

## XAMPP

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** Install XAMPP in Ubuntu or Windows.

### Theory

*How to Install XAMPP for Linux in Ubuntu 14.04 LTS*

**Step 1 Download XAMPP set up files from the Following link:**

<https://www.apachefriends.org/download.html>

From the above link you will get the both the setup file for Ubuntu and Windows.

**Step 2 after download change the permission**

Open Terminal and write down following commands one by one.

```
cd ~/Downloads
```

```
sudo su
```

```
sudo chmod 777 -R xampp-linux-x64-1.8.3-4-installer.run
```

```
./xampp-linux-x64-1.8.3-4-installer.run
```

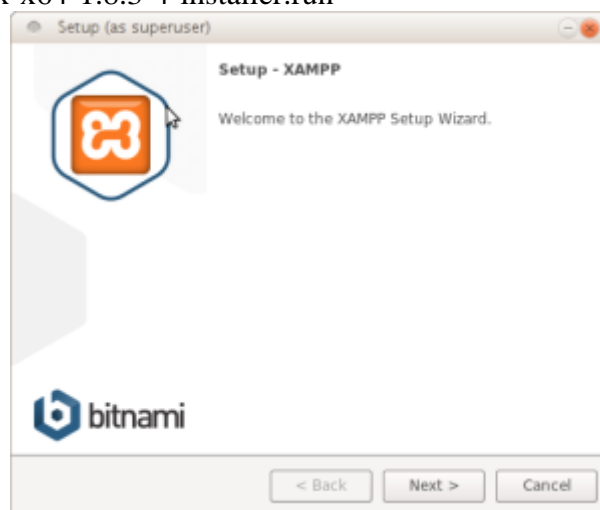


Fig. 1: XAMPP Installation wizard.

**Step 4 and then click next and wait until finish**

```
exit
```

```
exit
```

**Step 5 Now need to change the permission for htdocs:**

```
cd /opt/lampp
```

```
sudo chmod 777 htdocs
```

**Step 6 create one demo.php file and write code:**

```
<?php
```

```
echo "Hello World !";
```

```
?>
```

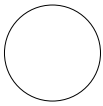
**Step 7 to start the XAMPP service:**

```
sudo su
```

```
/opt/lampp/lampp start
```

```
exit
```

```
exit
```

**Step 8 To stop the XAMPP service:**

```
sudo su  
/opt/lampp/lampp stop  
exit  
exit
```

**Step 9 To open the page of XAMPP**

Whenever you want, type in the address bar of your browser:

<http://localhost/xampp/>

**Step 10 To run the php file**

<http://localhost/xampp/demo.php>

***How to start XAMPP GUI?***

If you use 32 system:

```
sudo /opt/lampp/manager-linux.run
```

If you use 64 system:

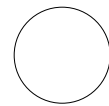
```
sudo /opt/lampp/manager-linux-x64.run
```

**Exercises:**

1. Install XAMPP or WAMP in your personal computer and take the screen shots of each and every step. Also write down the installation steps.

**Review Questions:**

1. What is Wamp and Xamp Server?
2. What is the difference between XAMPP or WAMP Server?



## Basics of HTML

**Practical No:**\_\_\_\_\_

**Date:**\_\_\_\_\_

**Aim:** Study of Formatting and fonts, commenting code, color, hyperlink, lists, tables, images.)

### Theory

#### *What is HTML?*

HTML is a markup language for describing web documents (web pages).

- HTML stands for Hyper Text Markup Language
- A markup language is a set of markup tags
- HTML documents are described by HTML tags
- Each HTML tag describes different document content

#### *What can I use HTML for?*

If you want to make websites, there is no way around HTML. Even if you're using a program to create websites, such as Dreamweaver, a basic knowledge of HTML can make life a lot simpler and your website a lot better. The good news is that HTML is easy to learn and use. In just two lessons from now you will have learned how to make your first website.

HTML is used to make websites. It is as simple as that!

#### *But what does H-T-M-L stand for?[2]*

HTML is an abbreviation of "HyperText Mark-up Language" - which is already more than you need to know at this stage. However, for the sake of good order, let us explain in greater detail.

- **Hyper** is the opposite of linear. In the good old days - when a mouse was something the cat chased - computer programs ran linearly: when the program had executed one action it went to the next line and after that, the next line and so on. But HTML is different - you can go wherever you want and whenever you want. For example, it is not necessary to visit MSN.com before you visit HTML.net.
- **Text** is self-explanatory.
- **Mark-up** is what you do with the text. You are marking up the text the same way you do in a text editing program with headings, bullets and bold text and so on.
- **Language** is what HTML is. It uses many English words.

#### *History of HTML*

Since its initial introduction in late 1991, HTML (and later its XML-based cousin, XHTML) has undergone many changes. Interestingly, the first versions of HTML used to build the earliest Web pages lacked a rigorous definition. Fortunately, by 1993 the Internet Engineering Task Force (IETF) began to standardize the language and later, in 1995, released the first real HTML standard in the form of HTML 2.0. You will likely encounter more than just the latest style of markup for many years to come, so Table 1.1 presents a brief summary of the version history of HTML and XHTML.[1]

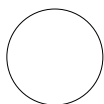


Table 1: History of HTML

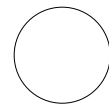
Sr.No.	HTML Version	Description
1	HTML 2.0	Classic HTML dialect supported by browsers such as Mosaic. This form of HTML supports core HTML elements and features such as tables and forms, but does not consider any of the browser innovations of advanced features such as style sheets, scripting, or frames.
2	HTML 3.0	The proposed replacement for HTML 2.0 that was never widely adopted, most likely due to the heavy use of browser-specific markup.
3	HTML 3.2	An HTML finalized by the W3C in early 1997 that standardized most of the HTML features introduced in browsers such as Netscape 3. This version of HTML supports many presentation-focused elements such as font, as well as early support for some scripting features.
4	HTML 4.0 Transitional	The 4.0 transitional form finalized by the W3C in December of 1997 preserves most of the presentational elements of HTML 3.2. It provides a basis of transition to Cascading Style Sheets (CSS) as well as a base set of elements and attributes for multiple-language support, accessibility, and scripting.
5	HTML 4.0 Strict	The strict version of HTML 4.0 removes most of the presentation elements from the HTML specification, such as font, in favor of using CSS for page formatting.
6	4.0 Frameset	The frameset specification provides a rigorous syntax for framed documents that was lacking in previous versions of HTML.
7	HTML 4.01 Transitional/Strict/Frameset	A minor update to the 4.0 standard that corrects some of the errors in the original specification.
8	HTML5	Addressing the lack of acceptance of the XML reformulation of HTML by the mass of Web page authors, the emerging HTML5 standard originally started by the WHATWG3 group and later rolled into a W3C effort aimed to rekindle the acceptance of traditional HTML and extend it to address Web application development, multimedia, and the ambiguities found in browser parsers. Since 2005, features now part of this HTML specification have begun to appear in Web browsers, muddying the future of XHTML in Web browsers.

You can read detailed history of HTML from link:

<http://www.w3.org/People/Raggett/book4/ch02.html>

### ***Creating Web Pages with HTML***

In today's Web, people shop, trade stocks, watch videos, share photos, play games, communicate via social networks, interact using live chat and video, and much more. They perform all of these activities using Web pages, text documents that Web browsers interpret and display. Despite the diversity of content, all of the Web pages on the World Wide Web have one thing in common. They all must be created using some form of the Hypertext Markup Language (HTML). It is astonishing to think that the entire World Wide Web, used every day by millions for shopping, research, banking, and a myriad of other applications is based on a simple text-based markup language that is easy to learn and use.



**HTML is a markup language, a structured language that lets you identify common sections of a Web page such as headings, paragraphs, and lists with markup tags that define each section.**

### ***What is Element and Tag?***

Elements give structure to a HTML document and tells the browser how you want your website to be presented. Generally elements consists of a start tag, some content, and an end tag.

"Tags"?

Tags are labels you use to mark up the beginning and end of an element.

All tags have the same format: they begin with a less-than sign "<" and end with a greater-than sign ">".

HTML tags are keywords (tag names) surrounded by angle brackets:

`<tagname>content</tagname>`

Generally speaking, there are two kinds of tags - opening tags: `<html>` and closing tags: `</html>`. The only difference between an opening tag and a closing tag is the forward slash "/". You label content by putting it between an opening tag and a closing tag.

HTML is all about elements. To learn HTML is to learn and use different tags.

### **Example:**

The element `em` emphasis text. All text between the opening tag `<em>` and the closing tag `</em>` is emphasised in the browser. ("em" is short for "emphasis".)

`<em>Emphasised text.</em>`

`<h1>This is a heading</h1>`

`<h2>This is a subheading</h2>`

### ***Always need an opening tag and a closing tag?***

As they say, there's an exception to every rule and in HTML the exception is that there are a few elements which both open and close in the same tag. These so-called empty elements are not connected to a specific passage in the text but rather are isolated labels, for example, a line break which looks like this: `<br />`.

### ***Should tags be typed in uppercase or lowercase?***

Most browsers might not care if you type your tags in upper, lower or mixed cases. `<HTML>`, `<html>` or `<HtMl>` will normally give the same result. However, the correct way is to type tags in lowercase. So get into the habit of writing your tags in lowercase.

