

- If these XML fragments were added together, there would be a name conflict.
- Both contain a <table> element, but the elements have different content and meaning.
- An XML parser will not know how to handle these differences

Solving the Name Conflict Using a Prefix

- Name conflicts in XML can easily be avoided using a name prefix.

- This XML carries information about an HTML table, and a piece of furniture:

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

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

- In the example above, there will be no conflict because the two <table> elements have different names.

XML Namespaces - The xmlns Attribute

- When using prefixes in XML, a so-called **namespace** for the prefix must be defined.
- The namespace is defined by the **xmlns attribute** in the start tag of an element.
- The namespace declaration has the following syntax. xmlns:*prefix*="*URI*".

```
<root>
```

<h:table xmlns:h="http://www.w3.org/TR/html4/"> <h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture"> <f:name>African Coffee Table</f:name>

<f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

- Namespaces can be declared in the elements where they are used or in the XML root element: <root xmlns:h="http://www.w3.org/TR/html4/" xmlns:f="http://www.w3schools.com/furniture">

<h:table>

<h:tr>

<h:td>Apples</h:td>

<h:td>Bananas</h:td>

</h:tr>

</h:table>

<f:table>

<f:name>African Coffee Table</f:name> <f:width>80</f:width>

<f:length>120</f:length>

</f:table>

</root>

Uniform Resource Identifier (URI)

- A **Uniform Resource Identifier** (URI) is a string of characters which identifies an Internet Resource.
- The most common URI is the **Uniform Resource Locator** (URL) which identifies an Internet domain address. Another, not so common type of URI is the **Universal Resource Name**(URN)

Internal DTD Declaration

- The DTD is declared inside the XML file
- It should be wrapped in a DOCTYPE definition

Syntax

```
<!DOCTYPE root-element [element-declarations]>
```

Example

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

<body>Don't forget me this weekend</body>

</note>Explanation

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 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 body** defines the body element to be of type "#PCDATA"

External DTD Declaration

- The DTD is declared in an external file
- It should be wrapped in a DOCTYPE definition

Syntax:

<!DOCTYPE root-element SYSTEM "filename">

Example

<?xml version="1.0"?>

<!DOCTYPE note SYSTEM "note.dtd"> <note>

<to>Tove</to>

<from>Jani</from>

<heading>Reminder</heading>

<body>Don't forget me this weekend!</body>

</note>

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)>

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

Document Type Definition Attribute

- Attributes are additional information associated with an element type
- They are intended for interpretation by an application.
- The ATTLIST declarations identify which element types may have attributes, what type of attributes they may be, and what the default value of the attributes are.

Syntax:

<!ATTLIST element_name attribute_name attribute_type default_value>

.

.

.

<element attribute_name="attribute_value">

where:-

element_name: name of the element to which the attribute applies.

Example:

<?xml version="1.0"?>

<!DOCTYPE image [<!ELEMENT image EMPTY>

<!ATTLIST image height CDATA #REQUIRED> <!ATTLIST image width CDATA #REQUIRED>

]>

<image height="32" width="32"/>

Rules:

- ✓ All attributes used in an XML document must be declared in the Document Type Definition (DTD) using an Attribute- List Declaration .
- ✓ Attributes may only appear in start or empty tags .
- ✓ The keyword ATTLIST must be in upper case

Default values

- ✓ The "default_value" signifies whether an attribute is required or not, and if not, what default value should be displayed.
- ✓ The possible default values are listed in below.

Default Value:	Description:
#REQUIRED	The attribute must always be included .
#IMPLIED	The attribute does not have to be included.
#FIXED "Default_Value"	The attribute must always have the default value that is specified . If the attribute is not physically added to the element tag in the XML document, the XML processor will behave as though the default value does exist.

Table 3.1 Attribute types

Attribute types

There are three main attribute types.

1. A string type
2. A tokenized types
3. A enumerated types

1. A string type:-

CDATA :-

CDATA stands for character data, that is, text that does not form markup.

Example:

```
<?xml version="1.0"?>
<!DOCTYPE image [
<!ELEMENT image EMPTY>
<!ATTLIST image height CDATA #REQUIRED>

<!ATTLIST image width CDATA #REQUIRED> ]>
<image height="32" width="32"/>
```

2. Tokenized Attribute Type

1. ID :-

ID is a unique identifier of the attribute. IDs of a particular value should not appear more than once in an XML document .

An element type may only have one ID attribute . An ID attribute can only have an #IMPLIED or #REQUIRED default value . The first character of an ID value must be a letter, '_', or ':'. Example:

```
<?xml version="1.0"?> <!DOCTYPE student_name [
<!ELEMENT student_name (#PCDATA)>
<!ATTLIST student_name student_no ID #REQUIRED>
]>
<student_name student_no="a9216735">Jo Smith</student_name>
```

2.IDREF

IDREF is used to establish connections between elements. The IDREF value of the attribute must refer to an ID value

declared elsewhere in the document . The first character of an ID value must be a letter, '_', or ':' .

Example:

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

```
<!DOCTYPE lab_group [

<!ELEMENT lab_group (student_name)*>

<!ELEMENT student_name (#PCDATA)>

<!ATTLIST student_name student_no ID #REQUIRED> <!ATTLIST student_name
tutor_1 IDREF #IMPLIED> <!ATTLIST student_name tutor_2 IDREF #IMPLIED>
]>

<lab_group>

<student_name student_no="a8904885">Alex Foo</student_name> <student_name
student_no="a9011133">Sarah Bar</student_name>

<student_name student_no="a9216735" tutor_1="a9011133"
tutor_2="a8904885">Jo Smith</student_name>
</lab_group>
```

3.IDREFS

Allows multiple ID values separated by whitespace.

4.ENTITY

ENTITYs are used to reference data that act as an abbreviation or can be found at an external location. The first character of an ENTITY value must be a letter, '_', or ':' .

Example

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

<!DOCTYPE experiment_a [

<!ELEMENT experiment_a (results)*>

<!ELEMENT results EMPTY>

<!ATTLIST results image ENTITY #REQUIRED>

<!ENTITY a SYSTEM "http://www.university.com/results/experimenta/a.gif"> ]>

<experiment_a> <results image="a"/>
```

<experiment_a>

5.ENTITIES

Allows multiple ENTITY names
separated by whitespace. Example:

<?xml version="1.0“ ?>

<!DOCTYPE experiment_a [

<!ELEMENT experiment_a (results)*> <!ELEMENT results EMPTY>

<!ATTLIST results images ENTITIES #REQUIRED>

<!ENTITY a1 SYSTEM "http://www.university.com/results/experimenta/a1.gif">

<!ENTITY a2 SYSTEM "http://www.university.com/results/experimenta/a2.gif">

<experiment_a>

<results images="a1 a2 "/>

</experiment_a>

6.NMTOKEN

The first character of an NMTOKEN value must be a letter, digit, '.', '-', '_', or ':'.

Example:-

<?xml version="1.0"?> <!DOCTYPE student_name [

<!ELEMENT student_name (#PCDATA)>

<!ATTLIST student_name student_no NMTOKEN #REQUIRED>

]>

<student_name student_no="9216735">Jo Smith</student_name>

7.NMTOKENS

Allows multiple NMTOKEN names separated by whitespace .

3. Enumerated attribute type

1. NOTATION

NOTATIONs are useful when text needs to be interpreted in a particular way, for example, by another application. The first character of a NOTATION name must be a letter, '_', or ':' .

Example:-

<?xml version="1.0"?> <!DOCTYPE code [

<!ELEMENT code (#PCDATA)> <!NOTATION vrml PUBLIC "VRML 1.0">

<!ATTLIST code lang NOTATION (vrml) #REQUIRED>]>

<code lang="vrml">Some VRML instructions</code>

2Enumerated

Enumerated attribute types allow you to make a choice between different attribute values. The first character of an Enumerated value must be a letter, digit, '.', '-', '_', or ':' .

Example

<?xml version="1.0"?>

<!DOCTYPE payment [<!ELEMENT payment EMPTY>

<!ATTLIST payment mode (cash|cheque) “cash” #REQUIRED>]>

<payment mode=”cash”>

Or

<payment mode=”cheque”>

What is CSS?

- CSS stands for Cascading Style Sheets
- Most popular way of formatting HTML files so that they can be easily maintained and updated
- CSS also well supported method of formatting XML document for web browsers

Basic CSS statement

- For creating style sheet, Identify the specific rules and values for each element's available properties

Example:

Selector { property : value}

- Selector is the element identifier
- Property is the name of one of the predefined CSS properties.
- Value is one of the predefined value types for specific property. company { background.color:LIGHTBLUE; font.size:12pt; }

To apply a series of properties values to a single element

Syntax:

Selector { property : value; property: value;property: value;…..property: value}

Example

company { background.color:LIGHTBLUE; font.size:12pt; }

Setting multiple properties values for a multiple elements

Syntax:

Selector1,selector2,selector3 {property:value; property: value}

Example:

head, company {font-family: "comic sans MS", sans-serif; font-size:16pt; font-style:normal}

Adding css to your document

- Style sheets can only be added to xml documents through the use of a processing instruction that references the external CSS file
- CSS using a system of pattern matching to determine which CSS rules are applied to which elements that are found within the document

Syntax:

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

- A CSS style contains a list of rules.
- Each rules gives the names of the element
- It applies the styles to those elements
- Cascading Style Sheets(CSS) is a very simple and straight forward language for applying styles to xml documents
- You can choose fonts, font weight, font size, background color, spacing between paragraph and etc…
- All style information is placed in a separate document called a style sheet
- A single xml document can be formatted different ways by charging the style sheet
- CSS is a declarative language in 1996 is a standard adding information about style properties such as fonts and borders to html elements

Example:

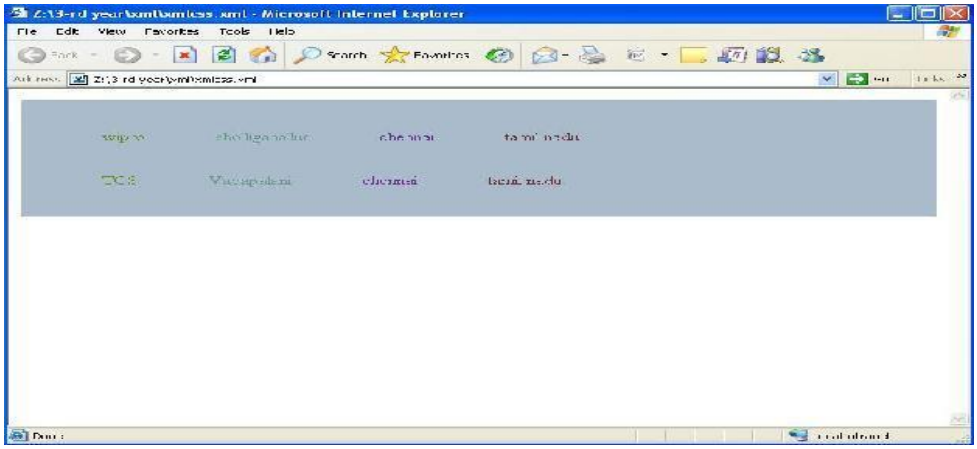
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="catalog.css"?> <company>
<IT>
<name>wipro</name>
<location>sholliganallur</location>
<city>chennai</city> <state>tamil nadu</state>
</IT>
<IT>
<name>TCS</name>
<location>Vadapalani</location>
<city>chennai</city> <state>tamil nadu</state>
</IT>
</company>
```

File Name :Catalog.css

company

{
background.color:#a
abbcc; width=100%
}
IT
{
display:block; margin.bottom:30pt; margin.left:30pt
}
name
{
color:#678910; margin.left:20pt; font.size:12pt
}
Location
{
color:#689888; margin.left:20pt; font.size:12pt
}
city
{
color:#682391; margin.left:20pt; font.size:12pt
}
state
{
color:#692931; margin.left=20pt; font.size=12pt
}

Output:-



CSS SELECTOR

CSS rule specifies which element it applies to is called a selector The simple selector is name of the element

Example:

Title { display:block; font-size:16pt;font-weight:bold } Selectors also specify multiple elements

CONTROLLING FONTS

- Font styles – control all aspects of your text appearance from font style, face, family and variants.
- Font-family Used to specify a prioritized list of font family names and / or generic family names
- Allows to identify a specific list of fonts, generally same style and size

Syntax:

Font-family: [[<family-name> | <generic-family>],]* [<family-name> | <generic-family>] | inherit

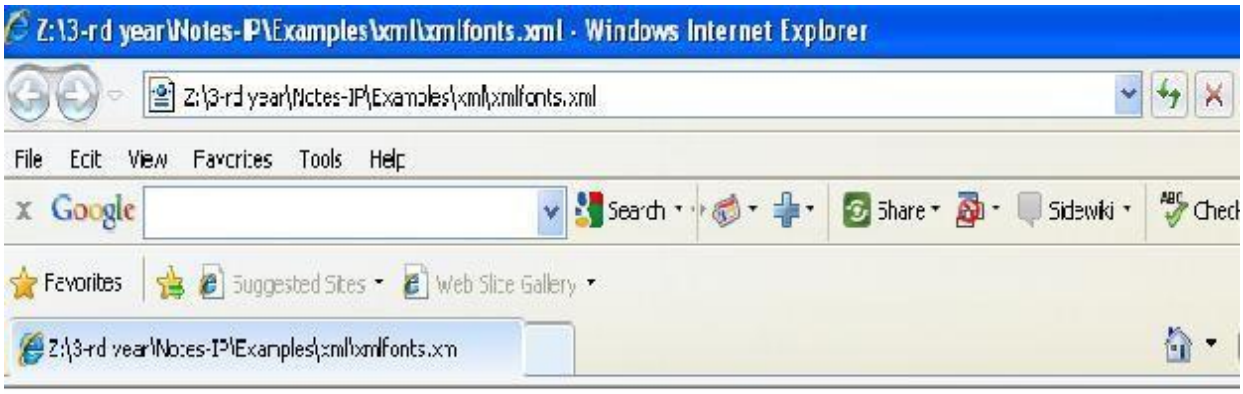
Family-name – name of the font family to use. font-family include : bookman, Lucida, and Times New Roman. All font family names within double quotes.
Generic-family – serif, sans-serif, cursive, fantasy, and monospace. because they are keywords, they cannot be included within quotation marks.

Example: xmlfonts.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="font.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

Filename : font.css

```
book { font-family: "comic sans"; } intro {font-family: "Monotype corsiva"; }
```



author { font-family: sans-serif; } publisher { font-family: "Helvetica"; } sample { font-family: serif; }

Font-size used to identify the size of the font
This setting based upon xml browsers default font-size settings

Syntax:

font-size: <absolute-size> | <relative-size> | <length> | <percentage> | inherit

Description:

<absolute-size> - (xx-small | x-small | small | medium | large | x-large). Medium-12pt ,small-10pt, scaling factor- 1.2 between each size

If the <relative-size> is specified (larger | smaller)

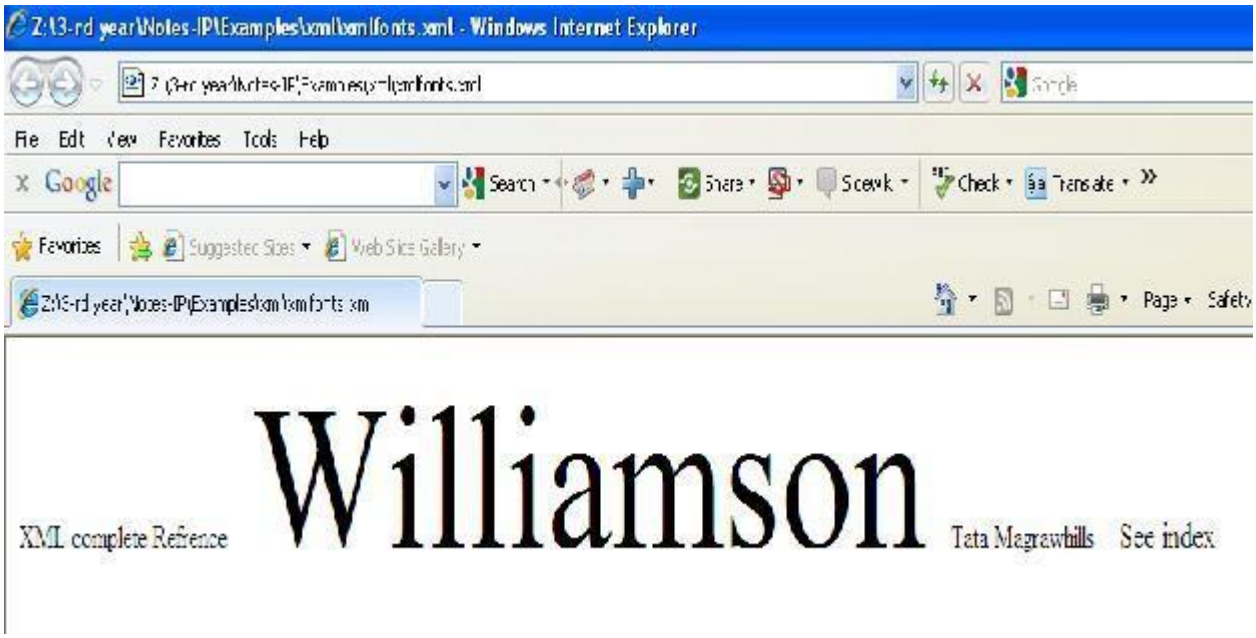
<length> - used to specify positive integer representing an absolute font size

<percentage> - to identify an absolute font size in relation to the parent
element's font size Example:-

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontsize.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

fontsize.css

book { font-size: "150"; } intro {font-size: small } author { font-size: larger; } publisher { font-size: "12pt"; } sample { font-size: medium; }



Setting font-stretch

- Font-stretch property is used to control the kerning of your font.
- It control the amount of space found between each individual character of the font.

Syntax:
font-stretch: normal | wider | narrower | ultra-condensed | xtra-condensed | condensed | semi-condensed | semi- expanded | expanded | extra-expanded | ultra-expanded | inherit

Description

ultra-condensed , xtra-condensed , condensed , semi-condensed , semi-expanded,expanded , extra-expanded, ultra-expanded – these values are organized from most condensed to least condensed. Each of them is a small change in horizontal spacing of text.

Wider – this value is relative to parent element and expands spacing. Without increasing the ultra-expanded level. Narrower – decreasing spacing one step without decreasing below ultra-condensedlevel

Example:

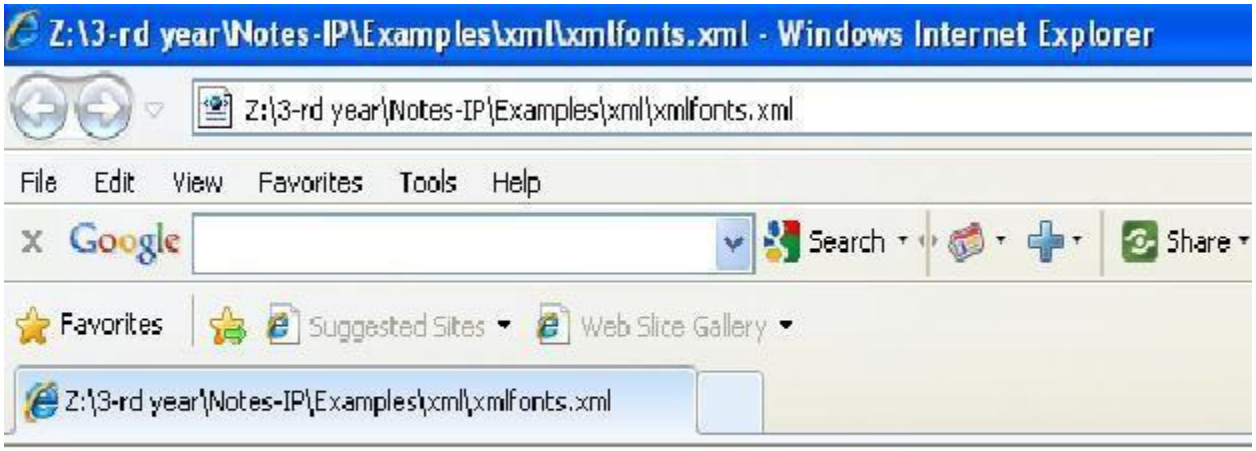
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontstretch.css"?>
<book>
<intro>XML complete Refrence</intro>
<author>Williamson</author>
```

<publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>

fontstretch.css

book { font-stretch: "ultra-expanded";} intro {font-stretch: narrower; }
author { font-stretch: "expanded"; } publisher { font-stretch: "normal"; }
sample { font-stretch: "wider"; }

Output:



XML complete Refrence Williamson Tata Magrawhills See index

Setting font-style

- Font-style property specifies the style of font being used
- You can specify normal opaque, or italic fonts

Syntax:
font-style: normal | italic | oblique | inherit

Description:-
normal ->no style change
Italic ->displays with an italic or cursive slant to each character oblique displays with an oblique, slanted, or inclined font
Example:

<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontstyle.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>

</book>

fontstyle.css
intro {font-style: "oblique"; } author { font-style: "italic"; } publisher { font-style: "normal"; }

Setting font-variant

- Font-variant property writes fonts using small caps for fonts supporting this style.
- Small caps are achieved by replacing all lower case letters with lowercase height capital letters.

Syntax:-
font-variant: normal | small-caps| inherit

Description:-

normal – the value does not change from the standard font character. small-caps – this value specifies to use small capital letters in place of standard lowercase letters within the specified text.

Example:-
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontvariant.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>

<sample>See index</sample>
</book>

fontvarian
t.css
intro {font-variant: "small-caps"; }

Setting font color

color property identifies the foreground color for the text content of an element.

Syntax
color:<colorname> |
<RGBcolor>

Description:-

colorname □□one of the valid color names.

RGBcolor□□name of the color returned in hexadecimal format representing Red- Green-Blue(RGB) value of the color

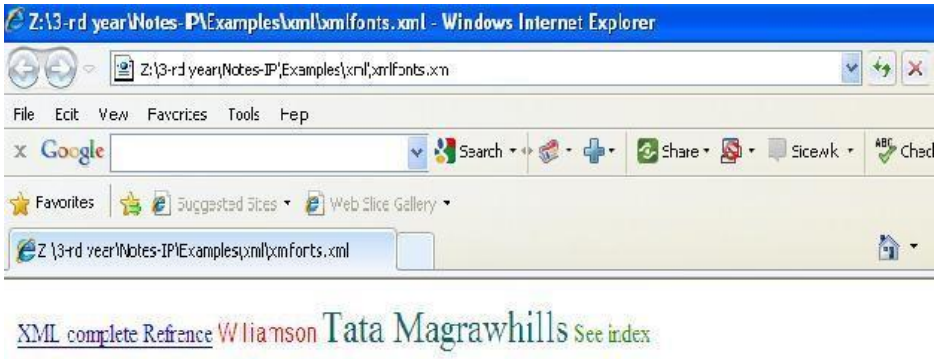
Example:-

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontcolor.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>
```

fontcolor.css

```
author {font-family: arial, helvetica, sans-serif; color:#FF0000 } intro { text-decoration: underline; color: Blue}
publisher { font-family: "times new roman"; font-size:"20pt"; color: rgb(10,100,100)}
sample { color: rgb(25,150,0)}
```

Output:-



Font-weight

Used to specify the weight or thickness of font to display a specified text Syntax:

font-weight: 100 | 200 | 300 | 400 | 500 | 700 | 800 | 900 | normal |bold | bolder | bolder | lighter | inherit

Description:-

Normal □□value is typically same as
value of 400 Bold□□same as value
700
Bolder □darker than the inherited one , increased by 100.
Lighter□□lighter than inherited one

Example:
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="fontweight.css"?> <book>
<intro>XML complete Refrence</intro> <author>Williamson</author> <publisher>Tata Magrawhills</publisher>
<sample>See index</sample>
</book>

Fontweight.css

book { font-weight: "400"; } intro { font-weight:"lighter";}sample { font-weight:"bold";}

XSL(EXTENSIBLE STYLESHEET LANGUAGE)

XSL is a stylesheet technology which address two important aspects

1.Transform the document from one type to
another (XSLT) 2.Styling the document
according to the formatting rules

XSLT stands for XSL
Transformation XSLT is the
most important part of XSL
Transform the document from one type to another

Syntax:- (root element)

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

The transformation language provides element that define rules for how one xml document is transformed into another xml element

XSL consists of three parts:

- XSLT - a language for transforming XML documents
- XPath - a language for navigating in XML documents
- XSL-FO - a language for formatting XML documents

What is XSLT?

- XSLT stands for XSL Transformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another XML document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT Uses XPath

XSLT uses XPath to find information in an XML document. XPath is used to navigate through elements and attributes in XML documents.

XSLT - Transformation

The root element that declares the document to be an XSL style sheet is <xsl:stylesheet> or <xsl:transform>. Note: <xsl:stylesheet> and <xsl:transform> are completely synonymous and either can be used!

The correct way to declare an XSL style sheet according to the W3C XSLT Recommendation is:

<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> (or)
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.

<xsl:template> Element

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched.
- The <xsl:template> element is used to build template

- The match attribute is used to associate a template with an xml element
- Match attribute can also be used to define a template for entire xml document
- Value of match attribute is an

xpath expression ie. match="/"
defines the whole document

Structure(Syntax)
<xsl:template match=" ">
…….
</xsl:tem
plate>
Example:
-
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"href="books.xsl"?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</bo
oks>
book
s.xsl
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>

<table border="10" bgcolor="green" cellpading="20" cellspacing="15" align="center">
<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>

<th>price</th>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

<xsl:value-of> Element

- This is most frequently used structure
- This is used to retrieve the value of the element and displaying the request
- The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation

Syntax:

<xsl:value-of select=“”>

Example:-

<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books>

books.xsl

<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">

```
<html>
<body>
<table border="10" bgcolor="green" cellpading="20" cellspacing="15" align="center">
<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<tr>
<td><xsl:value-of select ="books/book/title"/></td> <td><xsl:value-of select ="
books/book/author"/></td> <td><xsl:value-of select ="
books/book/publisher"/></td> <td><xsl:value-of select =" books/book/price"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
<xsl:for-each> Element
```

The <xsl:for-each> element allows you to do looping in XSLT.

The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set:

Syntax:

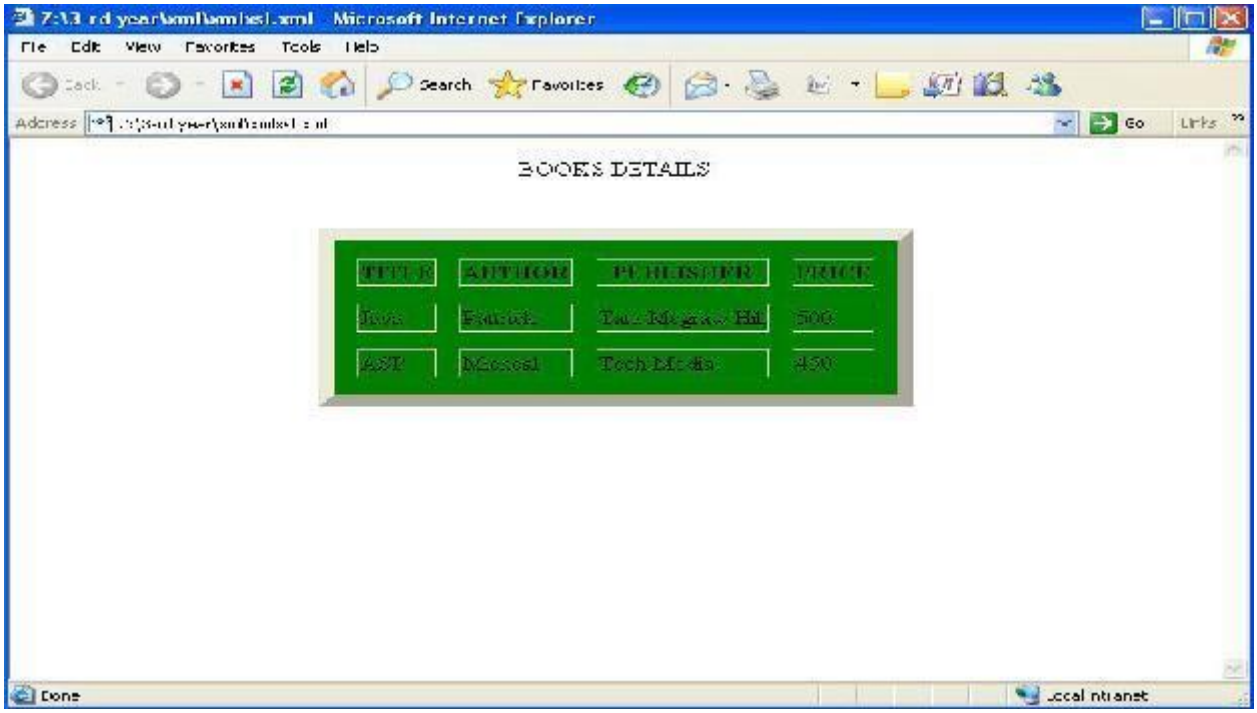
```
<xsl:for-each select=“”>
…….
</xsl:for-
each>
```

Example:-

```
<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
```

```
<book>
```

<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books>



books.xsl

<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
<table border="10" bgcolor="green" cellpading="20" cellspacing="15"
align="center">
<caption>books details</caption>
<tr>

<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<xsl:for-eachselect="books/book"> <tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td> <td><xsl:value-of select ="
price"/></td>
</tr> </xsl:for-each></table>
</body>
</html>
</xsl:template>
</xsl:stylesheet> Output:-

<xsl:if> Element

The <xsl:if> element is used to put a conditional test against the content of the XML file.

Syntax:
<xsl:if select=“ Condition”>
…..
</xsl:if> Example:-

<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books> books.xsl
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
<table border="10" bgcolor="green" cellpading="20" cellspacing="15" align="center">

<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<xsl:for-eachselect="books/book">
<xsl:if test="price > 451">
<tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td>
<td><xsl:value-of select =" price"/></td>
</tr>
</xsl:if >
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

<xsl:sort> Element

The <xsl:sort> element is used to sort the output.

Syntax:
<xsl:sort select=“ Condition”>
…..
</xsl:sort>

Example:-

<?xml version="1.0" ?>
<?xml-stylesheet type=”text/xsl”href=”books.xsl”?> <books>
<book>
<title>java</title>
<author>pattrick</author> <publisher>tata mcgraw hill</publisher> <price>500</price>
</book>
<book>

```
<title>asp</title>
<author>micheal</author> <publisher>tech media</publisher> <price>450</price>
</book>
</books>
```

books.xsl

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl"><xsl:template match="/">
<html>
<body>
<table border="10" bgcolor="green" cellpading="20" cellspacing="15" align="center">
<caption>books details</caption> <tr>
<th>title</th>
<th>author</th>
<th>publisher</th>
<th>price</th>
</tr>
<xsl:for-eachselect="books/book"> <xsl:sort select="title "/>
<tr>
<td><xsl:value-of select ="title"/></td> <td><xsl:value-of select =" author"/></td>
<td><xsl:value-of select =" publisher"/></td> <td><xsl:value-of select ="
price"/></td>
</tr> </xsl:sort >
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

XML SCHEMA

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

What is an XML Schema?

The purpose of an XML Schema is to define the legal building blocks of an XML document, just like a DTD.

An XML Schema:

- Defines elements that can appear in a document
- Defines attributes that can appear in a document
- Defines which elements are child elements
- Defines the order of child elements
- Defines the number of child elements
- Defines whether an element is empty or can include text
- Defines data types for elements and attributes
- Defines default and fixed values
  for

Elements and attributes advantages

- XML Schemas are extensible to future additions
- XML Schemas are richer and more powerful than DTDs
- XML Schemas are written in XML
- XML Schemas support data types
- XML Schemas support namespaces

Why use XML schema

- Schema support data types
- Easier to describe allowable document content
- Easier to validate the correctness of data
- Easier to work with data from database
- Easier to define data facets (restriction on data)
- Easier to define data pattern (data formats)
- Easier to convert data between different data types Well-Formed XML schema

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

Schema element is the root element of every xml schema

<?xml version="1.0"?>
<xs:schema>
…….
</xs:schema>

Schema declaration

<xs:schema
xmlns:xs=http://www.w3.org/2001/xmlschema targetNamespace=http://w3school.com
xmlns=http://www.w3school.com elementFormDefault="qualified">
……
……
</xs:schema>

Description:-

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

Indicates that the elements and data types used in the schema come from the "http://www.w3.org/2001/XMLSchema" namespace should be prefixed with xs:

targetNamespace=http://www.w3schools.com

Indicates that the elements defined by this schema (note, to, from, heading, body.) come from the "http://www.w3schools.com" namespace.

xmlns=http://www.w3schools.com

Indicates that the default namespace is "http://www.w3schools.com"

elementFormDefault="qualified"

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

XML SIMPLE ELEMENT

XML Schemas define the elements of your XML files.

What is a Simple Element?

A simple element is an XML element that contains 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.

syntax :

<xs:element
name="xxx"
type="yyy"/>
Description;-
xxx- the name of the element
yyy- the data type of the element.

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

1. xs:string
2. xs:decimal
3. xs:integer
4. xs:boolean
5. xs:date
6. xs:time

Example:-

HERE ARE SOME XML ELEMENTS:

<lastname>Refsnes</lastname>
<age>36</age>

<dob>1970-03-27</dob><per>97.5</per>

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="dob" type="xs:date"/> <xs:element name="per" type="xs:decimal"/>

Default and Fixed Values for Simple Elements

<xs:element name="color" type="xs:string" default="red"/>

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" fixed="red"/>

A fixed value is also automatically assigned to the element, and you cannot specify another value. In the following example the fixed value is "red"

Example:-

<?xml version="1.0"?>

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

note.xsd
<?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>

XSD ATTRIBUTES

All attributes are declared as simple types.

What is an Attribute?

Simple elements cannot have attributes. If an element has attributes, it is considered to be of a complex type.

But the attribute itself is always declared as a simple type.

syntax :

<xs:attribute name="xxx" type="yyy"/>

Description:-

xxx- the name of the attribute

yyy- specifies the data type of the attribute.

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

1. xs:string
2. xs:decimal
3. xs:integer
4. xs:boolean
5. xs:date
6. xs:time

Example:-

Here is an XML element with an attribute:

<lastname lang="EN">Smith</lastname>

And here is the corresponding attribute definition:

<xs:attribute name="lang" type="xs:string"/> Default and Fixed Values for Attributes
<xs:attribute name="lang" type="xs:string" default="EN"/>

A default value is automatically assigned to the attribute when no other value is specified. In the following example the default value is "EN":

<xs:attribute name="lang" type="xs:string" fixed="EN"/>

A fixed value is also automatically assigned to the attribute, and you cannot specify another value.In the following example the fixed value is "EN":

XSD Complex Elements

A complex element contains other elements and/or attributes.

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!

Example:

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>

Example Program:

Simple XML Element

<employee>

<firstname>John</firstname>

<lastname>

</employee>

XSD format

<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>

XML QUERY (XQUERY)

- Xquery is for XML like SQL for database
- Xquery is used to finding & Extracting elements and attributes from XML document
- XQuery is *the* language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all major databases
- XQuery is a W3C Recommendation

XQuery Basic Syntax Rules

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a $ followed by a name

Ex. $booklist XML Query (Xquery) – booklist.xml

<booklist>

<book>

<title>XML Complete Reference</title> <author>Williamson</author> <price>499</price>

</book>

<book>

<title>JavaScript Complete Reference</title> <author>Thomas Pawel</author> <price>400</price>

</book>

<book>

<title>Java Complete Reference</title> <author>Pattrick</author> <price>440</price>

</book>

<booklist>

For opening a particular file we use doc() function extract data from xml document Path expression used to navigate to elements in XML document

Syntax:

doc("booklist.xml")/booklist/book/title this will extract

<title>XML Complete Reference</title> <title>JavaScript Complete Reference</title> <title>Java Complete Reference</title>

Predicates

This is used to limit the extracted information from the document.

Syntax:
doc(“booklist.xml”)/booklist/book[price<
450]

Output:

<title>JavaScript Complete Reference</title>

<title>Java Complete Reference</title>

FLWOR expression

- for - (optional) binds a variable to each item returned by the in expression
- let - (optional)
- where - (optional) specifies a criteria
- order by - (optional) specifies the sort-order of the result
- return - specifies what to return in the result

Example:-

FLWOR – acronym for “For”, “Let”, “Where”, “orderedby”, “Return”

How to select the node using FLWOR function for $x in
doc("booklist.xml")/booklist/book

where $x/[price<450] order by $x/title return $x/title

- The for clause selects all book elements under the booklist element into a variable called

  $x

The where clause selects only book elements with a price element with a value less than 45

The order by clause defines the sort-order. Will be sort by the title element

The return clause specifies what should be returned
Here it returns the title elements

<title>Java Complete Reference</title>

<title>JavaScript Complete Reference</title> Present the result in a html list

Xquery

```
<ul>
{
for $x in doc("booklist.xml")/booklist/book/title order by $x
return<li>{$x}</li>
}
</ul>
```

The result of the above will be

```
<ul>
<li> <title>XML Complete Reference</title></li> <li><title>Javascript Complete Reference</title></li>
<li><title>Java Complete Reference</title></li>
</ul>
```

Now we want to eliminate the title element, and show only the data inside the title element:

```
<ul>
{
for $x in doc("booklist.xml")/booklist/book/title order by $x
return<li>{data($x)}</li>
}
</ul>
```

The result will be (an HTML list):

```
<ul>
<li> XML Complete Reference</li> <li>Javascript Complete Reference</li>
<li>Java Complete Reference</li>
</ul>
```

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

UNIT – IV – INTERNET PROGRAMMING SIT1302

UNIT IV

PHP

4.1 Introduction to PHP & Features

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

Example

```
<html>
<body>
<?php
echo "My first PHP script!"; ?>
</body>
</html>
```

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host with PHP Support

- If your server has activated support for PHP you do not need to do anything.
- Just create some .php files, place them in your web directory, and the server will automatically parse them for you.
- You do not need to compile anything or install any extra tools.
- Because PHP is free, most web hosts offer PHP support.
- Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

4.2 PHP Scripts Basic PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with <**?php** and ends with **?>**:

```
<?php
// PHP code goes
here ?>
```

Example

```
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>

</body>
</html>
```

- PHP statements end with a semicolon (;)

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code.

Example

```
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line comment /*
This is a multiple-lines comment block that spans over multiple lines
*/
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5; echo
$x; ?>
</body>
</html>
```

Example

```
<html>
<body>
<?php
ECHO "Hello World!<br>"; echo "Hello World!<br>"; EcHo "Hello World!<br>";
?>
</body>
```

```
</html>
```

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

Example

```
<html>
<body>
<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>"; echo "My boat is " . $coLOR . "<br>"; ?>
</body>
</html>
```

4.3 Data Types

- Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

Example

```
<html>
<body>
<?php
$x = "Hello world!";
$y = 'Hello world!'; echo $x;
echo "<br>"; echo $y;
?>
</body>
</html>
```

OUTPUT:

Hello world! Hello world!

String Functions

- Get The Length of a String
- The PHP strlen() function returns the length of a string.
- The example below returns the length of the string "Hello world!":

Example

```
<html>
<body>
<?php
echo strlen("Hello world!"); ?>
</body>
</html>
```

OUPUT:

12

Count The Number of Words in a String

The PHP str_word_count() function counts the number of words in a string:

Example

```
<html>
<body>
<?php
echo str_word_count("Hello world!");
?>
</body>
</html>
```

OUPUT:

2

Reverse a String

- The PHP strrev() function reverses a string:

Example

```
<html>
<body>
<?php
```

```
echo strrev("Hello world!");
?>
</body>
</html>
```

OUTPUT:

!dlrow olleH

Search For a Specific Text Within a String

- The PHP strpos() function searches for a specific text within a string.
- If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.
- The example below searches for the text "world" in the string "Hello world!":

Example

```
<html>
<body>
<?php
echo strpos("Hello world!", "world");
?>
</body>
</html>
```

OUPUT:

6

Replace Text Within a String

- The PHP str_replace() function replaces some characters with some other characters in a string.
- The example below replaces the text "world" with "Dolly":

Example

```
<html>
<body>
<?php
echo str_replace("world", "Dolly", "Hello world!");
?>
</body>
</html>
```

OUPUT:

Hello Dolly!