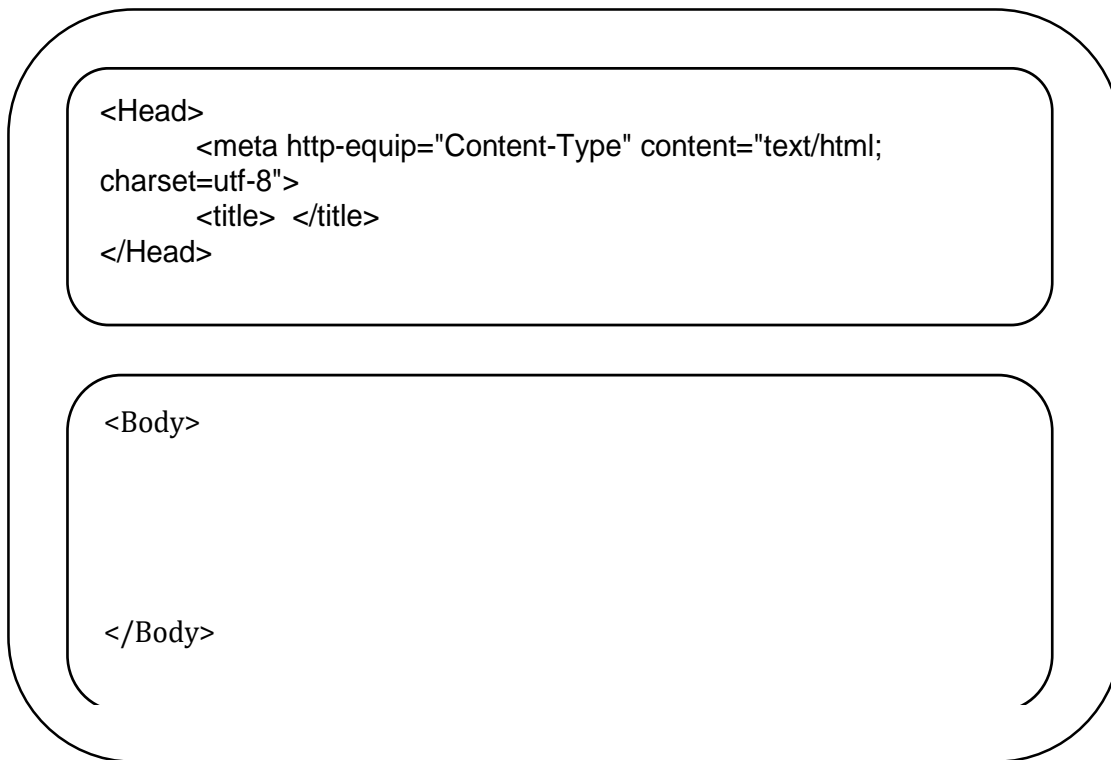
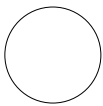


Fig. 1: HTML Document Structure

### **<!DOCTYPE> - Document Type Statements and Language Versions**

HTML documents should begin with a <!DOCTYPE> declaration. This statement identifies the type of markup that is supposedly used in a document. For example,

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Indicates that we are using the transitional variation of HTML 4.01 that starts with a root element html. In other words, an <html> tag will serve as the ultimate parent of all the content and elements within this document.

A <!DOCTYPE> declaration might get a bit more specific and specify the URI (Uniform Resource Identifier) of the DTD being used as shown here:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

In the case of an XHTML document, the situation really isn't much different:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

However, do note that the root html element here is lowercase, which hints at the case sensitivity found in XHTML.

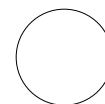


Table 2: Common HTML Doctype Declarations

HTML Version	!DOCTYPE Declaration
HTML 2.0	<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
HTML 3.2	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
HTML 4.0 Transitional	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
HTML 4.0 Frameset	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
HTML 4.0 Strict	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/html4/strict.dtd">
HTML 4.01 Transitional	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
HTML 4.01 Frameset	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">
HTML 4.01 Strict	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
HTML5	<!DOCTYPE html>
XHTML 2.0	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 2.0//EN" "http://www.w3.org/MarkUp/DTD/xhtml2.dtd">
XHTML Basic 1.1	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">

### <HTML> - Starting an HTML Page

Starts the HTML document and contains that document surrounding everything else except the <!DOCTYPE> tag.

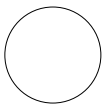
This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.

### <HEAD> - Creating a Web Page's Head

The information in the head element of an HTML document is very important because it is used to describe or augment the content of the document. The element acts like the front matter or cover page of a document. In many cases, the information contained within the head element is information about the page that is useful for visual styling, defining interactivity, setting the page title, and providing other useful information that describes or controls the document.

Head section include following tag also.

- **Base** - specifies a base URL for all relative URLs contained in the document
- **Link** - defines the relationship between the current document and other documents or resources
- **Meta** - provides general information about a document for indexing and other purposes
- **Script** - contains (or refers to) statements in a scripting language that are to be processed on the client side
- **Style** - contains CSS style information that's embedded into a page
- **Title** - defines the title of the web page or document



The complete syntax of the markup allowed in the head element under strict HTML is shown here:

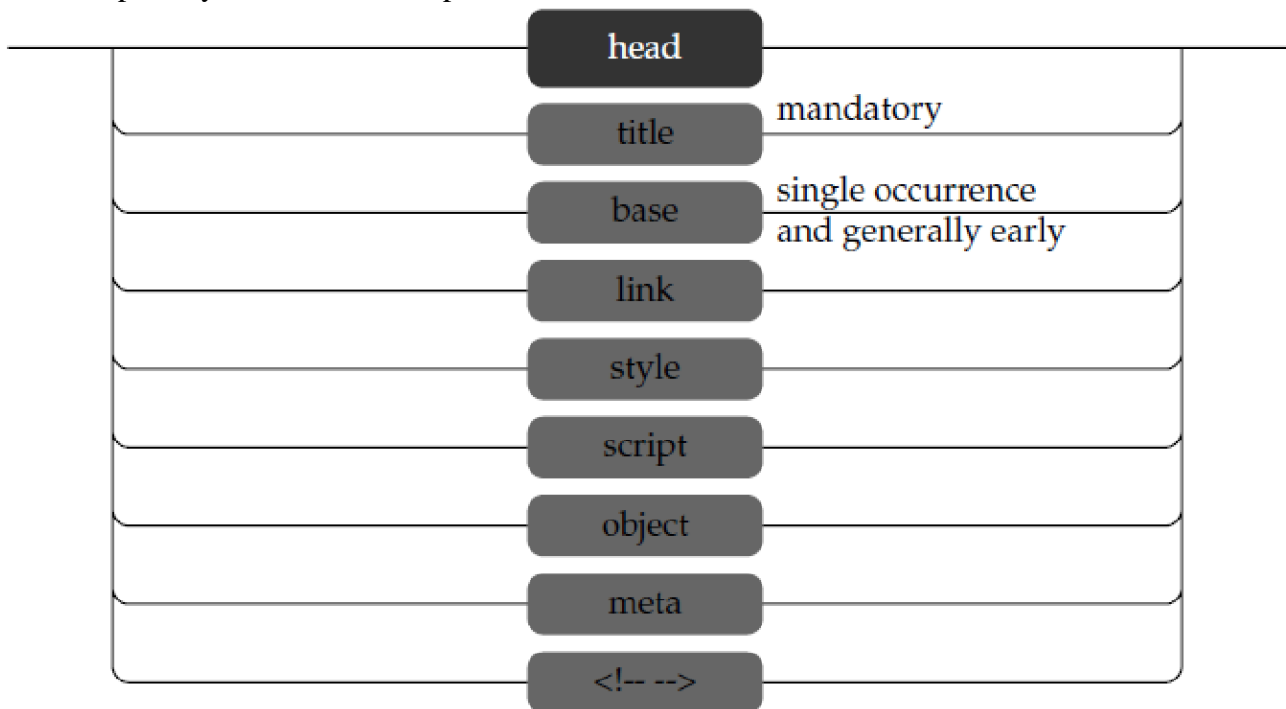


Fig. 2: List of Elements under <HEAD> tag

### <TITLE> - Giving a Web Page A Title [1]

A single title element is required in the head element and is used to set the text that most browsers display in their title bar. The value within a title is also used in a browser's history system, recorded when the page is bookmarked, and consulted by search engine robots to help determine page meaning. In short, it is pretty important to have a syntactically correct, descriptive, and appropriate page title. Thus, given

```
<title>Simple HTML Title Example</title>
```

When a title is not specified, most browsers display the URL path or filename instead

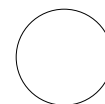
### <META> - Giving more information about your web page [1]

<meta>: Specifying Content Type, Character Set, and More.

A <meta> tag has a number of uses. For example, it can be used to specify values that are equivalent to HTTP response headers. For example, if you want to make sure that your MIME type and character set for an English-based HTML document is set, you could use

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
Because meta is an empty element, you would use the trailing-slash syntax shown  
here:  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
```





### **<BASE>**

A `<base>` tag specifies an absolute URL address that is used to provide server and directory information for partially specified URL addresses, called relative links, used within the document:

```
<base href="http://htmlref.com/basexample" >
```

Because of its global nature, a `<base>` tag is often found right after a `<title>` tag as it may affect subsequent `<script>`, `<link>`, `<style>`, and `<object>` tag referenced URIs.

### **<LINK>**

A `<link>` tag specifies a special relationship between the current document and another document. Most commonly, it is used to specify a style sheet used by the document

```
<link rel="stylesheet" media="screen" href="global.css" type="text/css" >
```

However, the `<link>` tag has a number of other interesting possible uses, such as to set up navigation relationships and to hint to browsers about pre-cacheable content. See the element reference in Chapter 3 for more information on this.

### **<OBJECT>**

An `<object>` tag allows programs and other binary objects to be directly embedded in a Web page. Here, for example, a nonvisible Flash object is being referenced for some use:

```
<object   classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"   width="0"
height="0" id="HiddenFlash" >
    <param name="movie" value="flashlib.swf" />
</object>
```

Using an `<object>` tag involves more than a bit of complexity, and there are numerous choices of technology, including Java applets, plug-ins, and ActiveX controls.

### **<SCRIPT>**

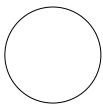
A `<script>` tag allows scripting language code to be either directly embedded within,

```
<script type="text/javascript">
    alert("Hi from JavaScript!");
    /* more code below */
</script>
```

or, more appropriately, linked to from a Web page:

```
<script type="text/javascript" href="ajaxter.js"></script>
```

Nearly always, JavaScript is the language in use, though other languages such as VBScript are possible.



## <STYLE>

A <style> tag is used to enclose document-wide style specifications, typically in Cascading Style Sheet (CSS) format, relating to fonts, colors, positioning, and other aspects of content presentation:

```
<style type="text/css" media="screen">
    h1 {font-size: xx-large; color: red; font-style: italic;}
    /* all h1 elements render as big, red and italic */
</style>
```

## COMMENTS

Finally, comments are often found in the head of a document. Following SGML syntax, a comment starts with <!-- and ends with --> and may encompass many lines:

```
<!-- Hi I am a comment -->
<!-- Author: Thomas A. Powell
      Book: HTML: The Complete Reference
      Edition: 5
-->
```

Comments can contain just about anything except other comments and are particularly sensitive to – symbols. Thus

```
<!------- THIS ISN'T A SYNTACTICALLY CORRECT COMMENT! ---->
```

## <BODY> - *Creating a Web Page's Body*

After the head section, the body of a document is delimited by <body> and </body>. Under the HTML 4.01 specification and many browsers, the body element is optional, but you should always include it, particularly because it is required in stricter markup variants.

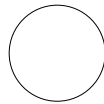
Only one body element can appear per document.

Within the body of a Web document is a variety of types of elements. For example, blocklevel elements define structural content blocks such as paragraphs (p) or headings (h1-h6).

Block-level elements generally introduce line breaks visually. Special forms of blocks, such as unordered lists (ul), can be used to create lists of information.

Within nonempty blocks, inline elements are found. There are numerous inline elements, such as bold (b), italic (i), strong (strong), emphasis (em), and numerous others. These types of elements do not introduce any returns.

Other miscellaneous types of elements, including those that reference other objects such as images (img) or interactive elements (object), are also generally found within blocks, though in some versions of HTML they can stand on their own. [1]



### *Example: Sample HTML page [1]*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Sample Head Element</title>

<!-- Some example meta tags -->
<meta name="keywords" content="Fake, Head Example, HTML Ref" />
<meta name="description" content="A simple head example that shows a
number of the elements presented in action." />
<meta name="author" content="Thomas A. Powell" />

<!-- Set a global URI stem for all references -->
<base href="http://htmlref.com/baseexample" />

<!-- Linked and document specific styles -->
<link rel="stylesheet" href="screen.css" media="screen" />
<link rel="stylesheet" href="printer.css" media="print" />

<style type="text/css">
    <!-- -
        h1 {font-size: xx-large; color: red; font-style: italic;}
    -->
</style>

<!-- Embedded and linked scripts -->

<script type="text/javascript">
<!--
        var globalDebug = true;
//-->
</script>

<script src="ajaxtr.js" type="text/javascript"></script>
<script src="effects.js" type="text/javascript"></script>

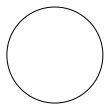
</head>

<body>
        <p>Some body content here.</p>
</body>

</html>
```

### *The Rules of HTML*

- **HTML Is Not Case Sensitive**
  - <B>Go boldly</B>



- `<B>Go boldly</b>`
- `<b>Go boldly</B>`
- `<b>Go boldly</b>`
- **Attribute Values May Be Case Sensitive**
  - HTML Is Sensitive to a Single Whitespace Character
  - Any white space between characters displays as a single space. This includes all tabs, line
  - breaks, and carriage returns. Consider this markup:
  - `<strong>T e s t o f s p a c e s</strong><br>`
  - `<strong>T e s t o f s p a c e s</strong><br>`
  - `<strong>T`
  - `e s`
  - `t o f s p a c e s</strong><br>`
- **Unused Elements May Minimize**
- **Attributes Should Be Quoted**

### HTML Basic Formatting Tags

HTML also defines special elements, for defining text with a special meaning.

HTML uses elements like `<b>` and `<i>` for formatting output, like bold or italic text.

Formatting elements were designed to display special types of text:

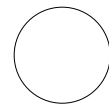
- Bold text
- Important text
- Italic text
- Emphasized text
- Marked text
- Small text
- Deleted text
- Inserted text
- Subscripts
- Superscripts

### **Heading Tags**

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, and `<h6>`. While displaying any heading, browser adds one line before and one line after that heading.

#### **Example:**

```
<!DOCTYPE html>
<html><head>
  <title>Heading Example</title>
</head><body>
  <h1>This is heading 1</h1>
  <h2>This is heading 2</h2>
  <h3>This is heading 3</h3>
  <h4>This is heading 4</h4>
  <h5>This is heading 5</h5>
  <h6>This is heading 6</h6>
</body></html>
```



## Output:

**This is heading 1**

**This is heading 2**

**This is heading 3**

**This is heading 4**

**This is heading 5**

**This is heading 6**

## Paragraph Tag

The `<p>` tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening `<p>` and a closing `</p>` tag as shown below in the example:

```
<p>Here is a first paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p>Here is a third paragraph of text.</p>
```

## Line Break Tag

Whenever you use the `<br />` element, anything following it starts from the next line. This tag is an example of an empty element, where you do not need opening and closing tags, as there is nothing to go in between them.

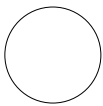
The `<br />` tag has a space between the characters `br` and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use `<br>` it is not valid in XHTML

```
<body>
    <p>Hello<br />
    You delivered your assignment ontime.<br />
    Thanks<br />
    Mahnaz</p>
</body>
```

## Centering Content

You can use `<center>` tag to put any content in the center of the page or any table cell.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Centring Content Example</title>
    </head>
    <body>
        <p>This text is not in the center.</p>
        <center>
            <p>This text is in the center.</p>
        </center>
    </body>
</html>
```



### **Horizontal Lines**

Horizontal lines are used to visually break up sections of a document. The `<hr>` tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Horizontal Line Example</title>
  </head>
  <body>
    <p>This is paragraph one and should be on top</p>
    <hr />
    <p>This is paragraph two and should be at bottom</p>
  </body>
</html>
```

Again `<hr />` tag is an example of the empty element, where you do not need opening and closing tags, as there is nothing to go in between them.

The `<hr />` element has a space between the characters `hr` and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use `<hr>` it is not valid in XHTML.

### **Preserve Formatting**

Sometimes you want your text to follow the exact format of how it is written in the HTML document. In those cases, you can use the preformatted tag `<pre>`.

Any text between the opening `<pre>` tag and the closing `</pre>` tag will preserve the formatting of the source document.

#### **Example:**

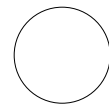
```
<!DOCTYPE html>
<html>
  <head>
    <title>Preserve Formatting Example</title>
  </head>
  <body>
    <pre>
      function testFunction( strText )
      {
        alert (strText)
      }
    </pre>
  </body>
```

### **Nonbreaking Spaces**

Suppose you want to use the phrase "12 Angry Men." Here you would not want a browser to split the "12, Angry" and "Men" across two lines:

An example of this technique appears in the movie "12 Angry Men."

In cases where you do not want the client browser to break text, you should use a nonbreaking space entity `&nbsp;` instead of a normal space. For example, when coding the "12 Angry Men" in a paragraph, you should use something similar to the following code:



### Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Nonbreaking Spaces Example</title>
  </head>
  <body>
    <p>An example of this technique appears in the movie
    "12&nbsp;Angry&nbsp;Men."</p>
  </body>
</html>
```

### Output:

An example of this technique appears in the movie "12 Angry Men."

### *Italic and Underlining*

Use the sub tag <i></i> for italic and <u></u> for Underlining.

### Example:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Nested Elements Example</title>
  </head>
  <body>
    <h1>This is <i>italic</i> heading</h1>
    <p>This is <u>underlined</u> paragraph</p>
  </body>
</html>
```

### Output:

This is *italic* heading

This is underlined paragraph

### **HTML Small Formatting [3]**

The HTML <small> element defines small text:

Example

```
<h2>HTML <small>Small</small> Formatting</h2>
```

### **HTML Marked Formatting [3]**

The HTML <mark> element defines marked or highlighted text:

Example

```
<h2>HTML <mark>Marked</mark> Formatting</h2>
```

### **HTML Deleted Formatting [3]**

The HTML <del> element defines deleted (removed) text.

Example

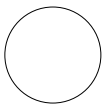
```
<p>My favorite color is <del>blue</del> red.</p>
```

### **HTML Subscript Formatting [3]**

The HTML <sub> element defines subscripted text.

Example

```
<p>This is <sub>subscripted</sub> text.</p>
```



### HTML Superscript Formatting [3]

The HTML <sup> element defines superscripted text.

Example

<p>This is <sup>superscripted</sup> text.</p>

Table 3: HTML Quotation and Citation Elements

Tag	Description
<abbr>	Defines an abbreviation or acronym
<address>	Defines contact information for the author/owner of a document
<bdo>	Defines the text direction
<blockquote>	Defines a section that is quoted from another source
<cite>	Defines the title of a work
<q>	Defines a short inline quotation

### Perform following examples:

- <p>WWF's goal is to: <q>Build a future where people live in harmony with nature.</q></p>
- <p>Here is a quote from WWF's website:</p>
  - <blockquote cite="http://www.worldwildlife.org/who/index.html">
  - For 50 years, WWF has been protecting the future of nature.
  - The world's leading conservation organization,
  - WWF works in 100 countries and is supported by
  - 1.2 million members in the United States and
  - close to 5 million globally.
  - </blockquote>
- <p>The <abbr title="World Health Organization">WHO</abbr> was founded in 1948.</p>
- Written by Jon Doe.<br>
  - Visit us at:<br>
  - Example.com<br>
  - Box 564, Disneyland<br>
  - USA
  - </address>
- <p><cite>The Scream</cite> by Edward Munch. Painted in 1893.</p>
- <bdo dir="rtl">This text will be written from right to left</bdo>

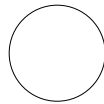
### HTML Fonts <Font> Tag

Font face and color depends entirely on the computer and browser that is being used to view your page but you can use HTML <font> tag to add style, size, and color to the text on your website. You can use a <basefont> tag to set all of your text to the same size, face, and color.

The font tag is having three attributes called size, color, and face to customize your fonts. To change any of the font attributes at any time within your webpage, simply use the <font> tag. The text that follows will remain changed until you close with the </font> tag. You can change one or all of the font attributes within one <font> tag.

You can set content font size using size attribute. The range of accepted values is from 1(smallest) to 7(largest). The default size of a font is 3.