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16- based - prefixed with 0x) or octal (8-based - prefixed with 0)
- In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

Example

```
<html>
<body>
<?php
$x = 5985;
var_dump($x);
?>
</body>
</html>
```

OUTPUT:

int(5985)

PHP Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

Example

```
<html>
<body>
<?php
$x = 10.365;
var_dump($x);
?>
```

```
</body>
</html>
```

OUTPUT:

float(10.365)

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

- An array stores multiple values in one single variable:
- An array is a special variable, which can hold more than one value at a time.
- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The solution is to create an array!
- An array can hold many values under a single name, and you can access the values by referring to an index number.

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

```
</body>
</html>
```

OUTPUT:

I like Volvo, BMW and Toyota.

Create an Array in PHP

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

4.4 Variables

- Variables are "containers" for storing information.
- Creating (Declaring) PHP Variables
- In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

```
<html>
<body>
<?php
$txt = "Hello world!";
$x = 5;

$y = 10.5;


echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
```

Output:

Hello world!
5
10.5

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Rules for PHP variables:

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A- z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

Output Variables

- The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
<html>
<body>
<?php
$txt = "W3Schools.com"; echo "I love $txt!";
?>
</body>
</html>
```

Output:

I love W3Schools.com!

Example

```
<html>
<body>
<?php
$txt = "W3Schools.com"; echo "I love " . $txt . "!";
?>
</body>
</html>
```

Output:

I love W3Schools.com!

Example

```
<html>
<body>
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
</body>
</html>
```

Output:

9

PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

```
<html>
<body>
<?php
$x = 5; // global scope function myTest() {
// using x inside this function will generate an error echo "<p>Variable x inside function is:
$x</p>";
}
myTest();
echo "<p>Variable x outside function is:
$x</p>"; ?>
</body>
</html>
```

OUTPUT:

Variable x inside function is:

Variable x outside function is: 5

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

```
<html>
<body>
<?php
function myTest() {
$x = 5; // local scope
echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error echo "<p>Variable x outside function is:
$x</p>"; ?>
</body>
</html>
```

OUTPUT:

Variable x inside function is: 5 Variable x outside function is:

The global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function):

Example

```
<html>
<body>
<?php
$x = 5;
$y = 10;
function myTest() { global $x, $y; $y
= $x + $y;
}
myTest(); // run function
echo $y; // output the new value for variable
$y ?>
```

Output:

15

```
</body>
</html>
```

The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the static keyword when you first declare the variable:

Example

```
<html>
<body>
<?php
function myTest() { static $x = 0; echo $x;
$x++;
}
myTest(); echo "<br>"; myTest(); echo "<br>"; myTest();
?>
</body>
</html>
```

Output:

0

1

2

echo and print Statements

- In PHP there are two basic ways to get output: echo and print.
- In this tutorial we use echo (and print) in almost every example. So, this chapter contains a little more info about those two output statements.
- echo and print are more or less the same. They are both used to output data to the screen.
- The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

echo Statement

The echo statement can be used with or without parentheses: echo or echo().

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<html>
<body>
<?php
echo "<h2>PHP is Fun!</h2>"; echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
```

```
echo "This ", "string ", "was ", "made ", "with multiple parameters."; ?>
</body>
</html>
```

OUTPUT:

PHP is Fun!
Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<html>
<body>
<?php
$txt1 = "Learn PHP"; $txt2
= "W3Schools.com"; $x = 5;
$y = 4;
echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
```

```
echo $x + $y;
?>
</body>
</html>
```

OUTPUT:

Learn PHP

Study PHP at W3Schools.com 9

The PHP print Statement

- The print statement can be used with or without parentheses: print or print().

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<html>
<body>
<?php
print "<h2>PHP is Fun!</h2>"; print "Hello world!<br>";
print "I'm about to learn PHP!"; ?>
</body>
</html>
```

OUTPUT:

PHP is Fun!

Hello world!

I'm about to learn PHP!

Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<html>
<body>
<?php
```

```
$txt1 = "Learn PHP";
$txt2= "W3Schools.com";
$x = 5;
$y = 4;
print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>"; print $x +
$y; ?>
</body>
</html>
```

OUTPUT:

Learn PHP

Study PHP at W3Schools.com 9

PHP Object

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<html>
<body>
<?php class Car {
                                    function Car() { $this-
                                      >model = "VW";
}
}
// create an object
$herbie = new Car();
// show object properties echo
$herbie->model; ?>
</body>
</html>
```

OUPUT:

VW

PHP NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- If a variable is created without a value, it is automatically assigned a value of NULL.
- Variables can also be emptied by setting the value to NULL:

Example

```
<html>
<body>
<?php
$x = "Hello world!";
$x = null; var_dump($x);
?>
</body>
</html>
```

OUTPUT:

NULL

Constants

- Constants are like variables except that once they are defined they cannot be changed or undefined.
- **PHP Constants**
- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

Create a PHP Constant

- To create a constant, use the define() function.

Syntax

define(*name*, *value*, *case-insensitive*)

Parameters:

- ***name*:** Specifies the name of the constant
- ***value*:** Specifies the value of the constant

- ***case-insensitive*:** Specifies whether the constant name should be case- insensitive. Default is false

The example below creates a constant with a **case-sensitive** name:

Example

```
<html>
<body>
<?php
// case-sensitive constant name
define("GREETING", "Welcome to W3Schools.com!"); echo GREETING;
?>
</body>
</html>
```

OUTPUT:

Welcome to W3Schools.com!

The example below creates a constant with a **case-insensitive** name:

Example

```
<html>
<body>
<?php
// case-insensitive constant name
define("GREETING", "Welcome to W3Schools.com!", true); echo greeting;
?>
</body>
</html>
```

OUPUT:

Welcome to W3Schools.com!

Constants are Global

- Constants are automatically global and can be used across the entire script.
- The example below uses a constant inside a function, even if it is defined outside the function:

Example

```
<html>
<body>
<?php
define("GREETING", "Welcome to W3Schools.com!"); function myTest() {
```

```
echo GREETING;
}
myTest();
?>
</body>
</html>
```

OUPUT:

Welcome to W3Schools.com!

4.5 PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic Operators

- The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y

/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

PHP Assignment Operators

- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description
x = y		The left operand gets set to the value of the expression on the right
x = x + y		Addition
x = x - y		Subtraction
x = x *		Multiplication

x += y	y	
x -= y		
x *= y	x = x / y	Division
x /= y	x = x % y	Modulus
x %= y		

PHP Comparison Operators

- The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y

===	Identical	$x===$y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x!=$y	Returns true if $x is not equal to $y
<>	Not equal	$x<>$y	Returns true if $x is not equal to $y
!==	Not identical	$x!==$y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x>$y	Returns true if $x is greater than $y
<	Less than	$x<$y	Returns true if $x is less than $y

>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Increment / Decrement Operators

- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

- The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
And	And	$x and $y	True if both $x and $y are true
Or	Or	$x or $y	True if either $x or $y is true
Xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

- PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2

.=	Concatenation assignment	$txt1 .= $txt2 Appends $txt2 to $txt1

PHP Array Operators

- The PHP array operators are used to compare arrays.

Operator		Example	
+	Unio	$x +	Union of $x and
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

4.6 PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to dothis.

In PHP we have the following conditional statements:

- if statement - executes some code if one condition is true
- if...else statement - executes some code if a condition is true and another code if that condition is false
- if...elseif ....else statement - executes different codes for more than two conditions
- switch statement - selects one of many blocks of code to be executed

The if Statement

- The if statement executes some code if one condition is true.

Syntax

if (*condition*) {
code to be executed if condition is true;
}

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<html>
<body>
<?php
$t = date("H");
if ($t < "20") {
echo "Have a good day!";
}
?>
</body>
</html>
```

OUTPUT:

Have a good day!

The if...else Statement

The if else statement executes some code if a condition is true and another code if that condition is false.

Syntax

if (*condition*) {
code to be executed if condition is true;
} else {
code to be executed if condition is false;

}

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<html>
<body>
<?php
```

```
$t = date("H");
if ($t < "20") {
echo "Have a good day!";
} else {
echo "Have a good night!";
}
?>
</body>
</html>
```

OUTPUT:

Have a good day!

The if.....else Statement

The if.....else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
code to be executed if this condition is true;
} elseif (condition) {
code to be executed if this condition is true;
} else {
code to be executed if all conditions are false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<html>
<body>
<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>"; if ($t < "10") {
echo "Have a good morning!";
} elseif ($t < "20") {
echo "Have a good day!";
} else {
echo "Have a good night!";
}
?>
</body>
</html>
```

OUTPUT:

The hour (of the server) is 01, and will give the following message: Have a good morning!

The switch Statement

- The switch statement is used to perform different actions based on different conditions.

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) { case label1:
code to be executed if n=label1;
    break; case label2:
code to be executed if n=label2;
    break; case label3:
code to be executed if n=label3;
break;
...
default:
code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the

code from running into the next case automatically. The default statement is used if no match is found.

Example

```
<html>
<body>
<?php
$favcolor = "red"; switch ($favcolor) {
case "red":
echo "Your favorite color is red!"; break;
case "blue":
echo "Your favorite color is blue!"; break;
case "green":
echo "Your favorite color is green!"; break;
default:
echo "Your favorite color is neither red, blue, nor green!";
}
?>
</body>
</html>
```

OUTPUT:

Your favorite color is red!

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- while- loops through a block of code as long as the specified condition istrue
- do...while - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- for- loops through a block of code a specified number of times
- foreach- loops through a block of code for each element in an array

The PHP while Loop

- The while loop executes a block of code as long as the specified conditionis true.

Syntax

```
while (condition is true) {
code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

Example

```
<html>
<body>
<?php
$x = 1;
while($x <= 5) {
echo "The number is: $x
<br>"; $x++;
}
?>
</body>
</html>
```

OUTPUT:

The number is: 1 The number is: 2 The number is: 3 The number is: 4 The number is: 5

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

do {

code to be executed; } while (*condition is true*);

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<html>
<body>
<?php
$x = 1; do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 5); ?>
</body>
</html>
```

OUTPUT:

The number is: 1 The number is: 2 The number is: 3 The number is: 4 The number is: 5

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<html>
<body>
<?php
$x = 6;
do {
echo "The number is: $x <br>";
$x++;
} while ($x <= 5); ?>
</body>
</html>
```

OUTPUT:

The number is: 6

for Loops

- PHP for loops execute a block of code a specified number of times.

The PHP for Loop

- The for loop is used when you know in advance how many times thescript should run.

Syntax

for (*init counter; test counter; increment counter*) {
code to be executed;
}

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop countervalue The example below displays the numbers

from 0 to 10: **Example**

```
<html>
<body>
<?php
for ($x = 0; $x <= 10; $x++) { echo "The number is: $x <br>";
}
?>
</body>
</html>
```

OUTPUT:

The number is: 0 The number is: 1 The number is: 2 The number is: 3 The number is: 4 The number is: 5 The number is: 6 The number is: 7 The number is: 8 The number is: 9 The number is: 10

The PHP foreach Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value)

{
code to be executed;
}
```

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.
- The following example demonstrates a loop that will output the values of the given array ($colors):

Example

```
<html>
<body>
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value)
{ echo "$value <br>";
}
?>
</body>
</html>
```

OUTPUT:

red green blue yellow

4.7 Working with Arrays PHP Indexed Arrays

- There are two ways to create indexed arrays:
- The index can be assigned automatically (index always starts at 0), like this:
- $cars = array("Volvo", "BMW", "Toyota");
- or the index can be assigned manually:
- $cars[0] = "Volvo";
- $cars[1] = "BMW";
- $cars[2] = "Toyota";
- The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
</body>
</html>
```

OUTPUT:

I like Volvo, BMW and Toyota.

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota"); echo count($cars);
?>
</body>
</html>
```

OUTPUT:

3

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) { echo $cars[$x];
echo "<br>";
}
?>
</body>
</html>
```

OUTPUT:

Volvo BMW
Toyota

PHP Associative Arrays

- Associative arrays are arrays that use named keys that you assign to them.
- There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

- The named keys can then be used in a script:

Example

```
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); echo "Peter is " . $age['Peter'] . " years old.";
?>
</body>
</html>
```

OUTPUT:

Peter is 35 years old.

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

<?php

$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); foreach($age as $x => $x_value) {

echo "Key=" . $x . ", Value=" . $x_value; echo "
";

}

?>

Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example

```
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota"); sort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) { echo $cars[$x];
echo "<br>";
}
?>
</body>
</html>
```

OUTPUT:

BMW
Toyota Volvo

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22,
11) ; sort($numbers);
?>
```

OUTPUT:

2
4
6
11
22

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); asort($age);
?>
```

OUTPUT:

Key=Peter, Value=35 Key=Ben, Value=37 Key=Joe, Value=43

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); ksort($age);
?>
```

OUTPUT:

Key=Ben, Value=37 Key=Joe, Value=43 Key=Peter, Value=35

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); arsort($age);
```

?>

OUTPUT:

Key=Joe, Value=43 Key=Ben, Value=37 Key=Peter, Value=35

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); krsort($age);
?>
```

OUTPUT:

Key=Peter, Value=35 Key=Joe, Value=43 Key=Ben, Value=37

- An array stores multiple values in one single variable.
- In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

Example

```
<html>
<body>
<?php
$cars = array("Volvo","BMW","Toyota"); var_dump($cars);
?>
</body>
</html>
```

OUTPUT:

array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

4.8 Functions

- The real power of PHP comes from its functions; it has more than 1000 built-in functions.

PHP User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user-defined function declaration starts with the word function:

Syntax

function *functionName*() {

code to be executed;
}

- Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<html>
<body>
<?php
function writeMsg() echo "Hello world!";
}
writeMsg();
?>
</body>
</html>
```

OUTPUT:

Hello world!

PHP Function Arguments

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<html>
<body>
<?php
function familyName($fname) { echo "$fname Refsnes.<br>";
}
familyName("Jani"); familyName("Hege"); familyName("Stale"); familyName("Kai Jim");
familyName("Borge");
?>
</body>
</html>
```

OUTPUT:

Jani Refsnes.
HegeRefsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

The following example has a function with two arguments ($fname and $year):

Example

```
<html>
<body>
<?php
function familyName($fname, $year) {
echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege","1975"); familyName("Stale","1978"); familyName("Kai Jim","1983"); ?>
</body>
</html>
```

OUTPUT:

Hege Refsnes. Born in 1975 Stale Refsnes. Born in 1978 Kai Jim Refsnes. Born in 1983

PHP Default Argument Value

- The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<html>
<body>
<?php
function setHeight($minheight = 50) { echo "The height is : $minheight <br>";
}
setHeight(350); setHeight(); setHeight(135); setHeight(80);
?>
</body>
</html>
```

OUTPUT:

The height is : 350 The height is : 50 The height is : 135 The height is : 80

PHP Functions - Returning values

- To let a function return a value, use the return statement:

Example

```
<html>
<body>
<?php
function sum($x, $y) {
$z = $x + $y; return $z;
}
echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>"; echo "2 + 4 = " . sum(2,4);
?>
</body>
</html>
```

OUTPUT: 5 + 10 = 15

7 + 13 = 20
2 + 4 = 6

4.9 Working with Files

- File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

- When you are manipulating files you must be very careful.
- You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

- The readfile() function reads a file and writes it to the output buffer.
- Assume we have a text file called **"webdictionary.txt",** stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

- The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```
<html>
<body>
<?php
echo readfile("webdictionary.txt");
?>
</body>
</html>
```

OUTPUT:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language236

- The readfile() function is useful if all you want to do is open up a file and read its contents.

File Open/Read/Close PHP Open File - fopen()

- A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, **"webdictionary.txt",** during the lessons: AJAX = Asynchronous

JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!"); echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

OUTPUT:

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists

r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already Exists

PHP Read File - fread()

- The fread() function reads from an open file.
- The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.
- The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Close File - fclose()

- The fclose() function is used to close an open file.
- It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!
- The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r"); // some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

- The fgets() function is used to read a single line from a file.
- The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!"); echo fgets($myfile);
fclose($myfile);
?>
```

OUTPUT:

AJAX = Asynchronous JavaScript and XML

After a call to the fgets() function, the file pointer has moved to the next line.

PHP Check End-Of-File - feof()

- The feof() function checks if the "end-of-file" (EOF) has been reached.
- The feof() function is useful for looping through data of unknown length.
- The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file while(!feof($myfile)) {
echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

OUTPUT:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

PHP Read Single Character - fgetc()

- The fgetc() function is used to read a single character from a file.
- The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) { echo fgetc($myfile);
}
fclose($myfile);
?>
```

OUTPUT:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language

- After a call to the fgetc() function, the file pointer moves to the next character.

File Create/Write

PHP Create File - fopen()

- The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.
- If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).
- The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

- The fwrite() function is used to write to a file.
- The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.
- The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!"); $txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n"; fwrite($myfile, $txt); fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

John Doe Jane Doe

PHP Overwriting

- Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.
- In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!"); $txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n"; fwrite($myfile, $txt); fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

OUTPUT:

Mickey Mouse Minnie Mouse

PHP ftp_fput() Function Example

- Open local file, and upload it to a file on the FTP server:

```
<?php
// connect and login to FTP server
$ftp_server = "ftp.example.com";
$ftp_conn = ftp_connect($ftp_server) or die("Could not connect to
$ftp_server"); $login = ftp_login($ftp_conn, $ftp_username, $ftp_userpass);

// open file for reading
$file = "test.txt";
$fp = fopen($file,"r");

// upload file
if (ftp_fput($ftp_conn, "somefile.txt", $fp, FTP_ASCII))
{
echo "Successfully uploaded $file.";
}
else
{
echo "Error uploading $file.";
}

// close this connection and file handler ftp_close($ftp_conn);
fclose($fp);
?>
```

Definition and Usage

- The ftp_fput() function uploads from an open file and saves it to a file on the FTP server.

Syntax
ftp_fput(*ftp_connection,remote_file,open_file,mode,startpos*);

Parameter	Description
ftp_connection	Required. Specifies the FTP connection to use
remote_file	Required. Specifies the file path to upload to
open_file	Required. Specifies an open local file. Reading stops at end of file
mode	Required. Specifies the transfer mode. Possible values: FTP_ASCII or FTP_BINARY
startpos	Optional. Specifies the position in the remote file to start uploading to

4.10 Working with Databases

PHP MySQL Database

- With PHP, you can connect to and manipulate databases.
- MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

- MySQL is developed, distributed, and supported by Oracle Corporation

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

PHP + MySQL Database System

- PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like". Both MySQLi and PDO have their

advantages:

- PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.
- So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.
- Both are object-oriented, but MySQLi also offers a procedural API.
- Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Examples in Both MySQLi and PDO Syntax

In this, and in the following chapters we demonstrate three ways of working with PHP and MySQL:

- MySQLi (object-oriented)
- MySQLi (procedural)
- PDO

MySQLi Installation

- For Linux and Windows: The MySQLi extension is automatically installed in most cases, when php5 mysql package is installed.
- For installation details, go to: http://php.net/manual/en/mysqli.installation.php

- PDO Installation
- For installation details, go to: http://php.net/manual/en/pdo.installation.php

Open a Connection to MySQL

- Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username  = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully"; ?>
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username  = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection if (!$conn) {
die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully"; ?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username  = "username";
$password = "password";

try {
$conn = new PDO("mysql:host=$servername;dbname=myDB",
$username, $password);
// set the PDO error mode to exception $conn-
>setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); echo
"Connected successfully";
}
catch(PDOException $e)
{
echo "Connection failed: " . $e->getMessage();
}
?>
```

Notice that in the PDO example above we have also specified a database (myDB). PDO require a valid database to connect to. If no database is specified, an exception is thrown.