**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Setting Font Size</title>
  </head>
  <body>
    <font size="1">Font size="1"</font><br />
    <font size="2">Font size="2"</font><br />
    <font size="3">Font size="3"</font><br />
    <font size="4">Font size="4"</font><br />
    <font size="5">Font size="5"</font><br />
    <font size="6">Font size="6"</font><br />
    <font size="7">Font size="7"</font>
  </body>
</html>
```

**Output:**

Font size="1"  
Font size="2"  
Font size="3"  
Font size="4"  
Font size="5"  
Font size="6"  
Font size="7"

***Relative Font Size***

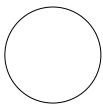
You can specify how many sizes larger or how many sizes smaller than the preset font size should be. You can specify it like <font size="+n"> or <font size="-n">

**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Relative Font Size</title>
  </head>
  <body>
    <font size="-1">Font size="-1"</font><br />
    <font size="+1">Font size="+1"</font><br />
    <font size="+2">Font size="+2"</font><br />
    <font size="+3">Font size="+3"</font><br />
    <font size="+4">Font size="+4"</font>
  </body>
</html>
```

**Output:**

Font size="-1"  
Font size="+1"  
Font size="+2"  
Font size="+3"  
Font size="+4"



### ***Setting Font Face***

You can set font face using face attribute but be aware that if the user viewing the page doesn't have the font installed, they will not be able to see it. Instead user will see the default font face applicable to the user's computer.

**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Font Face</title>
  </head>
  <body>
    <font face="Times New Roman" size="5">Times New Roman</font><br />
    <font face="Verdana" size="5">Verdana</font><br />
    <font face="Comic sans MS" size="5">Comic Sans MS</font><br />
    <font face="WildWest" size="5">WildWest</font><br />
    <font face="Bedrock" size="5">Bedrock</font><br />
  </body>
</html>
```

**Output:**

Times New Roman

Verdana

Comic Sans MS

WildWest

Bedrock

### ***Specify alternate font faces***

A visitor will only be able to see your font if they have that font installed on their computer. So, it is possible to specify two or more font face alternatives by listing the font face names, separated by a comma.

**Example:**

```
<font face="arial,helvetica">
```

```
<font face="Lucida Calligraphy,Comic Sans MS,Lucida Console">
```

### ***Setting Font Color***

You can set any font color you like using color attribute. You can specify the color that you want by either the color name or hexadecimal code for that color.

Note: You can check a complete list of HTML Color Name with Codes.

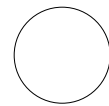
**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Setting Font Color</title>
  </head>
  <body>
    <font color="#FF00FF">This text is in pink</font><br />
    <font color="red">This text is red</font>
  </body></html>
```

**Output:**

This text is in pink

This text is red



### ***The <basefont> Element***

The <basefont> element is supposed to set a default font size, color, and typeface for any parts of the document that are not otherwise contained within a <font> tag. You can use the <font> elements to override the <basefont> settings.

The <basefont> tag also takes color, size and face attributes and it will support relative font setting by giving size a value of +1 for a size larger or -2 for two sizes smaller.

#### **Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>Setting Basefont Color</title>
  </head>
  <body>
    <basefont face="arial, verdana, sans-serif" size="2" color="#ff0000">
    <p>This is the page's default font.</p>
    <h2>Example of the &lt;basefont&gt; Element</h2>
    <p><font size="+2" color="darkgray">
    This is darkgray text with two sizes larger
    </font></p>
    <p><font face="courier" size="-1" color="#000000">
    It is a courier font, a size smaller and black in color.
    </font></p>
  </body>
</html>
```

#### **Output:**

This is the page's default font.

#### **Example of the <basefont> Element**

This is darkgray text with two sizes larger

It is a courier font, a size smaller and black in color.

### **HTML Colors**

You can specify colors on page level using <body> tag or you can set colors for individual tags using bgcolor attribute.

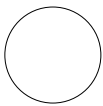
The <body> tag has following attributes which can be used to set different colors:

- bgcolor - sets a color for the background of the page.
- text - sets a color for the body text.
- alink - sets a color for active links or selected links.
- link - sets a color for linked text.
- vlink - sets a color for visited links - that is, for linked text that you have already clicked on.

#### **HTML Color Coding Methods**

















There are following three different methods to set colors in your web page:

1. Color names - You can specify color names directly like green, blue or red.
2. Hex codes - A six-digit code representing the amount of red, green, and blue that makes up the color.
3. Color decimal or percentage values - This value is specified using the rgb( ) property.



### W3C Standard 16 Colors

Table 4: W3C Standard 16 Colors

	Black		Gray		Silver		White
	Yellow		Lime		Aqua		Fuchsia
	Red		Green		Blue		Purple
	Maroon		Olive		Navy		Teal

#### Example:

Here are the examples to set background of an HTML tag by color name:










```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Colors by Name</title>
  </head>
  <body text="blue" bgcolor="green">
    <p>Use different color names for for body and table and see the
    result.</p>
    <table bgcolor="black">
      <tr>
        <td>
          <font color="white">This text will appear white on black
          background.</font>
        </td>
      </tr>
    </table>
  </body></html>
```

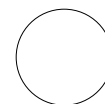
### HTML Colors - Hex Codes

A hexadecimal is a 6 digit representation of a color. The first two digits(RR) represent a red value, the next two are a green value(GG), and the last are the blue value(BB).

A hexadecimal value can be taken from any graphics software like Adobe Photoshop, Paintshop Pro or MS Paint.

Table 5: Colors - Hex Codes

Color	Color HEX
	#000000
	#FF0000
	#00FF00
	#0000FF
	#FFFF00
	#00FFFF
	#FF00FF
	#C0C0C0
	#FFFFFF

**Example:**

Here are the examples to set background of an HTML tag by color code in hexadecimal:










```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Colors by Hex</title>
  </head>
  <body text="#0000FF" bgcolor="#00FF00">
    <p>Use different color hexa for for body and table and see the
    result.</p>
    <table bgcolor="#000000">
      <tr>
        <td>
          <font color="#FFFFFF">This text will appear white on black
          background.</font>
        </td>
      </tr>
    </table>
  </body>
</html>
```

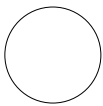
**HTML Colors - RGB Values**

This color value is specified using the `rgb( )` property. This property takes three values, one each for red, green, and blue. The value can be an integer between 0 and 255 or a percentage.

Note: All the browsers does not support `rgb()` property of color so it is recommended not to use it.

Table 6: Colors - RGB Values

Color	Color RGB
	<code>rgb(0,0,0)</code>
	<code>rgb(255,0,0)</code>
	<code>rgb(0,255,0)</code>
	<code>rgb(0,0,255)</code>
	<code>rgb(255,255,0)</code>
	<code>rgb(0,255,255)</code>
	<code>rgb(255,0,255)</code>
	<code>rgb(192,192,192)</code>
	<code>rgb(255,255,255)</code>

**Example:**

Here are the examples to set background of an HTML tag by color code using rgb() values:

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Colors by RGB code</title>
  </head>
  <body text="rgb(0,0,255)" bgcolor="rgb(0,255,0)">
    <p>Use different color code for for body and table and see the
    result.</p>
    <table bgcolor="rgb(0,0,0)">
      <tr>
        <td>
          <font color="rgb(255,255,255)">This text will appear white on
          black background.</font>
        </td>
      </tr>
    </table>
  </body>
</html>
```

**Linking Documents**

A link is specified using HTML tag <a>. This tag is called anchor tag and anything between the opening <a> tag and the closing </a> tag becomes part of the link and a user can click that part to reach to the linked document. Following is the simple syntax to use <a> tag.

`<a href="Document URL" ... attributes-list>Link Text</a>`

**Example:**

Let's try following example which links <http://www.google.com> at your page:

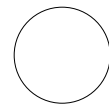
```
<!DOCTYPE html>
<html>
  <head>
    <title>Hyperlink Example</title>
  </head>
  <body>
    <p>Click following link</p>
    <a href="http://www.google.com" target="_self">google search
    engine</a>
  </body>
</html>
```

**The target Attribute**

We have used target attribute in our previous example. This attribute is used to specify the location where linked document is opened. Following are possible options:

Table 7: Target attribute options

Option	Description
_blank	Opens the linked document in a new window or tab.
_self	Opens the linked document in the same frame.
_parent	Opens the linked document in the parent frame.
_top	Opens the linked document in the full body of the window.
targetframe	Opens the linked document in a named <i>targetframe</i> .



### Example:

```
Example:
<!DOCTYPE html>
<html>
  <head>
    <title>Hyperlink Example</title>
    <base href="http://www.google.com/">
  </head>
  <body>
    <p>Click any of the following links</p>
    <a href="/html/index.htm" target="_blank">Opens in New</a> |
    <a href="/html/index.htm" target="_self">Opens in Self</a> |
    <a href="/html/index.htm" target="_parent">Opens in Parent</a> |
    <a href="/html/index.htm" target="_top">Opens in Body</a>
  </body>
</html>
```

### Use of Base Path

When you link HTML documents related to the same website, it is not required to give a complete URL for every link. You can get rid of it if you use <base> tag in your HTML document header. This tag is used to give a base path for all the links. So your browser will concatenate given relative path to this base path and will make a complete URL.