Tip: A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented)

```
$conn->close();
```

Example (MySQLi Procedural)

```
mysqli_close($conn);
```

Example (PDO)

```
$conn = null;
```

PHP Create a MySQL Database

- A database consists of one or more tables.
- You will need special CREATE privileges to create or to delete a MySQL database.
- **Create a MySQL Database Using MySQLi**
- The CREATE DATABASE statement is used to create a database in MySQL.
- The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username  = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username,
$password); // Check connection if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
    echo "Database created successfully"; } else {
echo "Error creating database: " . $conn->error;
}
$conn-
>close(); ?>
```

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)

PHP Create MySQL Table

- A database table has its own unique name and consists of columns and rows.
- Create a MySQL Table Using MySQLi
- The CREATE TABLE statement is used to create a table in MySQL.

- We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY, firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL, email VARCHAR(50),
reg_date TIMESTAMP
)
```

Notes on the table above:

- The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

  After the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password,
$dbname); // Check connection if ($conn->connect_error) {
die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY, firstname VARCHAR(30) NOT
```

```
NULL,
lastname VARCHAR(30) NOT NULL, email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) { echo "Table MyGuests created
successfully"; } else {
echo "Error creating table: " . $conn->error;
}

$conn-
>close(); ?>
```

PHP Insert Data Into MySQL

Insert Data Into MySQL Using MySQLi

- After a database and a table have been created, we can start adding data in them.

  Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)
```

Note: If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password,
$dbname); // Check connection if ($conn->connect_error) {
```

```
die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email) VALUES ('John', 'Doe', 'john@example.com')";

 if ($conn->query($sql) === TRUE) { echo "New record created
successfully"; } else {
echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn-
>close(); ?>
```

Example:

Student Application using PHP and Mysql aa.html:

```
<!DOCTYPE HTML>
```

```
<html>
<head>
<title>Student Table</title> </head>
<body>
<div id="dept"> <h3 align=center>Student table entry</h3> <form method="POST" action="connect.php">
SName <br><input type="text" name="sname"><br> Reg.No <br><input type="text" name="regno"><br> Mark1<br><input type="text" name="m1"><br> Mark2<br><input type="text" name="m2"><br> <input type="submit" value="ok">
</form>
</div>
</body>
</html>
```

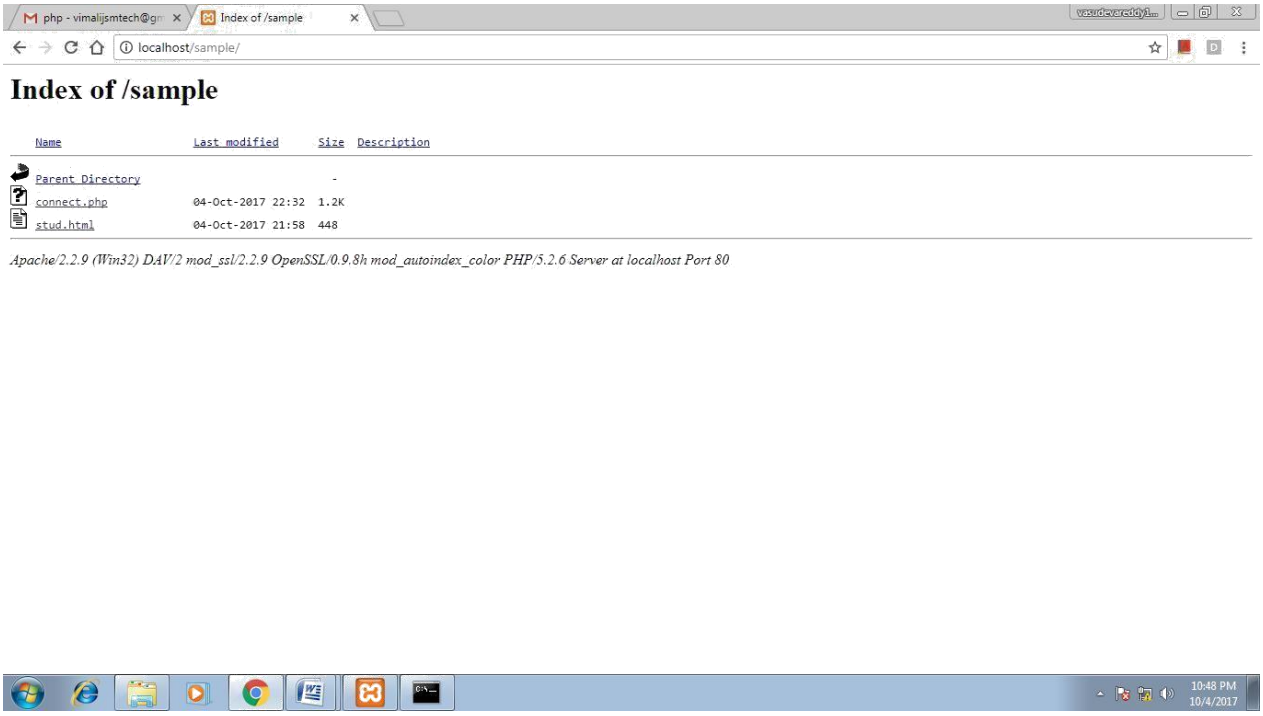
Conn.php:

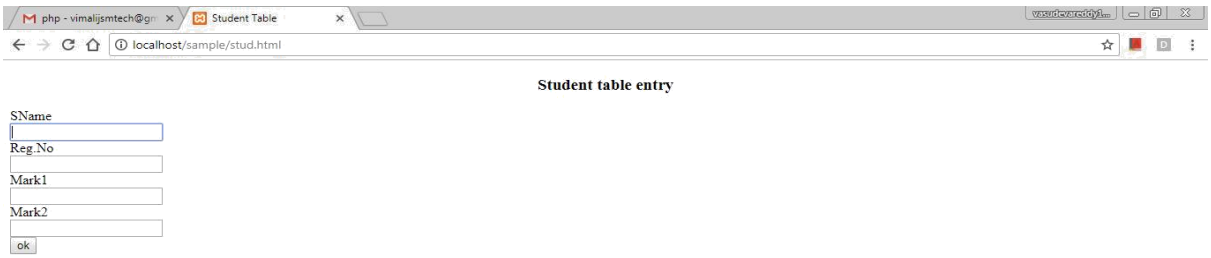
```
<?php
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "sample";
// Create connection
$conn = new mysqli($servername, $username, $password,$dbname);
// Check connection
if ($conn->connect_error)
{
die("Connection failed: " . $conn->connect_error);
}
```

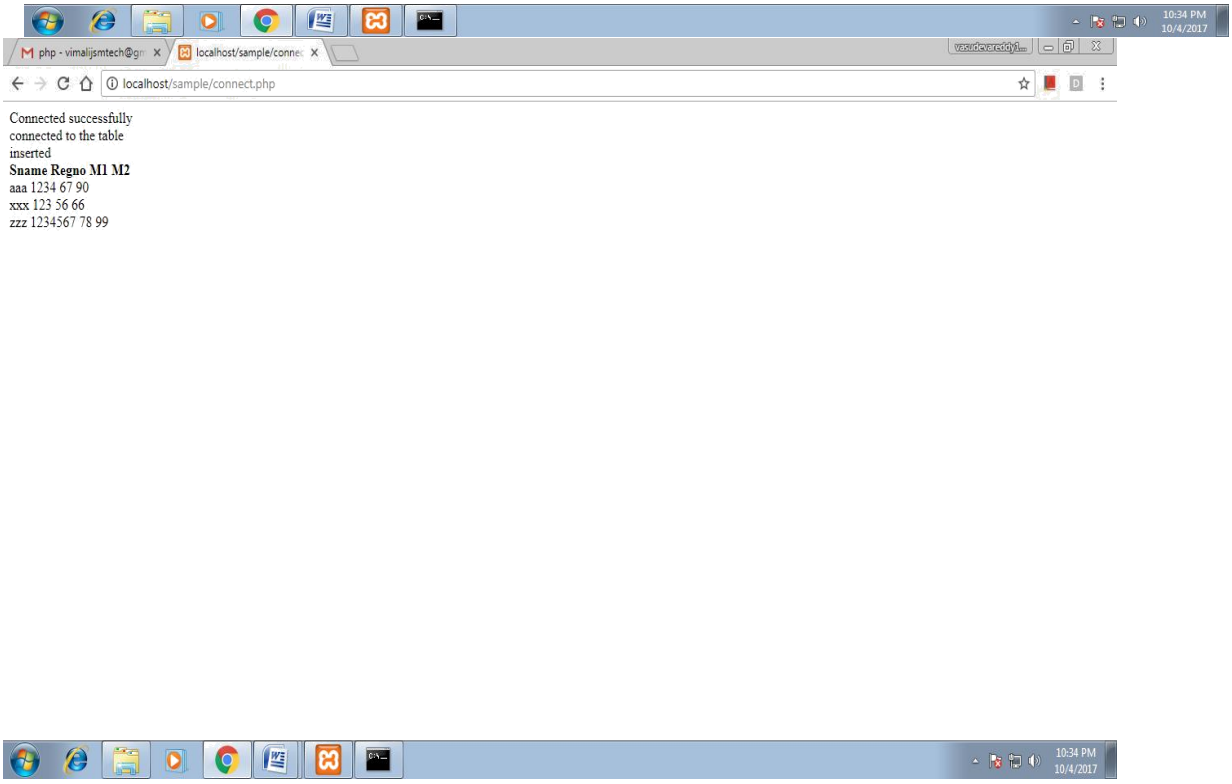
```
echo "Connected successfully"; //connect table
$sql="desc student"; if($conn-
>query($sql)==TRUE)
{
echo "<br>";
echo "connected to the table";
}
else
{
echo "error";
}
//Inserting the contents echo "<br>";
//insertion from html
$sname=$_POST['sname'];
$regno=$_POST['regno'];
$m1=$_POST['m1'];
$m2=$_POST['m2'];
$sql11="insert into student values('$sname',$regno,$m1,$m2)"; if($conn->query($sql11)==TRUE)
{
echo "inserted";
}
else
{echo "error";} echo "<br>";
$sql1="select * from student";
$result = $conn->query($sql1); if ($result->num_rows > 0) {
```