### Example:

Following example makes use of <base> tag to specify base URL and later we can use relative path to all the links instead of giving complete URL for every link.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hyperlink Example</title>
    <base href="http://www.google.com/">
  </head>
  <body>
    <p>Click following link</p>
    <a href="/html/index.htm" target="_blank">HTML Tutorial</a>
  </body>
</html>
```

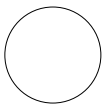
Now given URL <a href="/html/index.htm" is being considered as <a href="http://www.google.com/html/index.htm".

### Setting Link Colors

You can set colors of your links, active links and visited links using link, alink and vlink attributes of <body> tag.

### Example:

Save the following in test.htm and open it in any web browser to see how link, alink and vlink attributes work.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Hyperlink Example</title>
    <base href="http://www.tutorialspoint.com/">
  </head>
  <body alink="#54A250" link="#040404" vlink="#F40633">
    <p>Click following link</p>
    <a href="/html/index.htm" target="_blank" >HTML Tutorial</a>
  </body>
</html>
```

### HTML Lists [3]

#### **Unordered HTML Lists**

An unordered list starts with the <ul> tag. Each list item starts with the <li> tag. The list items will be marked with bullets (small black circles):

#### **Example:**

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

#### **Unordered HTML Lists**

Table 8: Style Attribute options

Style	Description
list-style-type:disc	The list items will be marked with bullets (default)
list-style-type:circle	The list items will be marked with circles
list-style-type:square	The list items will be marked with squares
list-style-type:none	The list items will not be marked

#### **Example:**

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

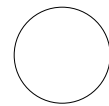
#### **Ordered HTML Lists**

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

#### **Example:**

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



**Output:**

1. Coffee
2. Tea
3. Milk

**Ordered HTML Lists**

Table 9: Type Attribute options

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

**Example:**

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

**Output:**

- A. Coffee
- B. Tea
- C. Milk

**HTML IMG Tag****IMG tag Includes**

Use the HTML `<img>` element to define an image

Use the HTML `src` attribute to define the URL of the image

Use the HTML `alt` attribute to define an alternate text for an image, if it cannot be displayed

Use the HTML `width` and `height` attributes to define the size of the image

Use the CSS `width` and `height` properties to define the size of the image (alternatively)

***Insert Image [1]***

You can insert any image in your web page by using `<img>` tag. Following is the simple syntax to use this tag.

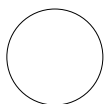
```

```

The `<img>` tag is an empty tag, which means that it can contain only list of attributes and it has no closing tag.

**Example:**

```
<!DOCTYPE html>
<html><head>
  <title>Using Image in Webpage</title>
</head>
<body><p>Simple Image Insert</p>
  
</body></html>
```



The alt attribute is a mandatory attribute which specifies an alternate text for an image, if the image cannot be displayed.

Table 10: IMG tag Attribute list

Attribute	Description	Possible values
alt	Alternative text that will replace the image if the image isn't available. This <b>must be included</b> , if possible. In circumstances where it isn't, the img must be the only content of a figure element in addition to a figcaption that contains a caption for the image.	Text.
width	The width of the image in pixels. This can also be done, or overridden, with CSS.	Number.
height	The height of the image in pixels. This can also be done, or overridden, with CSS.	Number.
ismap	Indicates that the element is used as a server-side image map.	None.
usemap	Image map name.	Hash name (eg. #flyer) matching the name of a map element (eg. <map name="flyer">...).
crossorigin	Used in conjunction with JavaScript to determine how Cross Origin Resource Sharing requests are handled.	anonymous use-credentials

### Images in another Folder [1]

If not specified, the browser expects to find the image in the same folder as the web page. However, it is common to store images in a sub-folder. You must then include the folder name in the src attribute:

```
<img src= "/images/html5.gif" alt= "HTML5 Icon" style= "width:128px; height:128px;">
```

### Images on another Server [1]

Some web sites store their images on image servers. Actually, you can access images from any web address in the world:

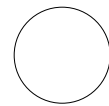
```
<img src= "http://www.google.com/images/green.jpg" alt= "W3Schools.com">
```

### Set Image Width/Height [2]

You can set image width and height based on your requirement using width and height attributes. You can specify width and height of the image in terms of either pixels or percentage of its actual size.

### Example:

```
<!DOCTYPE html>
<html><head>
    <title>Set Image Width and Height</title>
</head>
<body><p>Setting image width and height</p>
    <img src= "/html/images/test.png" alt= "Test Image" width="150"
    height="100">
</body></html>
```



### ***Set Image Border [2]***

By default image will have a border around it, you can specify border thickness in terms of pixels using border attribute. A thickness of 0 means, no border around the picture.

`<!DOCTYPE html>`

```
<html>
  <head>
    <title>Set Image Border</title>
  </head>
  <body>
    <p>Setting image Border</p>
    
  </body>
</html>
```

### ***Set Image Alignment [1]***

By default image will align at the left side of the page, but you can use align attribute to set it in the center or right.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Set Image Alignment</title>
  </head>
  <body>
    <p>Setting image Alignment</p>
    
  </body>
</html>
```

### ***Using an Image as a Link [1]***

To use an image as a link, simply nest the <img> tag inside the <a> tag:

Example:

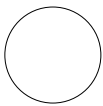
```
<a href="default.asp">
  
</a>
```

### **HTML TABLE Tag**

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the <table> tag in which the <tr> tag is used to create table rows and <td> tag is used to create data cells. [2]

- Tables are defined with the <table> tag.
- Tables are divided into table rows with the <tr> tag.
- Table rows are divided into table data with the <td> tag.
- A table row can also be divided into table headings with the <th> tag.

**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Tables</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <td>Row 1, Column 1</td>
        <td>Row 1, Column 2</td>
      </tr>
      <tr>
        <td>Row 2, Column 1</td>
        <td>Row 2, Column 2</td>
      </tr>
    </table>
  </body>
</html>
```

**Output:**

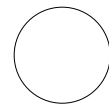
Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

**Table Heading**

Table heading can be defined using <th> tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use <th> element in any row.

**Example:**

```
<!DOCTYPE html>
<html><head>
  <title>HTML Table Header</title>
</head><body>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Salary</th>
    </tr>
    <tr>
      <td>Ramesh Raman</td>
      <td>5000</td>
    </tr>
    <tr>
      <td>Shabbir Hussein</td>
      <td>7000</td>
    </tr>
  </table></body></html>
```

**Output:**

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

***Cellpadding and Cellspacing Attributes***

There are two attributes called cellpadding and cellspacing which you will use to adjust the white space in your table cells. The cellspacing attribute defines the width of the border, while cellpadding represents the distance between cell borders and the content within a cell.

**Example:**

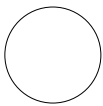
```
<!DOCTYPE html>
<html>
  <head>
    <title>HTML Table Cellpadding</title>
  </head>
  <body>
    <table border="1" cellpadding="5" cellspacing="5">
      <tr>
        <th>Name</th>
        <th>Salary</th>
      </tr>
      <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
      </tr>
      <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
      </tr>
    </table>
  </body>
</html>
```

**Output:**

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

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

### Output:

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

### Tables Backgrounds

You can set table background using one of the following two ways:

- bgcolor attribute - You can set background color for whole table or just for one cell.
- background attribute - You can set background image for whole table or just for one cell.

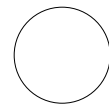
You can also set border color also using bordercolor attribute.

```
<table border="1" bordercolor="green" bgcolor="yellow">
</table>
```

```
<table border="1" bordercolor="green" background="/images/test.png">
</table>
```

### Table Height and Width

You can set a table width and height using width and height attributes. You can specify table width or height in terms of pixels or in terms of percentage of available screen area.



```
<table border="1" width="400" height="150">
</table>
```

### ***Table Caption***

The caption tag will serve as a title or explanation for the table and it shows up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

```
<table border="1" width="100%">
<caption>This is the caption</caption>
</table>
```

### ***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.

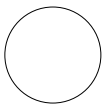
The three elements for separating the head, body, and foot of a table are:

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

A table may contain several <tbody> elements to indicate different pages or groups of data. But it is notable that <thead> and <tfoot> tags should appear before <tbody>

### **Example:**

```
<!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></html>
```

**Output:**

This is the head of the table			
Cell 1	Cell 2	Cell 3	Cell 4
This is the foot of the table			

**Exercises:**

## 1. Basic Exercise.

- Create a webpage that prints your name to the screen.
- Print the numbers 1 - 10, each number being a different color.
- Print some preformatted text of your choosing. (hint: use the <pre> tag)print like this.

i. class DataClass{

a. public static void main(String[] args){

i. System.out.println("Here is some text.");

b. }

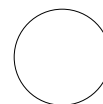
## 2. Hyperlink related

- Create links to five different pages on five different websites that should all open in a new window.
- Create a page with a link at the top of it that when clicked will jump all the way to the bottom of the page.

## 3. Make a Following Tables structure.

1		
2	3	4
5		6
7		



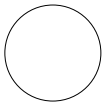


4. Make the Following color in the same format as shown.

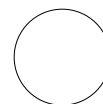
Branch	Information Technology (IT)	Computer Engineering (CE)
Total Seat	120	180
Course Duration	4 Year	4 Year
Key Subjects	<ul style="list-style-type: none"><li>• Languages<ul style="list-style-type: none"><li>◦ C, C++, JAVA</li></ul></li><li>• Data &amp; File Structures</li><li>• Database Management System</li><li>• Digital Logic &amp; Design</li><li>• Data Communication &amp; Networking</li><li>• Basic Electronics</li><li>• Computer Oriented Statistical Methods</li></ul>	<ul style="list-style-type: none"><li>• Languages<ul style="list-style-type: none"><li>◦ C, C++, JAVA, ASP.NET</li></ul></li><li>• Data &amp; File Structures</li><li>• Database Management System</li><li>• Fundamentals of Digital Electronics</li><li>• Data Communication Techniques</li><li>• Computer Oriented Statistical Methods</li></ul>

**Review Questions:**

1. What is HTML?
2. What is “Semantic HTML?”
3. What does DOCTYPE mean?
4. What are some new HTML5 markup elements?



**[This page is intentionally left blank]**



## Frame

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** Study about HTML Frame and Framesets.