```
// output data of each row
echo "<b>Sname Regno M1 M2</b><br>"; while($row = $result->fetch_assoc()) {
        echo $row["sname"]." ". $row["regno"]." ".$row["m1"]." ".$row ["m2"]."<br>";
}
} else {
echo "empty table";
}
$conn-
>close(); ?>
```

OUTPUT:




php - vimalijsmtech@gm
Student Table
localhost/sample/stud.html
Student table entry
SName
Reg.No
Mark1
Mark2
ok


php - vimalijsmtech@gm
localhost/sample/connec
localhost/sample/connect.php
Connected successfully
connected to the table
inserted
Sname Regno M1 M2
aaa 1234 67 90
xxx 123 56 66
zzz 1234567 78 99

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF INFORMATION TECHNOLOGY

UNIT – V – INTERNET PROGRAMMING SIT1302

UNIT V

5.1 Introducing ASP.NET

ASP.NET is a web development platform, which provides a programming model, a comprehensive software infrastructure and various services required to build up robust web applications for PC, as well as mobile devices.

ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.

ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.NET application codes can be written in any of the following languages:

- C#
- Visual Basic.Net
- Jscript
- J#

ASP.NET is used to produce interactive, data-driven web applications over the internet. It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.

ASP.NET Web Forms Model

ASP.NET web forms extend the event-driven model of interaction to the web applications. The browser submits a web form to the web server and the server returns a full markup page or HTML page in response.

All client side user activities are forwarded to the server for stateful processing. The server processes the output of the client actions and triggers the reactions.

Now, HTTP is a stateless protocol. ASP.NET framework helps in storing the information regarding the state of the application, which consists of:

- Page state

- Session state

The page state is the state of the client, i.e., the content of various input fields in the web form. The session state is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart.

User adds items to a shopping cart. Items are selected from a page, say the items page, and the total collected items and price are shown on a different page, say the cart page. Only HTTP cannot keep track of all the information coming from various pages. ASP.NET session state and server side infrastructure keeps track of the information collected globally over a session.

The ASP.NET runtime carries the page state to and from the server across page requests while generating ASP.NET runtime codes, and incorporates the state of the server side components in hidden fields.

This way, the server becomes aware of the overall application state and operates in a two-tiered connected way.

The ASP.NET Component Model

The ASP.NET component model provides various building blocks of ASP.NET pages. Basically it is an object model, which describes:

- Server side counterparts of almost all HTML elements or tags, such as <form> and <input>.
- Server controls, which help in developing complex user-interface. For example, the Calendar control or the Gridview control.

ASP.NET is a technology, which works on the .Net framework that contains all web-related functionalities. The .Net framework is made of an object-oriented hierarchy. An ASP.NET web application is made of pages. When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.

The ASP.NET runtime transforms the .aspx page into an instance of a class, which inherits from the base class page of the .Net framework. Therefore, each ASP.NET page is an object and all its components i.e., the server-side controls are also objects.

Components of .Net Framework 3.5

Let us go through at the various components of the .Net framework 3.5. The following table describes the components of the .Net framework 3.5 and the job they perform:

Components and their Description

(1) Common Language Runtime or CLR

It performs memory management, exception handling, debugging, security checking, thread execution, code execution, code safety, verification, and compilation. The code that is directly managed by the CLR is called the managed code. When the managed code is compiled, the compiler converts the source code into a CPU independent intermediate language (IL) code. A Just In Time(JIT) compiler compiles the IL code into native code, which is CPU specific.

(2) .Net Framework Class Library

It contains a huge library of reusable types. classes, interfaces, structures, and enumerated values, which are collectively called types.

(3) Common Language Specification

It contains the specifications for the .Net supported languages and implementation of language integration.

(4) Common Type System

It provides guidelines for declaring, using, and managing types at runtime, and cross-language communication.

(5) Metadata and Assemblies

Metadata is the binary information describing the program, which is either stored in a portable executable file (PE) or in the memory. Assembly is a logical unit consisting of the assembly manifest, type metadata, IL code, and a set of resources like image files.

(6) Windows Forms

Windows Forms contain the graphical representation of any window displayed in the application

(7) ASP.NET and ASP.NET AJAX

ASP.NET is the web development model and AJAX is an extension of ASP.NET for developing and implementing AJAX functionality. ASP.NET AJAX contains the components that allow the developer to update data on a website without a complete reload of the page.

(8) ADO.NET

It is the technology used for working with data and databases. It provides access to data sources like SQL server, OLE DB, XML etc. The ADO.NET allows connection to data sources for retrieving, manipulating, and updating data.

(9) Windows Workflow Foundation (WF)

It helps in building workflow-based applications in Windows. It contains activities, workflow runtime, workflow designer, and a rules engine.

(10) Windows Presentation Foundation

It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations, and more.

(11) Windows Communication Foundation (WCF)

It is the technology used for building and executing connected systems.

(12) Windows CardSpace

It provides safety for accessing resources and sharing personal information on the internet.

(13) LINQ

It imparts data querying capabilities to .Net languages using a syntax which is similar to the tradition query language SQL.

5.2 ASP.NET namespaces

1) The System.Web namespace

System.web namespace holds some basic ingredients which includes classes and built-in Objects like

a. Server
b. Application

c. Request
d. Response

And Classes used for managing

a. Cookies
b. Configuring page caching
c. Tracing Implementation
d. Retrieving Information of Web Server and client browser

We uses built-in object frequently .This namespace are very important for ASP.net application, it is also required for User Interface , Web forms and Web services.

2) The System.Web.services namespace

The System.Web.services namespace is a initial point for creating the web services. It contains web service classes , which allow to create XML web services using Asp.Net . XML web services provides feature to exchange messages in loosely coupled environment. using protocol like SOAP
, HTTP , XML .
Some classes are as follows

a. WebMethodAttribute
b. Webservice
c. WebServiceAttribute
d. WebServiceBindingAttribute

3) The System.web.UI.WebControls namespace

The System.web.UI.WebControls namespace contains classes that provides to create web server controls on web pages. These controls run on the server so we can programmaticaly control elements . It also include form controls like text boxes and buttons. special purpose controls such as calender , and Data Grid . Web controls are more abstract than HTML controls . They also provide a shophisticated formatting properties and events . The System.web.UI.WebControls namespace contains web control class which is the base class for all web controls

Some classes are as follows

a. Calendar

b. Check Box
c. Button
d. Base Data Bound Control
e. Data Control Field etc

4) The System.web.UI namespace

The System.web.UI namespace includes classes that allows to create server controls with data-binding functionality , which has ability to save view state of given control and pages (.aspx pages)

. Many of these types allows to support for controls System.web.UI .Html controls. It is a base class for all HTML ,Web, and User Controls. Every aspx pages comes under it. Control classes provides common set of functionslity

There are following controls available

- HTML server controls
- Web server controls
- User controls

Some classes are as follows

a. Attribute Collection
b. BaseParser
c. BaseTemplateParser
d. AsyncPostBackTrigger etc

5) The System.web.sessionstate namespace

Session state management comes under The System.web.sessionstate namespace . That enable storage of data specific to a single client with in a web application on the server. Session state is ideal for sensitive data such as mailing address , credit card number , password , important numbers etc. Sesssion state can be used with clients that do not support cookies.

Some classes are as follows

a. HttpSessionState
b. HttpSessionStateContainer

c. SessionIDManager

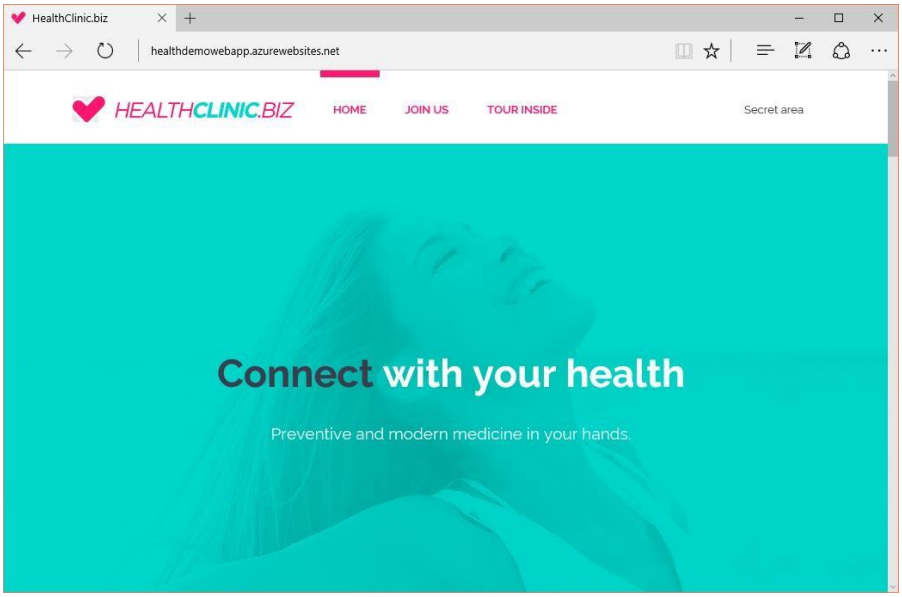
d. SessionStateModule etc

5.3 Creating and deploying ASP.NET applications

Azure App Service is the only cloud service that integrates everything you need to quickly and easily build web and mobile apps for any platform and any device. Built for developers, App Service is a fully managed platform that has powerful capabilities such as built-in DevOps, continuous integration with Visual Studio Team Services and GitHub, staging and production support, and automatic patching.

This walkthrough shows how to deploy an ASP.NET web application to a web app in Azure App Service by using Visual Studio 2015 or Visual Studio 2013. The walkthrough assumes that you are an ASP.NET developer who has no prior experience with Azure. After you complete the tutorial, you'll have a web app that's deployed to Azure from Visual Studio. Samples and tutorials in this article are based on the Connect(); //2015 event demos.

The following illustration shows the completed application:



This article will help you learn the following:

- How to prepare your computer for Azure development by installing the Azure SDK for .NET.
- How to set up Visual Studio to create a new web app in Azure App Service and an ASP.NET MVC 5 web project.
- How to compose the Demo project that is presented in Microsoft Connect(); //2015.
- How to deploy a web app project to App Service by using Visual Studio.
- How to use the Azure portal to monitor and manage your web app.

Sign up for Microsoft Azure

You need an Azure account to complete this tutorial. You can:

- Open an Azure account for free. You get credits that you can use to try paid Azure services. Even after the credits are used up, you can keep the account and use free Azure services and features, such as the Web Apps feature of Azure App Service.
- Activate Visual Studio subscriber benefits. Your Visual Studio subscription gives you credits every month that you can use for paid Azure services.
- Get credits every month by joining to Visual Studio Dev Essentials.

If you want to get started with Azure App Service before you sign up for an Azure account, go to Try App Service. There, you can immediately create a short-lived starter web app in App Service without a credit card or commitments.

Set up the development environment

To start, set up your development environment by installing the latest version of the Azure SDK.

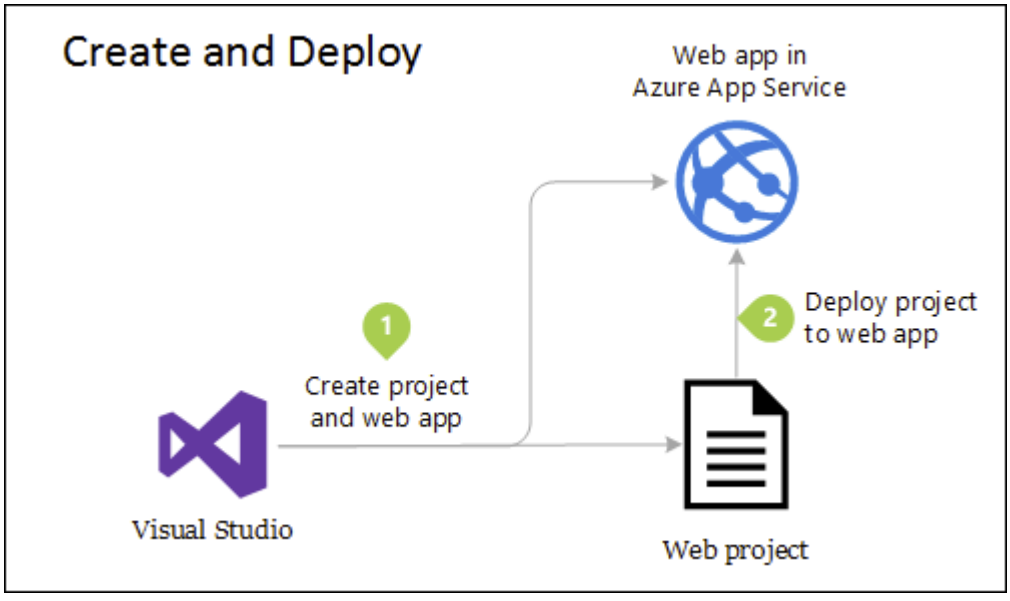
Visual Studio 2015▶ Visual Studio 2013▶

If you don't have Visual Studio installed, use the link for Visual Studio 2015, and Visual Studio will be installed along with the SDK.

Create a new project and a web app

Your first step is to create a web project in Visual Studio and a web app in Azure App Service. When that's done, you'll deploy the project to the web app to make it available on the Internet.

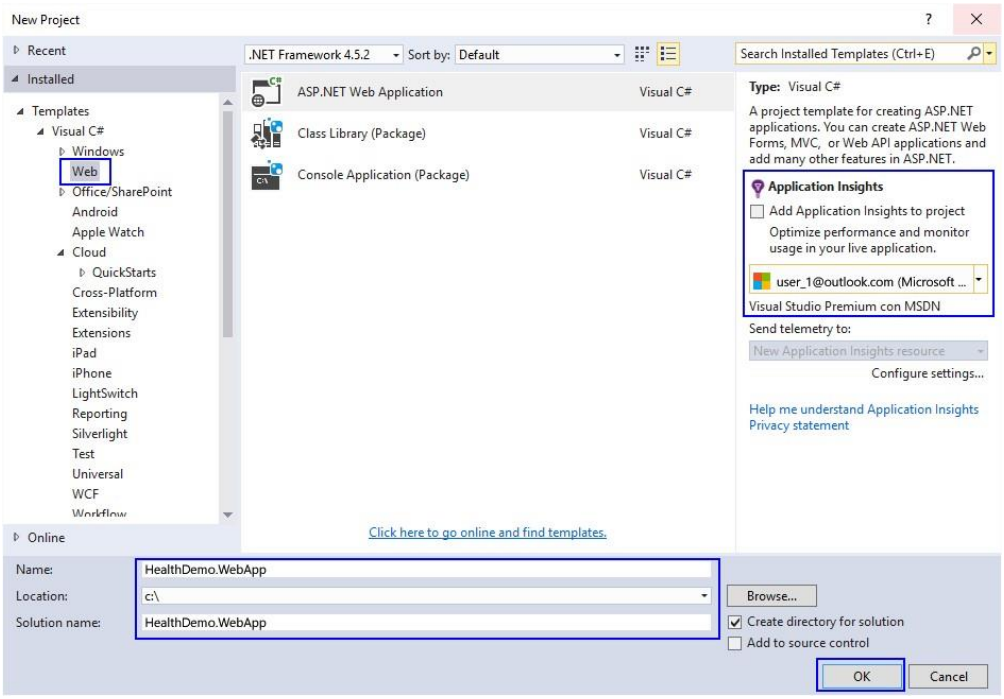
The diagram below illustrates what you're doing in the create and deploy steps.



1. Open Visual Studio 2015 or Visual Studio 2013.
2. On the **File** menu, click **New** > **Project**.
3. In the **New Project** dialog box, click **C#** > **Web** > **ASP.NET Web Application**.
4. Choose a good name for the project, for example, HealthDemo.WebApp.

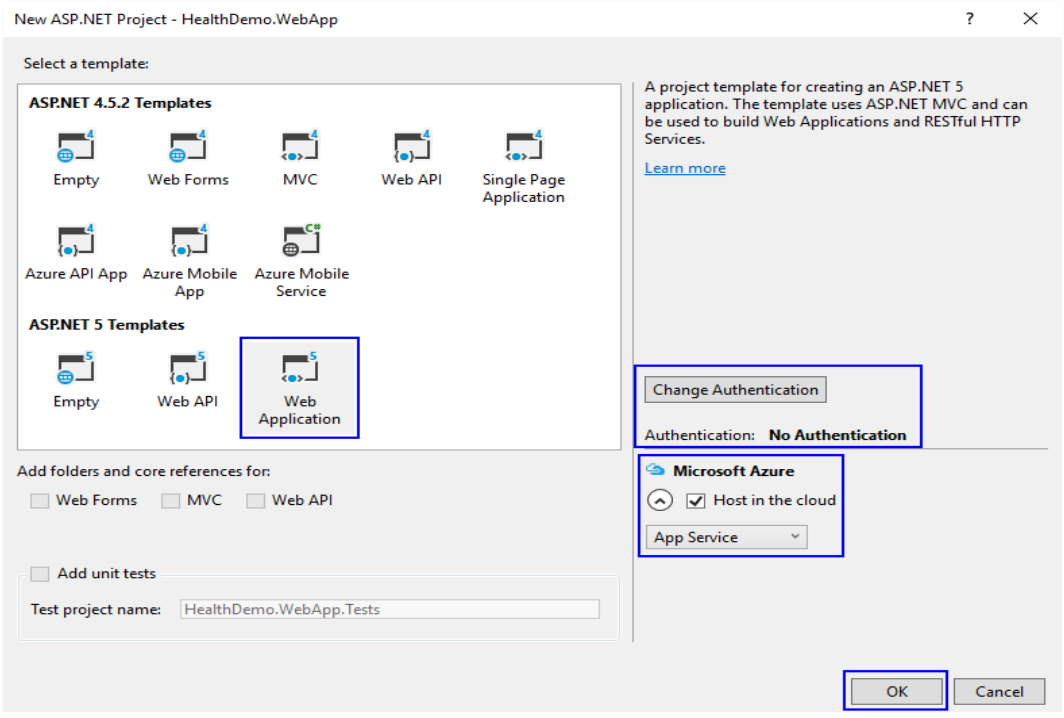
5. You can deactivate or activate **Application Insights** if you want your application to be monitored in terms of availability and performance by using Azure Application Insights.

6. Click **OK**.



{width="6.5in" height="4.510416666666667in"}

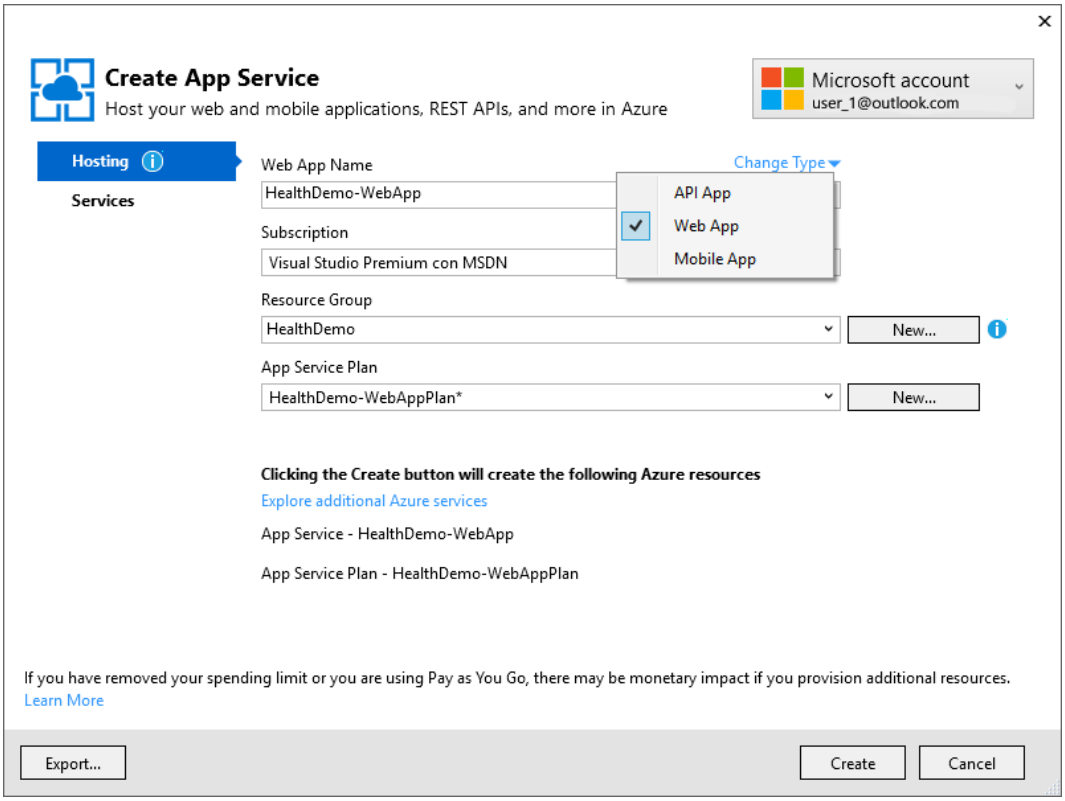
1. In the **Select a template** window, select **ASP.NET 5 Web Application**. On the other side, indicate that you want to host the **Web App** in **Azure**. Finally, indicate that you don't want authentication for the application.



{width="6.489583333333333in" height="5.0625in"}

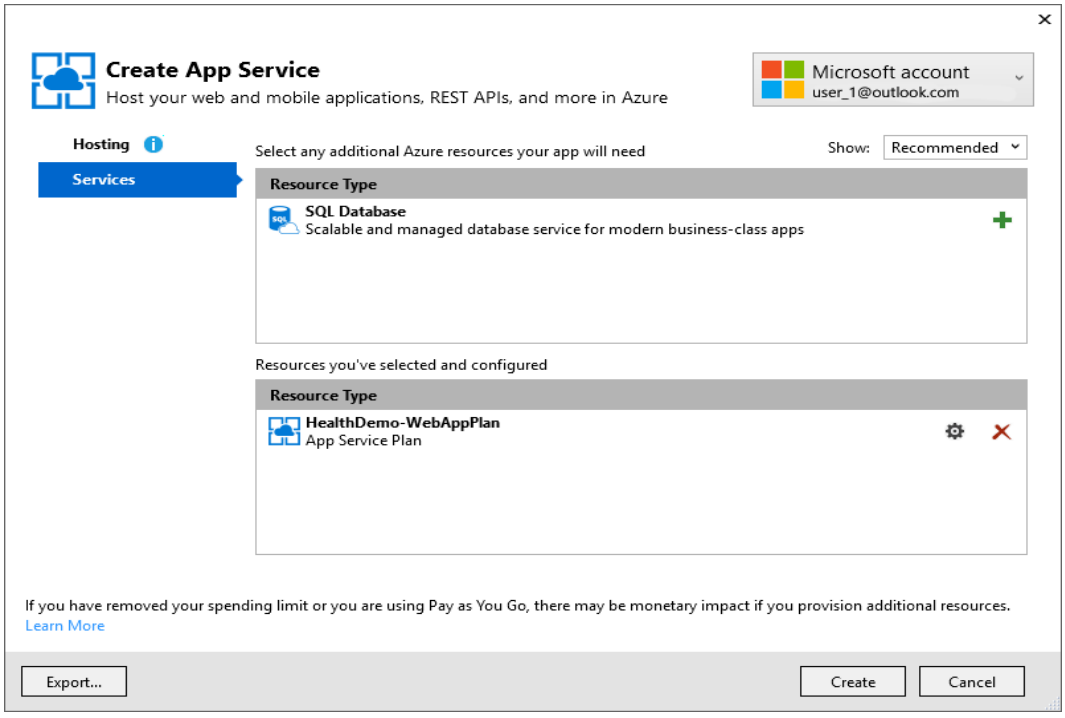
1. When you select Azure as the host, a new window will automatically open to indicate that the

resources that will be created in Azure, which will host all application resources. Fill in the required information and make sure to indicate the type of deployment as **Web App**.

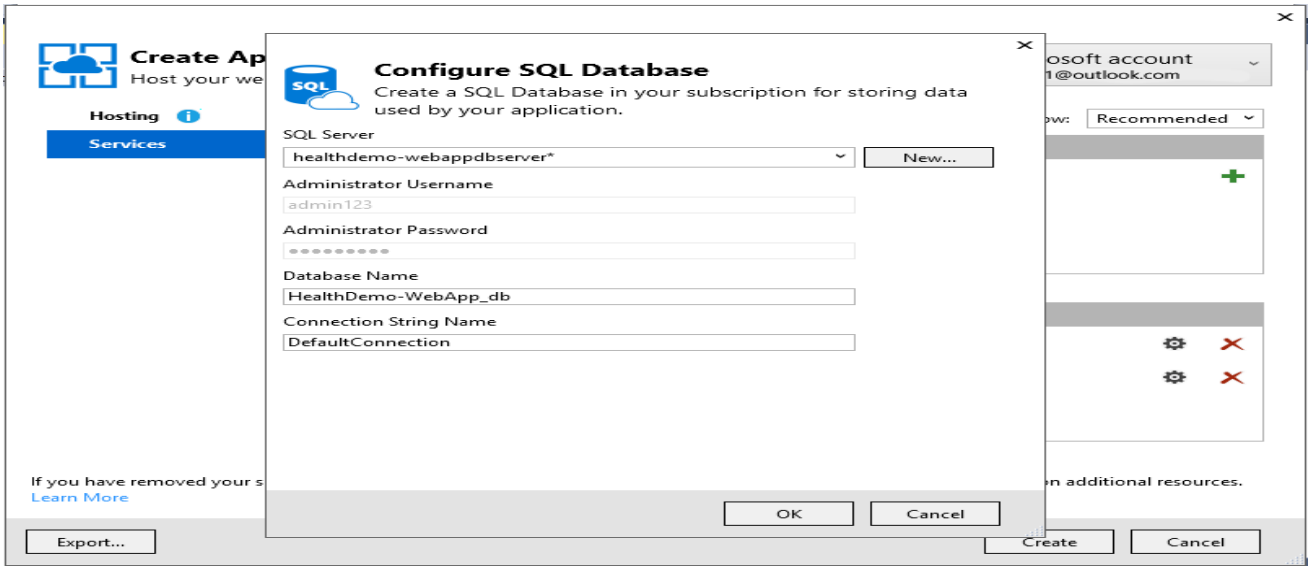


{width="6.489583333333333in" height="4.864583333333333in"}

1. Because your web application will have a SQL database in the future, you can add one as a service in the configuration window.

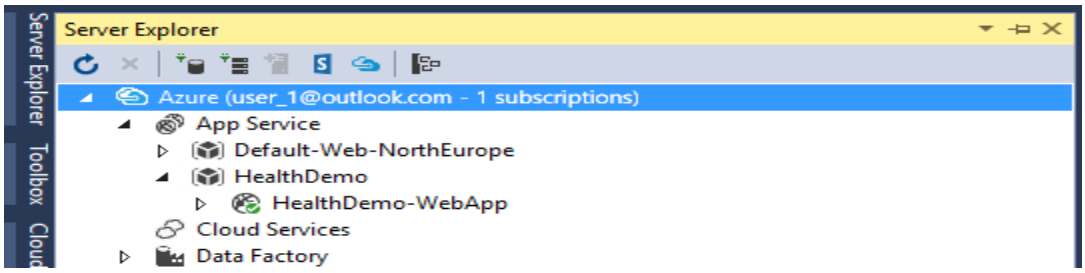


{width="6.489583333333333in" height="4.864583333333333in"}

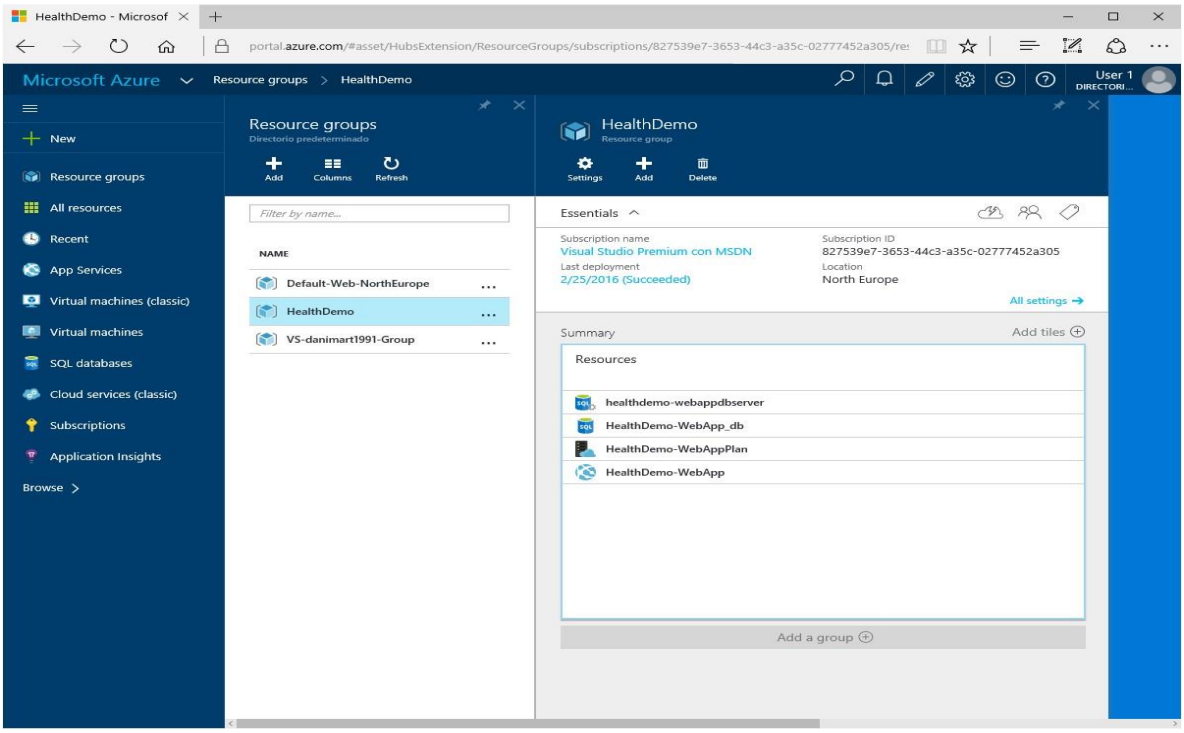


{width="6.489583333333333in" height="4.864583333333333in"}

1. When you click the **OK** button, Azure will begin to create the web app in Azure. After it is deployed, you can see all the information in the Azure portal and in Server Explorer in Visual Studio.



{width="5.614583333333333in" height="1.8645833333333333in"}



(width=”6.5in” height="5.041666666666667in"}

Azure Resource Manager templates

You can use an Azure Resource Manager template to provision the services that the applications need. In a single template, you can deploy multiple services along with their dependencies. In that way, you can use the same template to repeatedly deploy your application in different environments during every stage of the application lifecycle.

Azure Quickstart Templates are popular deployments from the community that you can adapt to the needs of the developed application.

Build the Connect(); //2015 Demo web app

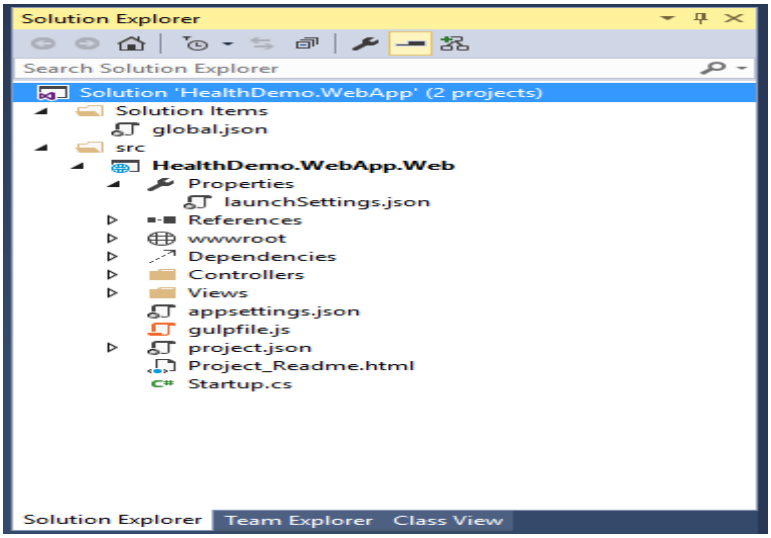
In this part of the tutorial, you learn how to develop the ASP.NET 5 Demo shown at Connect(); //2015. In this walkthrough, we show you how to create the skeleton and the structure of the solution. The completed project is in the following location:

Download the Demo solution, which is the result of this walkthrough, to see the structure and organization.

HealthDemo.WebApp.Web

Use the solution that you created in the previous section of this article, and rename the project as HealthDemo.WebApp.Web. The project contains the website that users see, the client-side code of the web application, and the definition of the view.

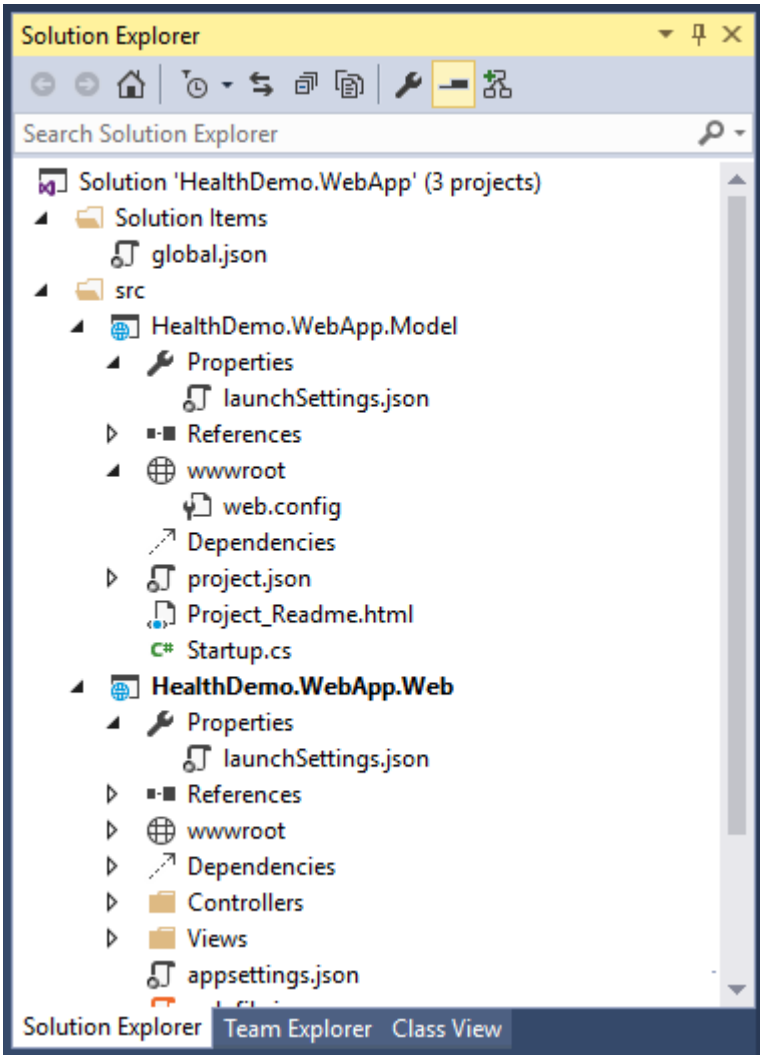
Make sure that you define this as a startup project and a project that will be deployed in Azure.This project will contain references to other projects including the one that's included in the solution. You will use the projects to add data, information, and context to the web application.



{width="3.96875in" height="5.458333333333333in"}

HealthDemo.WebApp.Model

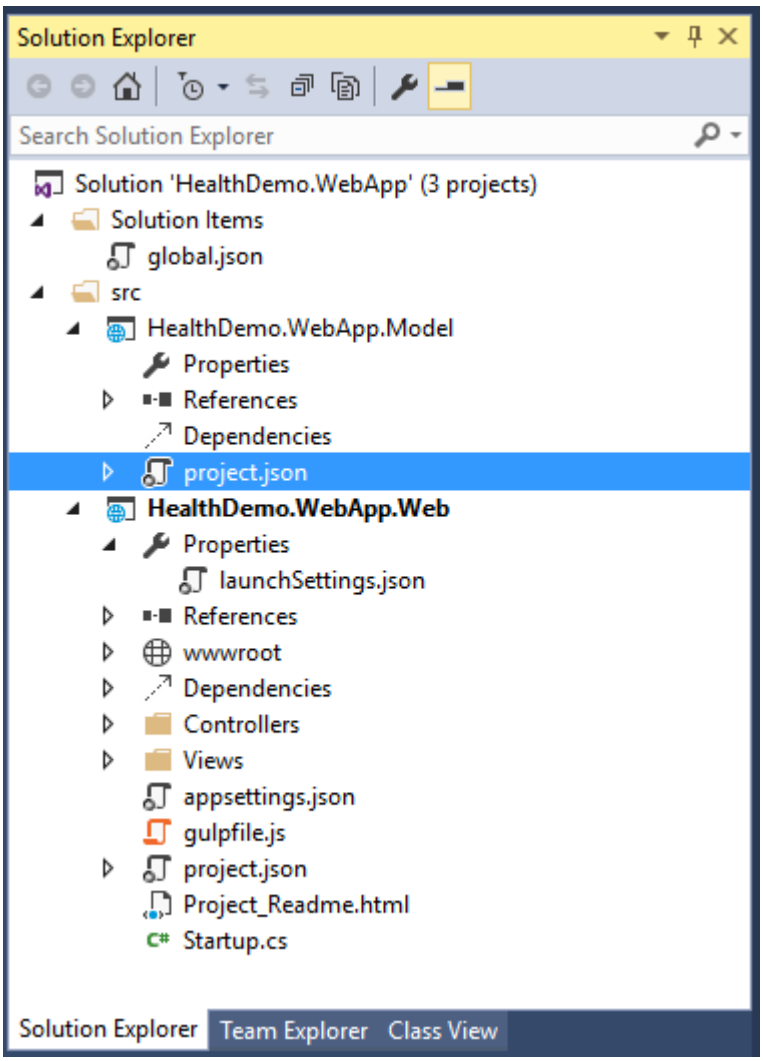
The following steps help you add an ASP.NET 5 empty project. You should call the project HealthDemo.WebApp.Model. The project contains the model files of the application. Model files are a list of classes that define the data model of the web application that the other projects will use.



{width="3.9375in" height="5.46875in"}

The following files that will be created must be deleted because they aren't necessary or useful:

- File Properties > launchSettings.json.
- Node wwwroot.
- File Project_Readme.html
- File Startup.cs

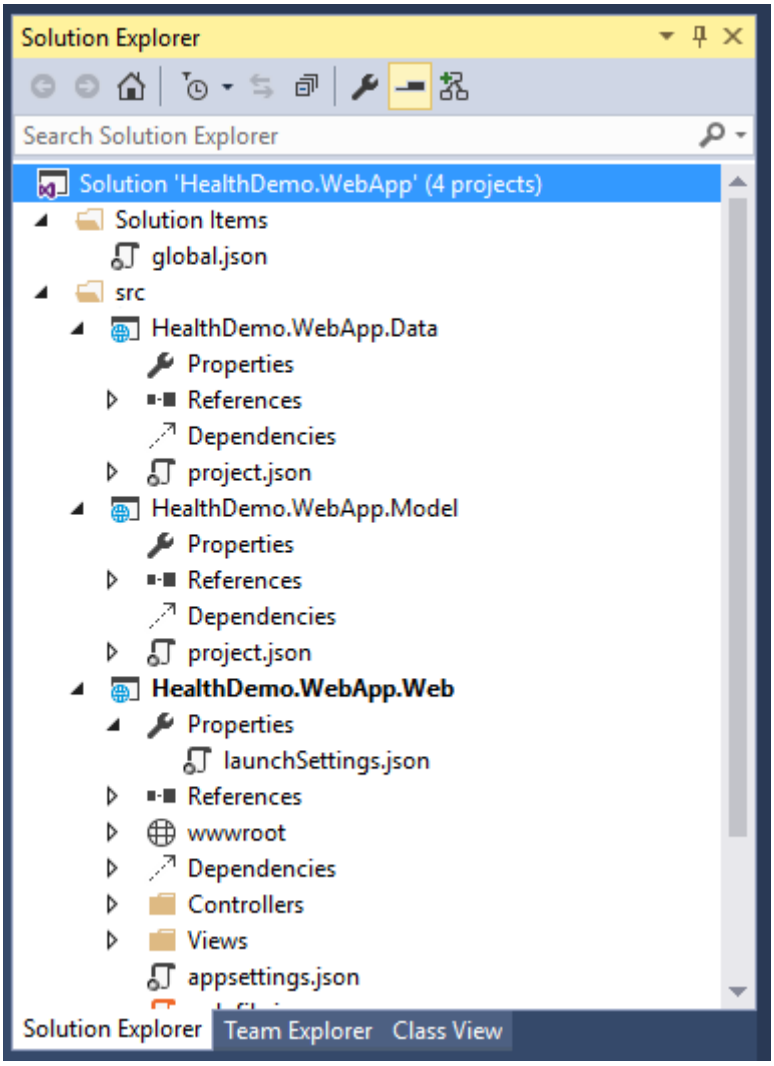


{width="3.9375in" height="5.46875in"}

Although you could create a Class Library project type , we recommend an ASP.NET project type to contain all references, dependencies, and components that a web project needs.

HealthDemo.WebApp.Data

Create an empty ASP.NET 5 project named HealthDemo.WebApp.Data. Like the previous project, you need to delete the same files and leave an empty project. This project will contain repositories, initial application data, and the infrastructure that is used in the database to store the information about the web application.



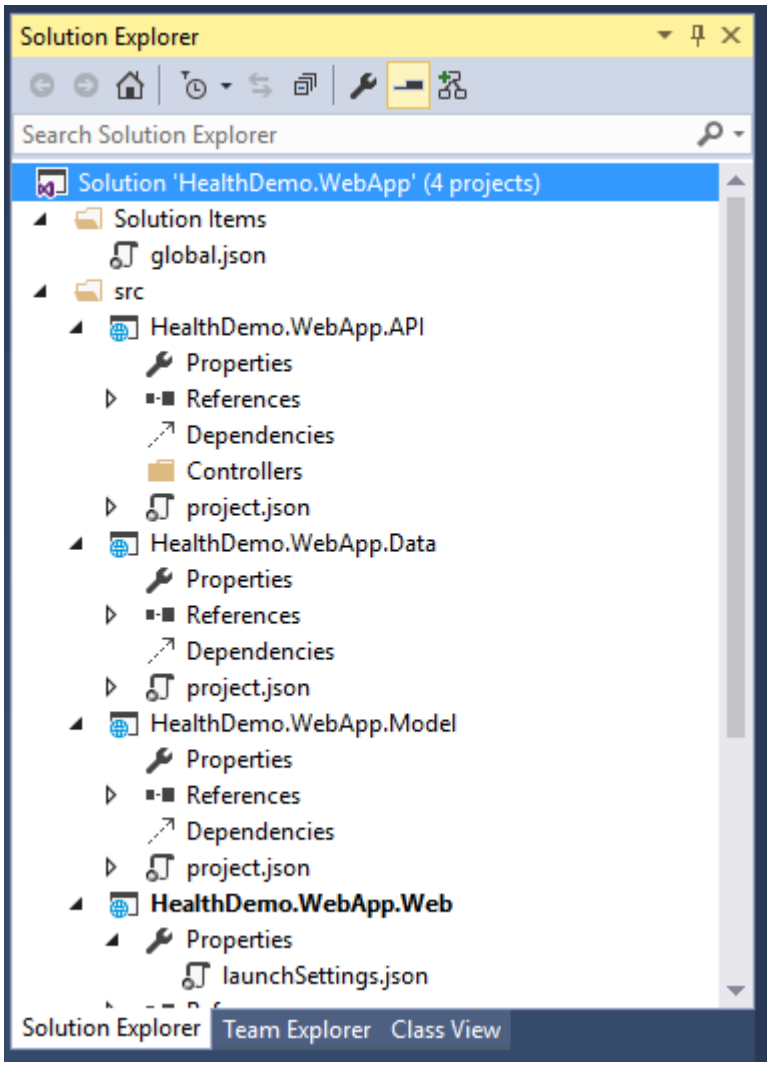
{width="3.9895833333333335in" height="5.5in"}

The MyHealthContext file in the project contains the database context as well as two directories:

- **Infrastructure**: Contains the data infrastructure that the web application uses, the data initialization, and the sample images that are used in the application itself.
- **Repositories**: Contains the database repositories that the web application uses. You can find the repositories that access the data in the database and that will be used by the controllers of the application.

HealthDemo.WebApp.API

Finally, create an ASP.NET 5 empty project named HealthDemo.WebApp.API. Delete the files that are indicated in the previous steps so that the final structure of the solution looks like the following illustration:



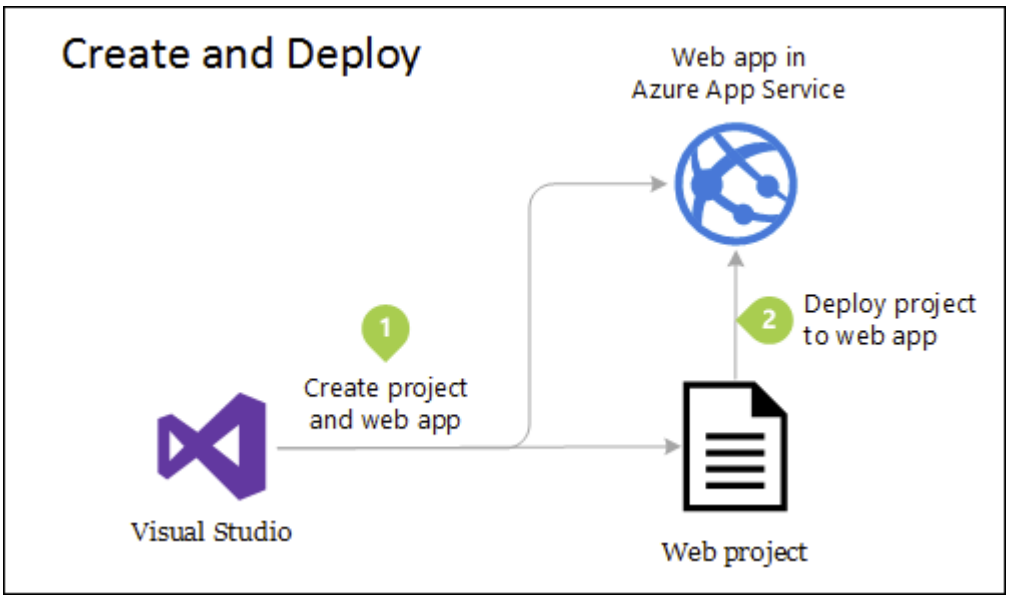
{width="3.96875in" height="5.5in"}

This project will contain the Web Application API. This is the web service that will provide the data to the web application. The apps have the controllers that have the calls that the application needs to access. The project also uses the repositories that are located in the Data project and the data model that's in the Model project.

From this point, you have the basic structure of the web application and the availability to deploy it in Azure. Details about the content of each of the projects can be consulted in the repository of the demo shown in Microsoft Connect(); //2015.

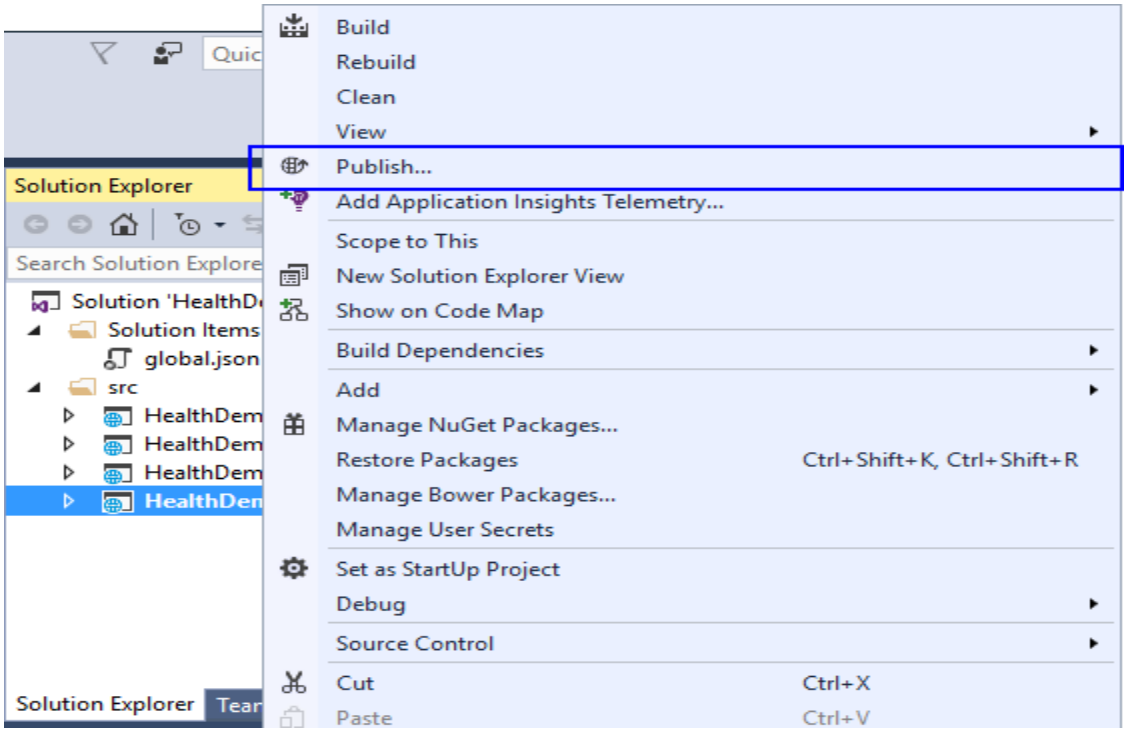
Deploy the project to a web app in Azure

To have the web application in an App Service in Azure, just follow a few steps.

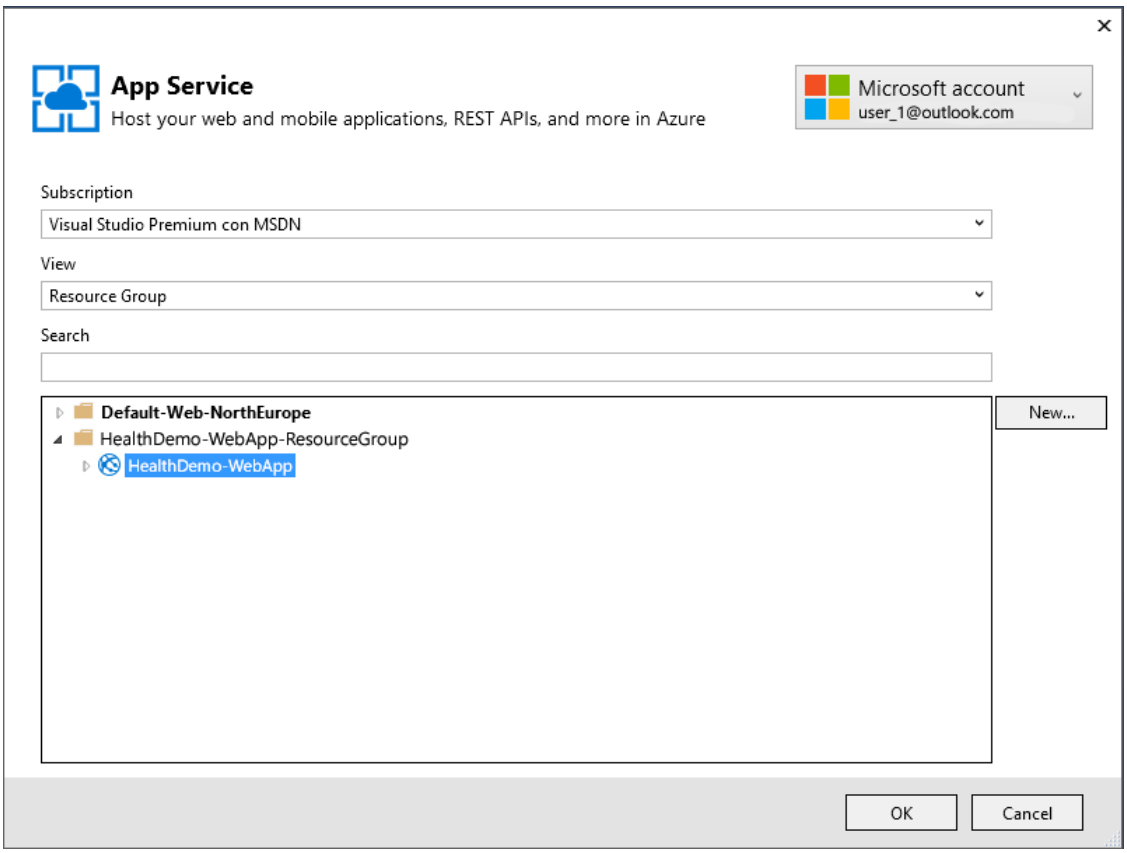


With the previous steps, you already generated the Azure resources that can help create more deployments.

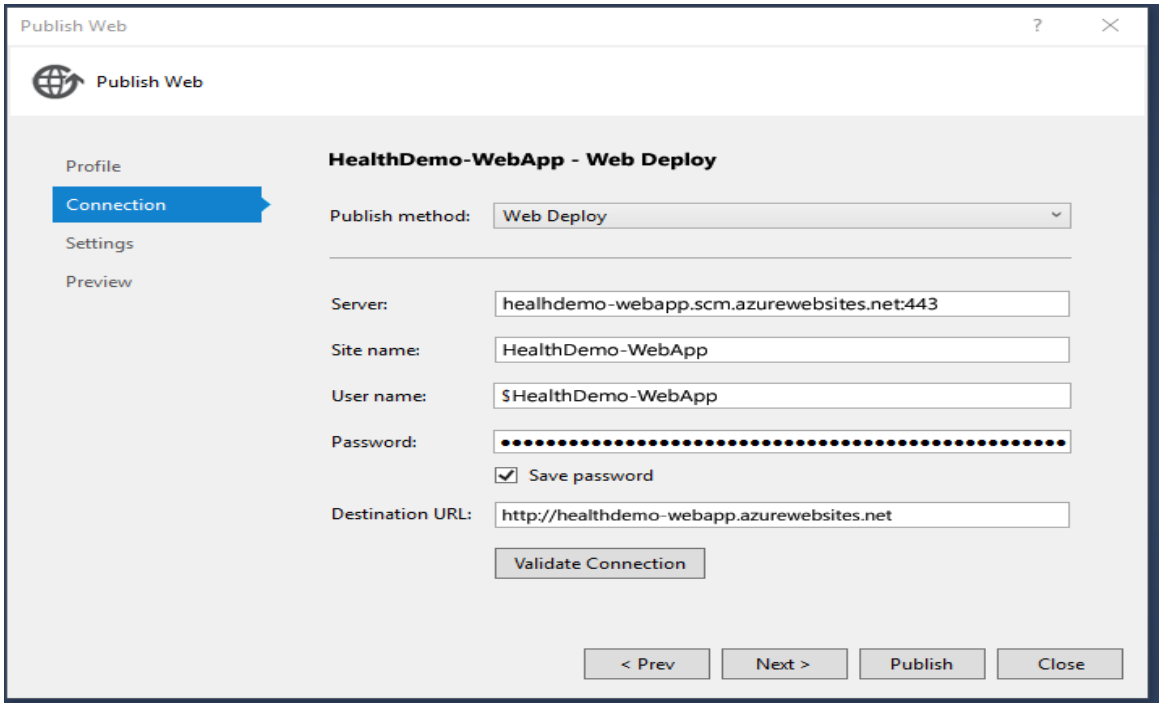
1. In Solution Explorer, right-click the HealthDemo.WebApp.Web project, and then click **Publish**.



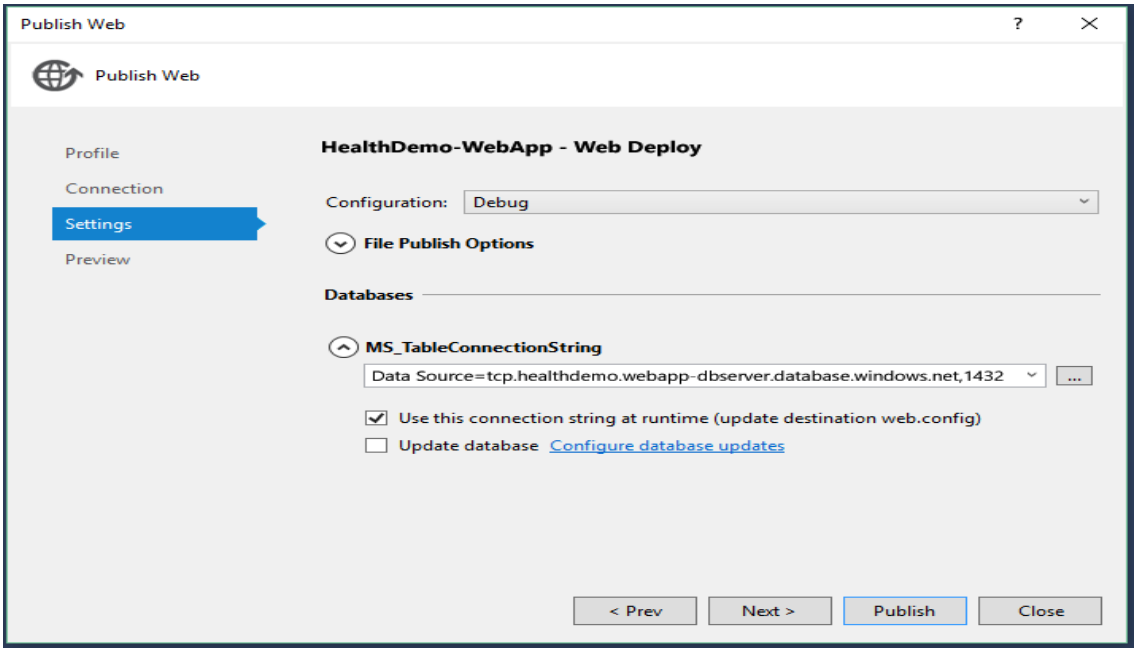
1. You will see the **Publish Web** dialog box. The wizard shows you a list of available **Publish**profiles. If you select **Microsoft Azure App Service**, you can see a list of available Azure subscriptions and the resource groups that were previously created. Among them is the resource group that you used to deploy the web application that you configured in the previous steps.



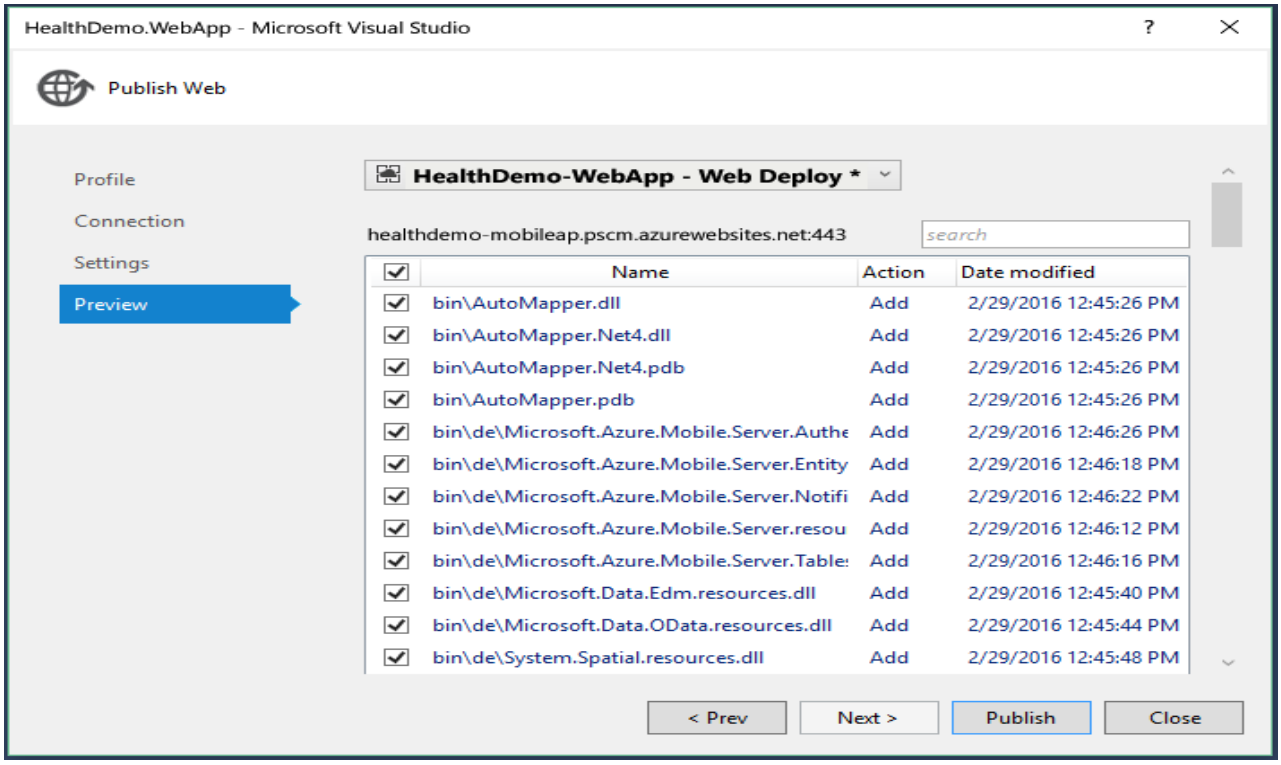
1. After you select the resource group, the page that opens shows the connection information. The default parameters will populate the fields. You can modify the fields if necessary. To check that the connection works correctly, click **Validate Connection**.



1. In the **Settings** page, you can configure the deployment type depending on whether you require a deployment in a production environment or in a debug environment. You can see the connection to the database.

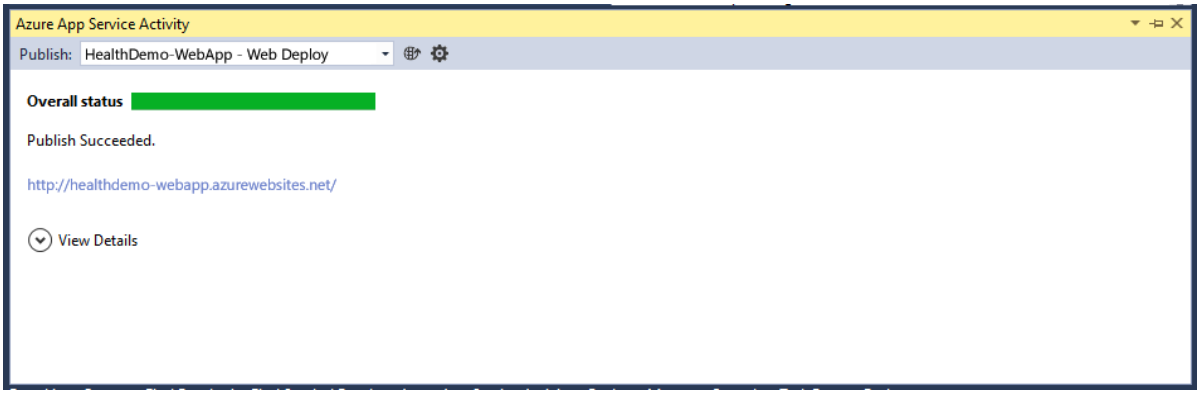


1. On the last page, **Preview**, you can determine the changes that affect the Azure environment.

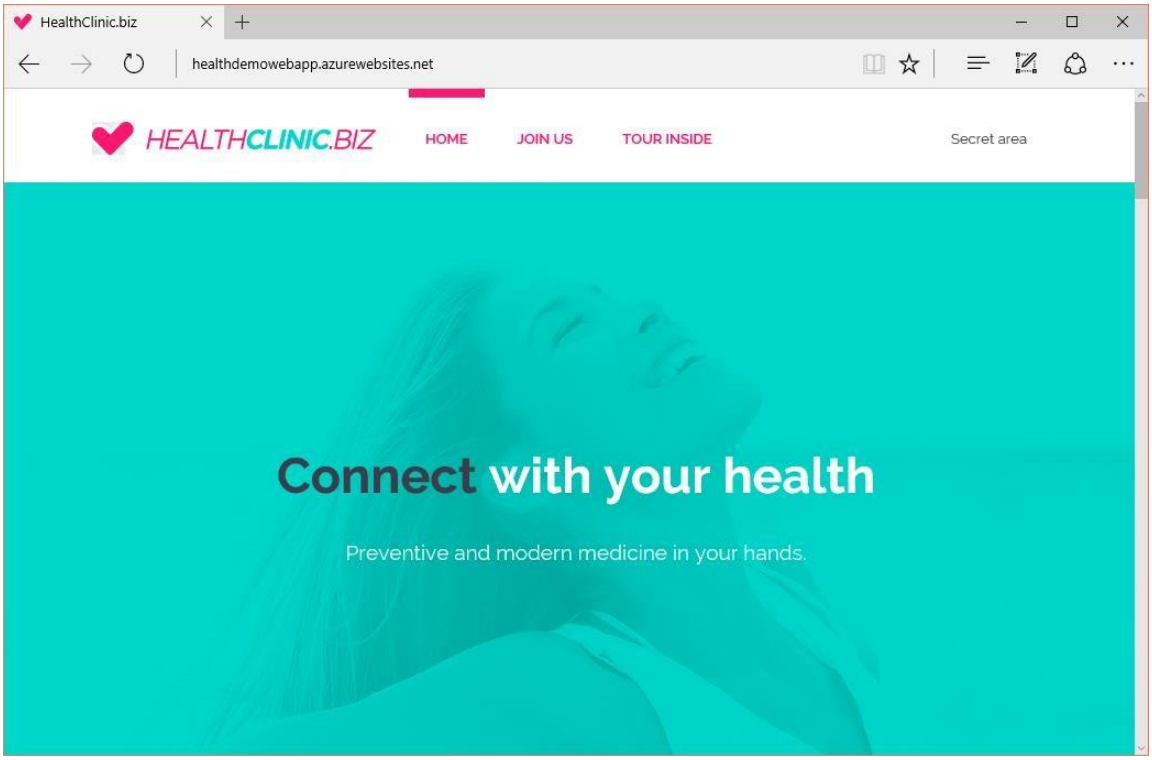


1. At last, click **Publish**.

The **Output** window displays information about the deployment. When it's finished, a message that everything has been completed successfully is displayed.



You can see the final result in the browser that will open.



5.4 Web forms - Basic Web Controls - Working with events

Basic Web Controls

1. ASP.NET Label Control

The Label control used to display text in a set location on a Web page, also it can customize the displayed text through the Text property.

HTML Code

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="blankpage.aspx.cs"
Inherits="asptutorial.blankpage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

  <html xmlns="http://www.w3.org/1999/xhtml">

  <head runat="server">

  <title> ASP.NET </title>

  </head>

  <body>

  <form id="form1"
  runat="server">

<div>

<asp:Label ID="Label1" runat="server" Text="Welcome to ASP.NET Tutorial">

      </div>

      </for
      m>

   </bo
   dy>

   </ht
   ml>
```