### Theory

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

### *Disadvantages of Frames*

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages:

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's back button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

### *Creating Frames*

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

### **Example:**

Following is the example to create three horizontal frames:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
<frameset rows="10%,80%,10%">
  <frame name="top" src="/html/top_frame.htm" />
  <frame name="main" src="/html/main_frame.htm" />
  <frame name="bottom" src="/html/bottom_frame.htm" />
</frameset>
<body>
  Your browser does not support frames.
</body>
</html>
```

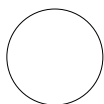


Fig. 1: Output result

Let's put above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically:

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Frames</title>
</head>
<frameset cols="25%,50%,25%">
  <frame name="left" src="/html/top_frame.htm" />
  <frame name="center" src="/html/main_frame.htm" />
  <frame name="right" src="/html/bottom_frame.htm" />
</frameset>
<body>
  Your browser does not support frames.
</body>
</frameset>
</html>
```

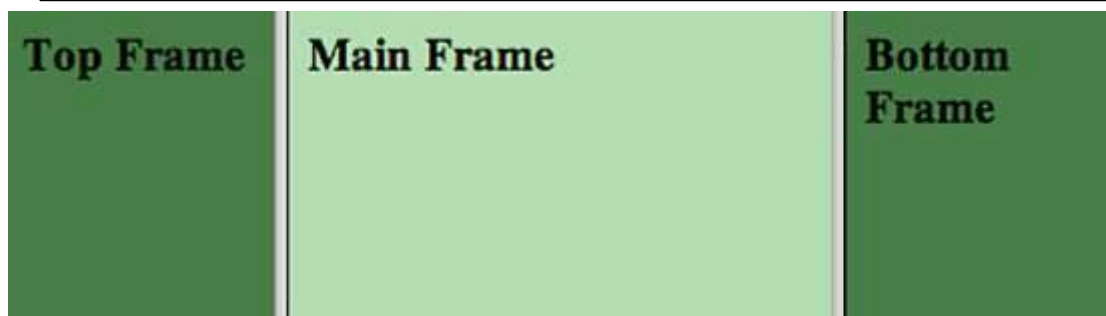
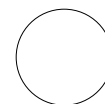


Fig. 2: Output result

### The <frameset> Tag Attributes

Table 1: Attributes of the <frameset> tag

Attribute	Description
Cols	<p>specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of four ways:</p> <ul style="list-style-type: none"><li>• Absolute values in pixels. For example to create three vertical frames, use <code>cols="100, 500, 100"</code>.</li><li>• A percentage of the browser window. For example to create three vertical frames, use <code>cols="10%, 80%, 10%"</code>.</li><li>• Using a wildcard symbol. For example to create three vertical frames, use <code>cols="10%, *, 10%"</code>. In this case wildcard takes remainder of the window.</li><li>• As relative widths of the browser window. For example to create three vertical frames, use <code>cols="3*, 2*, 1*"</code>. This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.</li></ul>



Rows	This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example to create two horizontal frames, use <code>rows="10%, 90%"</code> . You can specify the height of each row in the same way as explained above for columns.
Border	This attribute specifies the width of the border of each frame in pixels. For example <code>border="5"</code> . A value of zero means no border.
frameborder	This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example <code>frameborder="0"</code> specifies no border.
framespacing	This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example <code>framespacing="10"</code> means there should be 10 pixels spacing between each frames.

### *The <frame> Tag Attributes*

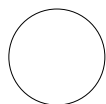
Table 2: Attributes of the <frame> tag

Attribute	Description
Src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, <code>src="/html/top_frame.htm"</code> will load an HTML file available in html directory.
Name	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
marginwidth	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example <code>marginwidth="10"</code> .
marginheight	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example <code>marginheight="10"</code> .
Noresize	By default you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example <code>noresize="noresize"</code> .
Scrolling	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example <code>scrolling="no"</code> means it should not have scroll bars.
Longdesc	This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example <code>longdesc="framedescription.htm"</code>

### *Frame's name and target attributes*

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.

Let's see following example where a test.htm file has following code:



```
<!DOCTYPE html>
<html><head>
<title>HTML Target Frames</title>
</head>
<frameset cols="200, *">
  <frame src="/html/menu.htm" name="menu_page" />
  <frame src="/html/main.htm" name="main_page" />
</frameset>
<body>
  Your browser does not support frames.
</body>
</frameset></html>
```

Following is the content of menu.htm file

```
<!DOCTYPE html>
<html>
<body bgcolor="#4a7d49">
<a href="http://www.google.com" target="main_page">Google</a>
<br /><br />
<a href="http://www.microsoft.com" target="main_page">Microsoft</a>
<br /><br />
<a href="http://news.bbc.co.uk" target="main_page">BBC News</a>
</body>
</html>
```

Following is the content of main.htm file:

```
<!DOCTYPE html>
<html>
<body bgcolor="#b5dcb3">
<h3>This is main page and content from any link will be displayed here.</h3>
<p>So now click any link and see the result.</p>
</body></html>
```

When we load test.htm file, it produces following result:

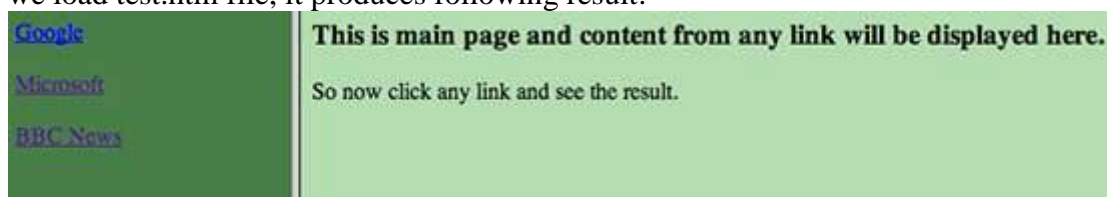
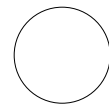


Fig. 3: Output Result

Now you can try to click links available in the left panel and see the result.

Table 3: The target attribute possible values

Option	Description
_self	Loads the page into the current frame.
_blank	Loads a page into a new browser window.opening a new window.
_parent	Loads the page into the parent window, which in the case of a single frameset is the main browser window.
_top	Loads the page into the browser window, replacing any current frames.
targetframe	Loads the page into a named targetframe.

**Exercises:**

1. Create a following layout with horizontal and vertical frames.

<b>Page 1</b>	<b>Page 1</b>
	<b>Page 2</b>
	<b>Page 3</b>

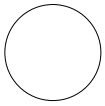
2. Create a vertical navigation bar using frames.

When user click a hyperlinks in the navigation bar (left side of the page), the appropriate page is loaded into the hyperlinks target frame. (Right side of the page)

<u><a href="#">Page 1</a></u>	Display page content Here
<u><a href="#">Page 2</a></u>	
<u><a href="#">Page 3</a></u>	

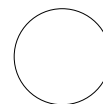
**Review Questions:**

1. How do I create frames? What is a frameset?
2. How can I check for errors?



**[This page is intentionally left blank]**





## HTML Forms

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** To study About HTML Forms.

### Theory

HTML Forms are required when you want to collect some data from the site visitor. For example during user registration you would like to collect information such as name, email address, credit card, etc. [2]

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application. [2]

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

### *The <form> Element*

HTML forms are used to collect user input. The <form> element defines an HTML form:

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

### *Form Attributes*

Table 1: Most frequently used form attributes

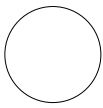
Attribute	Description
Action	Backend script ready to process your passed data.
Method	Method to be used to upload data. The most frequently used are GET and POST methods.
Target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.
Enctype	You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are: <ul style="list-style-type: none"><li>• <b>application/x-www-form-urlencoded</b> - This is the standard method most forms use in simple scenarios.</li><li>• <b>multipart/form-data</b> - This is used when you want to upload binary data in the form of files like image, word file etc.</li></ul>

We will study about the following [1]

- HTML Forms
- HTML Form Elements
- HTML Input Types
- HTML Input Attributes

### HTML Form Elements

- <input>
- <select>
- <textarea>
- <button>



### ***The <input> Element***

The most important form element is the <input> element.

The <input> element can vary in many ways, depending on the type attribute.

### ***The <select> Element (Drop-Down List)***

The <select> element defines a drop-down list:

**Example:**

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

The <option> elements defines the options to select. The list will normally show the first item as selected. You can add a selected attribute to define a predefined option.

**Example:**

```
<option value="fiat" selected>Fiat</option>
```

### ***The <textarea> Element***

The <textarea> element defines a multi-line input field (a text area):

**Example:**

```
<textarea name="message" rows="10" cols="30">
  The cat was playing in the garden.
</textarea>
```

### ***The <button> Element***

The <button> element defines a clickable button:

**Example:**

```
<button type="button" onclick="alert('Hello World!')">Click Me!</button>
```

### ***HTML5 Form Elements***

HTML5 added the following form elements:

- <datalist>
- <keygen>
- <output>

## **HTML Input Types**

### ***HTML 4 Input Types***

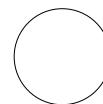
Describes the input types of the <input> element.

- Text
- Password
- Submit
- Radio
- Checkbox
- Button

### ***HTML5 Input Types***

HTML5 added several new input types:

- color
- date
- datetime



- datetime-local
- email
- month
- number
- range
- search
- tel
- time
- url
- week

### 1. *Input Type: text*

`<input type="text">` defines a one-line input field for text input:

**Example:**

```
<form>
  First name:<br>
  <input type="text" name="firstname">
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

**Output:**

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

First name:

Last name:

### 2. *Input Type: password*

`<input type="password">` defines a password field:

**Example:**

```
<form>
  User name:<br>
  <input type="text" name="username">
  <br>
  User password:<br>
  <input type="password" name="psw">
</form>
```

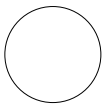
### 3. *Input Type: submit*

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

**Example:**

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

Note: If you omit the submit button's value attribute, the button will get a default text



#### 4. Input Type: radio

`<input type="radio">` defines a radio button.

Radio buttons let a user select ONLY ONE of a limited number of choices:

**Example:**

```
<form>
  <input type="radio" name="sex" value="male" checked> Male
  <br>
  <input type="radio" name="sex" value="female"> Female
</form>
```

**Output:**

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

- ☒ Male
- ☐ Female

#### 5. Input Type: checkbox

`<input type="checkbox">` defines a checkbox.

Checkboxes let a user select ZERO or MORE options of a limited number of choices.

**Example:**

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

**Output:**

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

- ☐ I have a bike
- ☐ I have a car

#### 6. Input Type: button

`<input type="button">` defines a button:

Example

```
<input type="button" onclick="alert('Hello World!')" value="Click Me!">
```

### HTML Input Attributes

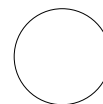
#### **HTML 4 Input Attributes**

- The value Attribute
- The readonly Attribute
- The disabled Attribute
- The size Attribute
- The maxlength Attribute

#### **HTML5 Attributes**

HTML5 added the following attributes for `<input>`:

- autocomplete
- autofocus
- form
- formaction



- formenctype
- formmethod
- formnovalidate
- formtarget
- height and width
- list
- min and max
- multiple
- pattern (regex)
- placeholder
- required
- step

And the following attributes for <form>:

- autocomplete
- novalidate

### 1. *The value Attribute*

The value attribute specifies the initial value for an input field:

Example

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John">
  <br>
</form>
```

Output:

First name:

### 2. *The readonly Attribute*

The readonly attribute specifies that the input field is read only (cannot be changed):

Example:

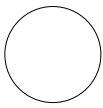
```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" readonly>
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

### 3. *The disabled Attribute*

The disabled attribute specifies that the input field is disabled. A disabled element is un-usable and un-clickable. Disabled elements will not be submitted.

Example:

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" disabled>
  <br>
  Last name:<br>
  <input type="text" name="lastname"></form>
```



#### 4. *The size Attribute*

The size attribute specifies the size (in characters) for the input field:

**Example:**

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" value="John" size="40">
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

#### 5. *The maxlength Attribute*

The maxlength attribute specifies the maximum allowed length for the input field:

**Example:**

```
<form action="">
  First name:<br>
  <input type="text" name="firstname" maxlength="10">
  <br>
  Last name:<br>
  <input type="text" name="lastname">
</form>
```

### **HTTP Methods: GET vs. POST**

#### ***What is HTTP?***

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

#### ***Two HTTP Request Methods: GET and POST***

Two commonly used methods for a request-response between a client and server are: GET and POST.

- GET - Requests data from a specified resource
- POST - Submits data to be processed to a specified resource

#### **The GET Method**

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo\_form.asp?name1=value1&name2=value2

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

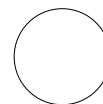
#### **The POST Method**

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

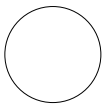


Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Table 2: Comparison of GET and POST

	<b>GET</b>	<b>POST</b>
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL  Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

**Exercises:**

1. To study About HTML Forms.
  - a) Make the following form in HTML editor.

The form is displayed on a light blue background. It contains the following elements:

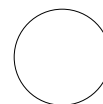
- First Name:
- Last Name:
- Gender:
  - Male: ☐
  - Female: ☐
- Favorite Food:
  - Steak: ☐
  - Pizza: ☐
  - Chicken: ☐
- Enter your favorite quote!
- Select a Level of Education:
  - Jr.High
- Select your favorite time of day:
  - Morning
  - Day
  - Night

At the bottom right of the form area is a dark blue button labeled "Continue".

**Review Questions:**

1. How can I use tables to structure forms?
2. How do I use forms?
3. Can I have two or more actions in the same form?
4. What is the difference between form get and form post?
5. Explain HTML5 Attributes.





## CSS

**Practical No:**\_\_\_\_\_

**Date:**\_\_\_\_\_

**Aim:** To study about Cascading Style Sheets (CSS).

### Theory

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable.

CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, layout designs, and variations in display for different devices and screen sizes as well as a variety of other effects.

CSS is easy to learn and understand but it provides powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML. [1]

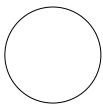
### *Advantages of CSS* [1]

- CSS saves time – You can write CSS once and then reuse same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many Web pages as you want.
- Pages load faster – If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So less code means faster download times.
- Easy maintenance – To make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
- Superior styles to HTML – CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- Multiple Device Compatibility – Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cell phones or for printing.
- Global web standards – Now HTML attributes are being deprecated and it is being recommended to use CSS. So its a good idea to start using CSS in all the HTML pages to make them compatible to future browsers.
- Offline Browsing – CSS can store web applications locally with the help of an offline cache. Using of this, we can view offline websites. The cache also ensures faster loading and better overall performance of the website.
- Platform Independence – The Script offer consistent platform independence and can support latest browsers as well.

### *CSS Versions* [1]

**CSS1** - Cascading Style Sheets, level 1 (CSS1) was came out of W3C as a recommendation in December 1996. This version describes the CSS language as well as a simple visual formatting model for all the HTML tags.

**CSS2** - was became a W3C recommendation in May 1998 and builds on CSS1. This version adds support for media-specific style sheets e.g. printers and aural devices, downloadable fonts, element positioning and tables.



**CSS3** - was became a W3C recommendation in June 1999 and builds on older versions CSS. it has divided into documentations is called as Modules and here each module having new extension features defined in CSS2.

### ***CSS3 Modules***

CSS3 Modules are having old CSS specifications as well as extension features.

- Selectors
- Box Model
- Backgrounds and Borders
- Image Values and Replaced Content
- Text Effects
- 2D/3D Transformations
- Animations
- Multiple Column Layout
- User Interface

## **Styling HTML with CSS**

Styling can be added to HTML elements in 3 ways:

- **Inline** - using a style attribute in HTML elements
- **Internal** - using a <style> element in the HTML <head> section
- **External** - using one or more external CSS files

### ***1. Inline Styling (Inline CSS)***

Inline styling is used to apply a unique style to a single HTML element:

Inline styling uses the style attribute.

This example changes the text color of the <h1> element to blue:

Example

```
<h1 style="color:blue;">This is a Blue Heading</h1>
```

### ***2. Internal Styling (Internal CSS)***

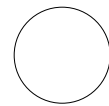
Internal styling is used to define a style for one HTML page.

Internal styling is defined in the <head> section of an HTML page, within a <style> element:

**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <style>
      body {background-color:lightgrey;}
      h1 {color:blue;}
      p {color:green;}
    </style>
  </head>

  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```



### 3. External Styling (External CSS)

An external style sheet is used to define the style for many pages. With an external style sheet, you can change the look of an entire web site by changing one file! To use an external style sheet, add a link to it in the <head> section of the HTML page:

**Example:**

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

  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
  </body>
</html>
```

Here is how the "styles.css" looks:

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

h1
{
  color: blue;
}

p
{
  color:green;
}
```

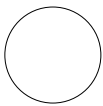
### CSS Syntax

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

Table	// Selector
{	
Color: Blue;	//Declaration
Font-size: 15px;	//Declaration
}	

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

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



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

## CSS Selectors

- Element/Type Selector
- Id Selector
- Class Selector
- Grouping Selector
- Child Selector

### 1. *Element/Type Selector*

This is the same selector we have seen above. Again, one more example to give a color to all level 1 headings:

```
h1  
{  
    color: #36CFFF;  
}
```

### 2. *Id Selector*

The id selector uses the id attribute of an HTML element to select a specific element. An id should be unique within a page, so the id selector is used if you want to select a single, unique element.

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

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

**Example:**

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

Note: Do NOT start an ID name with a number!

This rule renders the content in black for every element with id attribute set to black in our document.

You can make it a bit more particular. For example –

```
h1#black {  
    color: #000000;  
}  
}
```

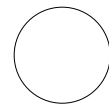
This rule renders the content in black for only <h1> elements with id attribute set to black.

The true power of id selectors is when they are used as the foundation for descendant selectors, For

**Example:**

```
#black h2 {  
    color: #000000; }
```

In this example all level 2 headings will be displayed in black color when those headings will lie with in tags having id attribute set to black.



### 3. *Class Selector*

The class selector selects elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the name of the class.

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

#### **Example:**

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

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

#### **Example:**

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

### 4. *Grouping Selector*

If you have elements with the same style definitions, like this:

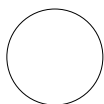
You can group the selectors, to minimize the code. To group selectors, separate each selector with a comma.

In the example below we have grouped the selectors from the code above:

```
h1 {
    text-align: center;
    color: red;
}
h2 {
    text-align: center;
    color: red;
}
p {
    text-align: center;
    color: red;
}}
```

#### **Example:**

```
h1, h2, p
{
    text-align: center;
    color: red;}
```



## 5. Child Selector

You have seen the descendant selectors. There is one more type of selector, which is very similar to descendants but have different functionality. Consider the following example –

```
body > p
{
    color: #000000;
}
```

This rule will render all the paragraphs in black if they are direct child of <body> element. Other paragraphs put inside other elements like <div> or <td> would not have any effect of this rule.

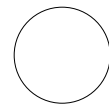
```
p {
    color: red;
    /* This is a single-line comment */
    text-align: center;
}
/* This is a multi-line comment */
```

Many times, you may need to put additional comments in your style sheet blocks. So, it is very easy to comment any part in style sheet. You can simply put your comments inside /\*.....this is a comment in style sheet.....\*/.

## CSS - Measurement Units

Table 1: CSS Measurement Units

Unit	Description	Example
%	Defines a measurement as a percentage relative to another value, typically an enclosing element.	p {font-size: 16pt; line-height: 125%;}
cm	Defines a measurement in centimeters.	div {margin-bottom: 2cm;}
em	A relative measurement for the height of a font in em spaces. Because an em unit is equivalent to the size of a given font, if you assign a font to 12pt, each "em" unit would be 12pt; thus, 2em would be 24pt.	p {letter-spacing: 7em;}
ex	This value defines a measurement relative to a font's x-height. The x-height is determined by the height of the font's lowercase letter x.	p {font-size: 24pt; line-height: 3ex;}
in	Defines a measurement in inches.	p {word-spacing: .15in;}
mm	Defines a measurement in millimeters.	p {word-spacing: 15mm;}
pc	Defines a measurement in picas. A pica is equivalent to 12 points; thus, there are 6 picas per inch.	p {font-size: 20pc;}
pt	Defines a measurement in points. A point is defined as 1/72nd of an inch.	body {font-size: 18pt;}
px	Defines a measurement in screen pixels.	p {padding: 25px;}
vh	1% of viewport height.	h2 {font-size: 3.0vh;}
vw	1% of viewport width	h1 {font-size: 5.9vw;}
vmin	1vw or 1vh, whichever is smaller	p {font-size: 2vmin;}



## CSS – Fonts

In CSS, there are two types of font family names:

- Generic family - a group of font families with a similar look (like "Serif" or "Monospace")
- Font family - a specific font family (like "Times New Roman" or "Arial")

Table 2: CSS Font property

Property	Description
<b>Font</b>	Sets all the font properties in one declaration
<b>font-family</b>	Specifies the font family for text
<b>font-size</b>	Specifies the font size of text
<b>font-style</b>	Specifies the font style for text
<b>font-variant</b>	Specifies whether or not a text should be displayed in a small-caps font
<b>font-weight</b>	Specifies the weight of a font

### *Set the Font Family*

Following is the example, which demonstrates how to set the font family of an element. Possible value could be any font family name.

```
<html>
<head>
</head>
<body>
  <p style="font-family:georgia,garamond,serif;">
    This text is rendered in either georgia, garamond, or the default serif font
    depending on which font you have at your system.
  </p>
</body>
</html>
```

### *Font Style*

The font-style property is mostly used to specify italic text.