Properties

Property	Description
runat	Specifies that the control is a server control. Must be set to "server"
Text	The text to display in the label

ASP.NET Control Standard Properties

AppRelativeTemplateSourceDirectory, BindingContainer, ClientID, Controls, EnableTheming, EnableViewState, ID, NamingContainer, Page, Parent, Site, TemplateControl, TemplateSourceDirectory, UniqueID, Visible

For a full description, go to Web Control Standard Attributes.

ASP.NET Web Control Standard Properties

AccessKey, Attributes, BackColor, BorderColor, BorderStyle, BorderWidth, CssClass, Enabled, Font, EnableTheming, ForeColor, Height, IsEnabled, SkinID, Style, TabIndex,

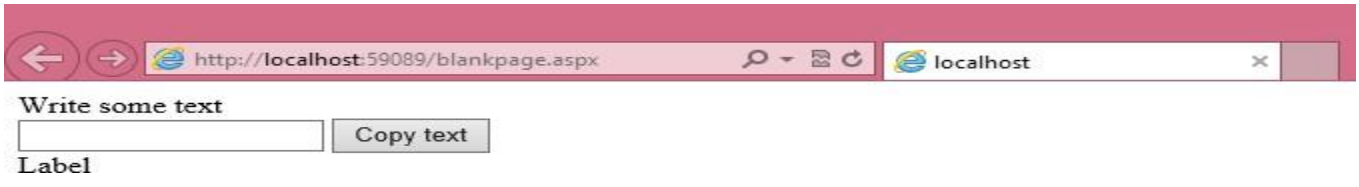
For a full description, go to Web Control Standard Attributes.

Example of Label Control

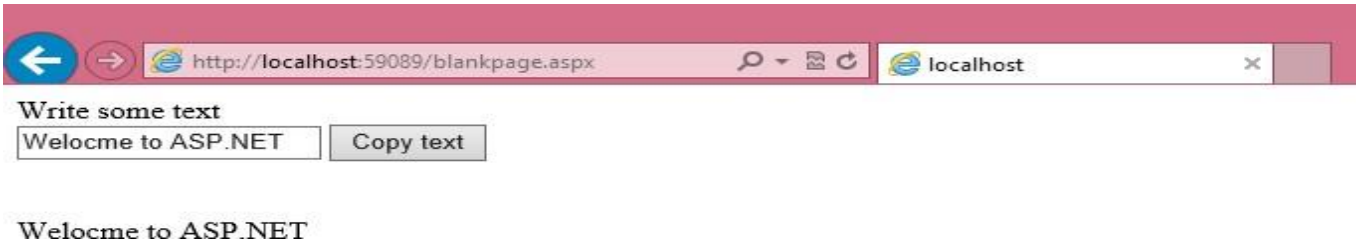
Declare one Label control, one TextBox control, and one Button control in an .aspx file. When the user cl1icks on the button, the submit subroutine is executed. The subroutine copies the content of the TextBox control to the Label control.

HTML Code

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title> ASP.NET </title>

</head>

<body>

<form id="form1"
runat="server">

<div>

        <asp:Label ID="Label1" runat="server" Text="Welcome to ASP.NET Tutorial">

        <br/>

        <asp:TextBox ID="TextBox1" runat="server">

        <asp:Button ID="Button1" runat="server" onclick="Button1_Click"Text="Button"
        />
        <br />

        <asp:Label ID="Label2" runat="server" Text="Label">

        </div>

          </form>

          </body>


        </html>
```



Output



2. ASP.NET Button Control

- The Button control is used to display a push button. The push button may be a submit button or a command button. By default, this control is a submit button.
- A submit button does not have a command name and it posts the Web page back to the server when it is clicked. It is possible to write an event handler to control the actions performed when the submit button is clicked.
- A command button has a command name and allows you to create multiple Button controls on a page. It is possible to write an event handler to control the actions performed when the command button is clicked.

HTML Code

```
<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title> Button Control </title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Button ID="Button1" runat="server" onclick="Button1_Click"Text="Button" />

<asp:Label ID="Label1" runat="server" Text="Label1"/>
```

```
</div>
    </form>
</body>
</html>
```

Properties

Property	Description
CausesValidation	Specifies if a page is validated when a button is clicked
CommandArgument	Specifies additional information about the command to perform
CommandName	Specifies the command associated with the Command event
OnClientClick	Specifies the name of the function to be executed when a button is clicked
PostBackUrl	Specifies the URL of the page to post to from the current page when a button is clicked
runat	Specifies that the control is a server control. Must be set to "server"
Text	Specifies the text on a button
UseSubmitBehavior	Specifies whether or not a button uses the browser's submit mechanism or the ASP.NET postback mechanism
ValidationGroup	Specifies the group of controls a button causes validation, when it posts back to the server

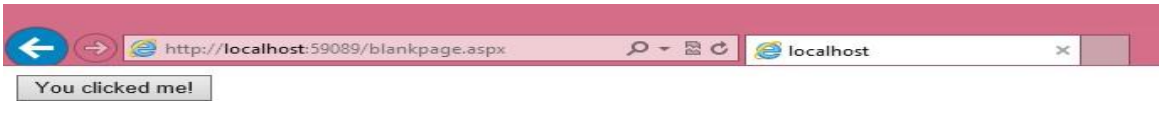
ASP.NET Control Standard Properties

AppRelativeTemplateSourceDirectory, BindingContainer, ClientID, Controls, EnableTheming, EnableViewState, ID, NamingContainer, Page, Parent, Site, TemplateControl, TemplateSourceDirectory, UniqueID, Visible

HTML Code

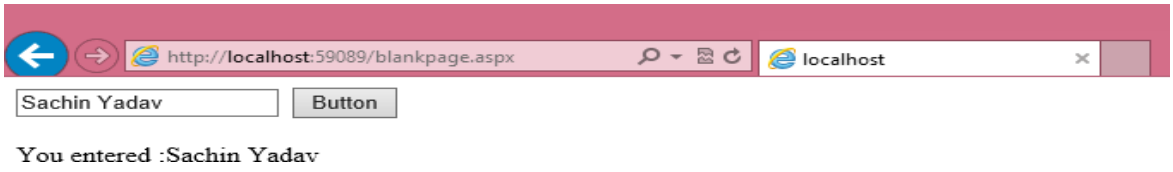
```
<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title> Button Control </title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Button ID="Button1" runat="server" onclick="Button1_Click"Text="Button" />

    </div>

    </for
    m>

</bo
dy>

</ht
ml>
```

Output



3. ASP.NET Textbox Control

The TextBox control is used to create a text box where the user can input text.



HTML Code

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title> Button Control </title>
  </head>
  <body>
  <form id="form1" runat="server">
<div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <asp:Button ID="Button1" runat="server" onclick="Button1_Click"Text="Button" />
        <br
        />
        <br
        />
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        <br />
</div>
  </form>
  </body>
</html>
```

ASP.NET - Event Handling

An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events. For example, interrupts are system-generated events. When events occur, the application should be able to respond to it and manage it.

Events in ASP.NET raised at the client machine, and handled at the server machine. For example, a user clicks a button displayed in the browser. A Click event is raised. The browser handles this client-side event by posting it to the server.

The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the Click event has an associated event handler. If it has, the event handler is executed.

Event Arguments

ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.

The general syntax of an event is:

```
private void EventName (object sender, EventArgs e);
```

Application and Session Events

The most important application events are:

- **Application_Start** - It is raised when the application/website is started.
- **Application_End** - It is raised when the application/website is stopped.

Similarly, the most used Session events are:

- **Session_Start** - It is raised when a user first requests a page from the application.
- **Session_End** - It is raised when the session ends.

Page and Control Events

Common page and control events are:

- **DataBinding** - It is raised when a control binds to a data source.
- **Disposed** - It is raised when the page or the control is released.
- **Error** - It is a page event, occurs when an unhandled exception is thrown.
- **Init** - It is raised when the page or the control is initialized.
- **Load** - It is raised when the page or a control is loaded.
- **PreRender** - It is raised when the page or the control is to be rendered.
- **Unload** - It is raised when the page or control is unloaded from memory.

  Event Handling Using Controls

  All ASP.NET controls are implemented as classes, and they have events which are fired when a user performs a certain action on them. For example, when a user clicks a button the 'Click' event is generated. For handling events, there are in-built attributes and event handlers. Event handler is coded to respond to an event, and take appropriate action on it.

  By default, Visual Studio creates an event handler by including a Handles clause on the Sub procedure. This clause names the control and event that the procedure handles.

  The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" />
```

  The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
  btnCancel.Click
  End Sub
```

  An event can also be coded without Handles clause. Then, the handler must be named according to the appropriate event attribute of the control.

  The ASP tag for a button control:

```
<asp:Button ID="btnCancel" runat="server" Text="Cancel" Onclick="btnCancel_Click" />
```

The event handler for the Click event:

```
Protected Sub btnCancel_Click(ByVal sender As Object, ByVal e As System.EventArgs)


```

The common control events are:

Event	Attribute	Controls
Click	OnClick	Button, image button, link button, image map
Command	OnCommand	Button, image button, link button
TextChanged	OnTextChanged	Text box
SelectedIndexChanged	OnSelectedIndexChanged	Drop-down list, list box, radio button list, check box list.
CheckedChanged	OnCheckedChanged	Check box, radio button

Some events cause the form to be posted back to the server immediately, these are called the postback events. For example, the click event such as, Button.Click.

Some events are not posted back to the server immediately, these are called non-postback events.

For example, the change events or selection events such as TextBox.TextChanged or CheckBox.CheckedChanged. The nonpostback events could be made to post back immediately by setting their AutoPostBack property to true.

Default Events

The default event for the Page object is Load event. Similarly, every control has a default event. For example, default event for the button control is the Click event.

The default event handler could be created in Visual Studio, just by double clicking the control in design view. The following table shows some of the default events for common controls:

Control	Default Event
AdRotator	AdCreated
BulletedList	Click
Button	Click
Calender	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
DataGrid	SelectedIndexChanged
DataList	SelectedIndexChanged
DropDownList	SelectedIndexChanged
HyperLink	Click
ImageButton	Click
ImageMap	Click
LinkButton	Click
ListBox	SelectedIndexChanged
Menu	MenuItemClick
RadioButton	CheckedChanged
RadioButtonList	SelectedIndexChanged

5.5 Rich Web Controls : AdRotator Control - Calendar Control - Custom web Controls - Validation controls

ASP.NET -Ad Rotator

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

The AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator runat = "server" AdvertisementFile = "adfile.xml" Target = "_blank" />
```

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

Extensible Markup Language (XML) is a W3C standard for text document markup. It is a text-based markup language that enables you to store data in a structured format by using meaningful tags. The term 'extensible' implies that you can extend your ability to describe a document by defining meaningful tags for the application.

XML is not a language in itself, like HTML, but a set of rules for creating new markup languages. It is a meta-markup language. It allows developers to create custom tag sets for special uses. It structures, stores, and transports the information.

Following is an example of XML file:

```
<BOOK>
  <NAME> Learn XML </NAME>
  <AUTHOR> Samuel Peterson </AUTHOR>
  <PUBLISHER> NSS Publications </PUBLISHER>
  <PRICE> $30.00</PRICE>
</BOOK>
```

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. There are the following standard XML elements that are commonly used in the advertisement file:

Element	Description
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.
Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.
Width	Width of the image to be displayed.

Apart from these tags, customs tags with custom attributes could also be included. The following code illustrates an advertisement file ads.xml:

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
```

```
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
    </Ad>
    <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
    </Ad>
    <Ad>
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
    </Ad>
    <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
      <NavigateUrl>http://www.edibleblooms.com</NavigateUrl>
      <AlternateText>Edible Blooms</AlternateText>
      <Impressions>20</Impressions>
      <Keyword>gifts</Keyword>
   </Ad>
</Advertisements>
```

Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	Description
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.
DataSourceID	Id of the control from where it would retrieve data.
Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.
Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation

	of the control, but before the page is rendered
DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after the server control binds to a data source.
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

Working with AdRotator Control

Create a new web page and place an AdRotator control on it.

```
<form id="form1" runat="server">

   <div>

      <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile ="~/ads.xml"
onadcreated="AdRotator1_AdCreated" />

   </div>

</form>
```

The ads.xml file and the image files should be located in the root directory of the web site.

ASP.NET – Calendars

The calendar control is a functionally rich web control, which provides the following capabilities:

- Displaying one month at a time
- Selecting a day, a week or a month
- Selecting a range of days
- Moving from month to month
    - Controlling the display of the days programmatically The basic syntax of a calendar control is:

```
<asp:Calender ID = "Calendar1" runat = "server">

</asp:Calender>
```

Properties and Events of the Calendar Control

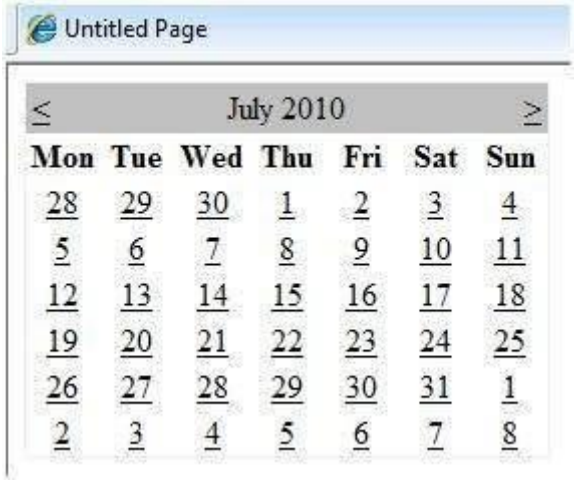
The calendar control has many properties and events, using which you can customize the actions and display of the control. The following table provides some important properties of the Calendar control:

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	Description
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Working with the Calendar Control

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.



Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

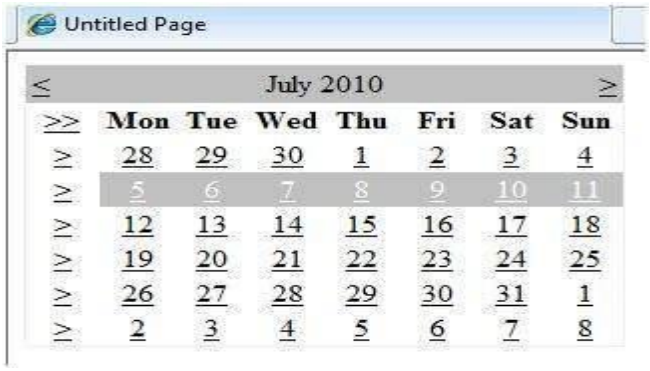
Properties	Description
Day	To select a single day.

DayWeek	To select a single day or an entire week.
DayWeekMonth	To select a single day, a week, or an entire month.
None	Nothing can be selected.

The syntax for selecting days:

```
<asp:Calender ID = "Calendar1" runat = "server" SelectionMode="DayWeekMonth">

</asp:Calender>
```

When the selection mode is set to the value DayWeekMonth, an extra column with the > symbol appears for selecting the week, and a >> symbol appears to the left of the days name for selecting the month.



Example

The following example demonstrates selecting a date and displays the date in a label: The

content file code is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="calendardemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

  <head runat="server">

    <title>

      Untitled Page
```

```
      </title>
    </head>
      <body>
      <form id="form1" runat="server">
           <div>
           <h3> Your Birthday:</h3>
          <asp:Calendar ID="Calendar1" runat="server SelectionMode="DayWeekMonth"
onselectionchanged="Calendar1_SelectionChanged">
          </asp:Calendar>
        </div>
            <p>Todays date is:
          <asp:Label ID="lblday" runat="server"></asp:Label>
        </p>
            <p>Your Birday is:
          <asp:Label ID="lblbday" runat="server"></asp:Label>
        </p>
         </form>
  </body>
</html>
```

The event handler for the event SelectionChanged:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
  lblday.Text = Calendar1.TodaysDate.ToShortDateString(); lblbday.Text =
  Calendar1.SelectedDate.ToShortDateString();
}
```

When the file is run, it should produce the following output:

Your Birthday:

Asp.Net – Validators

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

1. RequiredFieldValidator
2. RangeValidator
3. CompareValidator
4. RegularExpressionValidator
5. CustomValidator
6. ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnEr	It indicates whether in case of an invalid control, the focus should

ror

	switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator Control

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate" runat="server" ControlToValidate ="ddlcandidate"
ErrorMessage="Please choose a candidate" InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range. It has three specific properties:

Properties	Description
Type	It defines the type of the data. The available values are: Currency, Date, Double, Integer, and String.
MinimumValue	It specifies the minimum value of the range.
MaximumValue	It specifies the maximum value of the range.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)" MaximumValue="12" MinimumValue="6"
Type="Integer">
</asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description

Type	It specifies the data type.
ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ErrorMessage="CompareValidator">
    </asp:CompareValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event

The server side validation routine should be written in any .Net language, like C# or

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
    ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

ValidationSummary

- The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error

handler.

ShowSummary : shows the error messages in specified format.

ShowMessageBox : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server" DisplayMode = "BulletList"
ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, you should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

5.6 ASP.NET APPLICATION LOCALIZATION

ASP.NET is one of the most common web application development platforms. As with any other
.NET framework application, ASP.NET provides great support for localization and globalization straight out of the box. Following best practices from the beginning of development will result in a properly internationalized application, making ASP.NET application localization an enjoyable process. In order to save money, reduce localization/internationalization defects and shorten the time to market, it's important to consider localization as part of the development process.

ASP.NET Application Localization: What Needs to be Localized/Translated?

In a typical ASP.NET application, the following four items may differ depending on the user's language and regional preferences:

Text resources – The text that resides in the aspx page. Both asp server tags and traditional html tags may contain text on an aspx page. This includes the UI text that may appear in C-Sharp(C#) and Visual Basic code behind files.

Database content – Most of the translatable text resides in a database for a typical ASP.NET application particularly in content management systems (product information, articles etc.).

Images and graphics – Graphics and images may contain translatable text.

Regional options – Date/time, currency, number/decimal formatting will vary depending on the user's region and/or preferences.

Text Resources

All text resources should be externalized to resx files in order to make translation and maintenance easier. Also known as “resources file”, a resx file consists of XML entries which specify objects and strings. Visual Studio does provide an automated way to externalize the strings after the page has been created but this function is limited. It only works on the server tags on aspx pages. Visual Studio cannot externalize the text in CS/VB code or process regular html tags.

The best approach is to externalize the tags as the pages are being developed. More information on the use of resx files for ASP.NET localization can be found on MSDN.

Database Content

The database schema and relations will have to be carefully designed to externalize translatable text to a table(s) which maps to languages and parent tables. Although adding additional columns to existing tables for each language may seem easier; maintenance of this solution is extremely difficult and not recommended.

Images and Graphics

If possible, try avoiding images and graphics with embedded text. Using background images and retrieving the text from a resource file will make the localization effort much easier. If that is not possible, point the source of the image to the resx file which will let you easily change it depending on the language. This applies to other external assets such as pdf and doc files as well.

Regional Options

As in any other programming language and platform, avoid manually formatting and hard coding date/time and number formats. .NET offers a CultureInfo class which provides access to culture- specific instances of objects such as: DateTimeFormatInfo and NumberFormatInfo. Using these objects, culture-specific operations can be performed easily such as formatting dates and numbers, casing and comparing strings.

Asp.net localization and globalization

Most websites on the internet today are designed and developed to support a single language and culture, but making a website or web application to support different languages and cultures is not difficult and is relatively easy to implement in ASP.Net.

When a web application supports globalization, it means that the application can render it contents based on the language preference set by a user in his or her browser. Localization involves translating a web application into different languages and cultures.

Making an already deployed website to support globalization is not an easy task, thus it is always advised to make an application support globalization right from the development stage, even if there is no current plan to make it support more than one language. If later there is a decision to translate the application to another language, it will be easy to do, since the structure has already been put in place.

ASP.Net makes it easy to localize an application through the use of resource files. Resource files are xml files with .resx extension. Resources can either be a global resource, in which it is accessible throughout the site and placed in App_GlobalResources folder in a website or a local resource which is specific to a page only and is placed in App_LocalResources folder.

Now, lets get started, we are going to have a page that displays Good Morning in four different Languages(French, Spanish, German and Yoruba). Create a new ASP.Net website in Visual Studio and name it localizationexample, right click the website in solution explorer and select Add ASP.Net Folder then select App_GlobalResources. After that, right click the App_GlobalResources and select Add New Item in the list of items displayed in the dialog box then select Resource File , give it any name you like, but in this case I name it content.resx this will serve as the default resource file that will be selected if the language selected in the user browser is not available. Note that of great importance is the languageid and cultureid because ASP.Net checks the languageid and cultureid combination on the browser to determinge the resource file to be used in rendering contents to that browser, to see the detailed list of language and culture identification click here.

The next thing is to create our language specific resource file, for the four languages that the localizationexample website will be supporting, the culture names are

fr for French de for German

es for Spanish and yo-NG for Yoruba

Apart from the content.resx file in the App_GlobalResources four other resource files with the format resourcename.languageId-cultureId.resx are to be created content.de.resx, content.es.resx, content.fr.resx and content.yo-NG.resx

Global resource files

A resource file takes a key value pair, so all the resource files will contain the same key but the different values as the language translation may be, so I have a key Greeting in all the files with their values.

Resource file **Key** **Value** content.resx Greeting - Good Morning content.de.resx

Greeting - Guten Morgen content.es.resx Greeting - buenos días

content.fr.resx Greeting - bonjour content.yo-NG.resx Greeting - E kaa ro

The next thing to do is add a Label control that will be used to display the localized text to the Default.aspx page, then switch to the page source view in visual studio and set the Text property of the Label control to Text="< %$ Resources:content, Greeting % >"

Where Resource:content specifies that we are retrieving the content from a resource file named content and the value to be retrieved from the resource file has the name or key Greeting. Note that content was used and not content.de or any of the remaining three resource files, because this is our default resource file that ASP.Net will always use should in case we do not have translation for the user's browser language preference.

We need to override the page's InitializeCulture() method and set the page UICulture property, this property determines which global or local resource to be loaded for the page. The browser's topmost language preference will be obtainted by Request.UserLanguages[0] and then call the page’s base InitializeCulture method.

```
protected override void InitializeCulture()

{

UICulture = Request.UserLanguages[0]; base.InitializeCulture();

}
```