This property has three values:

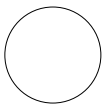
- normal - The text is shown normally
- italic - The text is shown in italics
- oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

Example

```
p.normal {
  font-style: normal;
}

p.italic {
  font-style: italic;
}

p.oblique {
  font-style: oblique;
}
```



## Font Size

The font-size property sets the size of the text. Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs.

The font-size value can be an absolute, or relative size.

Absolute size:

- Sets the text to a specified size
- Does not allow a user to change the text size in all browsers (bad for accessibility reasons)
- Absolute size is useful when the physical size of the output is known

Relative size:

- Sets the size relative to surrounding elements
- Allows a user to change the text size in browsers

Note: If you do not specify a font size, the default size for normal text, like paragraphs, is 16px (16px=1em).

## Set Font Size with Pixels

Setting the text size with pixels gives you full control over the text size:

Example:

```
h1 {
  font-size: 40px;
}

h2 {
  font-size: 2.5em; /* 40px/16=2.5em */
}
```

## Font Weight

The font-weight property specifies the weight of a font:

Example 1

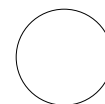
```
p.normal {
  font-weight: normal;
}

p.thick {
  font-weight: bold;
}
```

Example 2

```
<html>
  <head>
  </head>
  <body>
    <p style="font-weight:bold;">This font is bold.</p>
    <p style="font-weight:bolder;">This font is bolder.</p>
    <p style="font-weight:500;">This font is 500 weight.</p>
  </body>
</html>
```





### Font Variant

The font-variant property specifies whether or not a text should be displayed in a small-caps font.

In a small-caps font, all lowercase letters are converted to uppercase letters. However, the converted uppercase letters appears in a smaller font size than the original uppercase letters in the text.

Example

```
p.normal {  
    font-variant: normal;}  
p.small {  
    font-variant: small-caps;}
```

CSS uses color values to specify a color. Typically, these are used to set a color either for the foreground of an element (i.e., its text) or else for the background of the element. They can also be used to affect the color of borders and other decorative effects.

You can specify your color values in various formats. Following table lists all the possible formats –

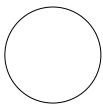
Table 3: CSS colors

Format	Syntax	Example
Hex Code	#RRGGBB	p{color:#FF0000;}
Short Hex Code	#RGB	p{color:#6A7;}
RGB %	rgb(rrr%,ggg%,bbb%)	p{color:rgb(50%,50%,50%);}
RGB Absolute	rgb(rrr,ggg,bbb)	p{color:rgb(0,0,255);}
keyword	aqua, black, etc.	p{color:teal;}

### CSS Text

Table 4: CSS Text property

Property	Description
Color	Sets the color of text. Possible value could be any color name in any valid format.
Direction	Specifies the text direction/writing direction. Possible values are ltr or rtl.
letter-spacing	Increases or decreases the space between characters in a text. Possible values are normal or a number specifying space.
line-height	Sets the line height
text-align	Specifies the horizontal alignment of text. Possible values are left, right, center, justify.
text-decoration	Specifies the decoration added to text. Possible values are none, underline, overline, line-through, blink.
text-indent	Specifies the indentation of the first line in a text-block. Possible values are % or a number specifying indent space.
text-shadow	Specifies the shadow effect added to text
text-transform	Controls the capitalization of text. Possible values are none, capitalize, uppercase, lowercase.
unicode-bidi	Used together with the <a href="#">direction</a> property to set or return whether the text should be overridden to support multiple languages in the same document
vertical-align	Sets the vertical alignment of an element
white-space	Specifies how white-space inside an element is handled. Possible values are normal, pre, nowrap.
word-spacing	Increases or decreases the space between words in a text. Possible values are normal or a number specifying space.
Line Height	The line-height property is used to specify the space between lines



### Examples:

```
<p style="color:red;">
<p style="direction:rtl;">
<p style="letter-spacing:5px;">
<p style="word-spacing:5px;">
<p style="text-indent:1cm;">
<p style="text-align:right;">
<p style="text-decoration:underline;">
<p style="text-transform:capitalize;">
<p style="white-space:pre;">
```

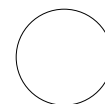
```
body
{
    color: blue;
}

h1
{
    color: green;
    text-align: center;
    text-decoration: none;
    text-decoration: underline;
    text-transform: capitalize;
    text-indent: 50px;
    line-height: 0.8;
}
```

## CSS Backgrounds

Table 5: CSS Background properties

Property	Description
Background	Sets all the background properties in one declaration
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page
background-color	Sets the background color of an element
background-image	Sets the background image for an element
background-position	Sets the starting position of a background image
background-repeat	Sets how a background image will be repeated. Background Image - Repeat Horizontally or Vertically



### Examples:

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

```
body {  
  background-color: lightblue;  
}
```
2. The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.  

```
body {  
  background-image: url("paper.gif");  
}
```
3. 

```
body {  
  background-image: url("gradient_bg.png");  
  background-repeat: repeat-x;  
}
```
4. 

```
body {  
  background-image: url("img_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
}
```
5. Background Image - Fixed position  
To specify that the background image should be fixed (will not scroll with the rest of the page), use the background-attachment property:  
  

```
background-attachment: fixed;
```

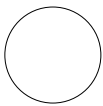
## CSS - Borders

The border properties allow you to specify how the border of the box representing an element should look. There are three properties of a border you can change:

- **The border-color** specifies the color of a border.
- **The border-style** specifies whether a border should be solid, dashed line, double line, or one of the other possible values.
- **The border-width** specifies the width of a border.

Table 6: CSS Border property

border-color Properties	border-style Properties	border-width Properties
border-bottom-color border-top-color border-left-color border-right-color	none: No border. (Equivalent of border-width:0;) solid: Border is a single solid line. dotted: Border is a series of dots. dashed: Border is a series of short lines. double: Border is two solid lines. groove: Border looks as though it is carved into the page. ridge: Border looks the opposite of groove. inset: Border makes the box look like it is	border-bottom-width changes the width of bottom border. border-top-width changes the width of top border. border-left-width changes the width of left border. border-right-width changes the width of right border



	<p>embedded in the page.</p> <p>outset: Border makes the box look like it is coming out of the canvas.</p> <p>hidden: Same as none, except in terms of border-conflict resolution for table elements.</p> <p>border-bottom-style changes the style of bottom border.</p> <p>border-top-style changes the style of top border.</p> <p>border-left-style changes the style of left border.</p> <p>border-right-style changes the style of right border.</p>	
--	---	--

### ***Border - Shorthand Property***

As you can see from the examples above, there are many properties to consider when dealing with borders. To shorten the code, it is also possible to specify all the individual border properties in one property. The border property is a shorthand property for the following individual border properties:

- border-width
- border-style (required)
- border-color

#### **Example:**

```
p {  
  border: 5px solid red;  
}
```

#### ***Example***

```
Bordercss1  
{  
  border:1px solid;  
  border-bottom-color:#009900; /* Green */  
  border-top-color:#FF0000; /* Red */  
  border-left-color:#330000; /* Black */  
  border-right-color:#0000CC; /* Blue */  
}
```

## **CSS Margin Properties**

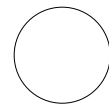
The margin property defines the space around an HTML element. It is possible to use negative values to overlap content.

The values of the margin property are not inherited by the child elements. Remember that the adjacent vertical margins (top and bottom margins) will collapse into each other so that the distance between the blocks is not the sum of the margins, but only the greater of the two margins or the same size as one margin if both are equal.

### ***Margin - Individual Sides***

CSS has properties for specifying the margin for each side of an element:

- margin-top
- margin-right
- margin-bottom
- margin-left



All the margin properties can have the following values:

- auto - the browser calculates the margin
- length - specifies a margin in px, pt, cm, etc.
- % - specifies a margin in % of the width of the containing element
- inherit - specifies that the margin should be inherited from the parent element

**Example:**

```
<!DOCTYPE html>
<html>
<head>
<style>
p {
  background-color: yellow;
}
p.ex {
  border: 1px solid red;
  margin-top: 100px;
  margin-bottom: 100px;
  margin-right: 150px;
  margin-left: 80px;
}
</style>
</head>
<body>

<h2>Using Individual margin Properties:</h2>

<p>This is a paragraph with no specified margins.</p>
<p class="ex">This paragraph has a top and bottom margin of 100px, a left margin of 80px, and a right margin of 150px.</p>

</body>
</html>
```

**Output:**

### Using Individual margin Properties:

This is a paragraph with no specified margins.

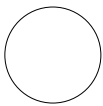
This paragraph has a top and bottom margin of 100px, a left margin of 80px, and a right margin of 150px.

### ***Margin - Shorthand Property***

To shorten the code, it is possible to specify all the margin properties in one property.

The margin property is a shorthand property for the following individual margin properties:

margin-top margin-right margin-bottom margin-left

**Example:**

```
p {  
  margin: 100px 150px 100px 80px;  
}
```

## CSS Padding Properties

The CSS padding properties are used to generate space around content.

The padding properties set the size of the white space between the element content and the element border.

### *Padding - Individual Sides*

CSS has properties for specifying the padding for each side of an element:

- padding-top
- padding-right
- padding-bottom
- padding-left

All the padding properties can have the following values:

- length - specifies a padding in px, pt, cm, etc.
- % - specifies a padding in % of the width of the containing element
- inherit - specifies that the padding should be inherited from the parent element

The following example sets different padding for all four sides of a <p> element:

**Example:**

```
p {  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}
```

**Output:**

This paragraph has a top and bottom padding of 50px, a left padding of 80px, and a right padding of 30px.

## CSS Height and Width Dimensions

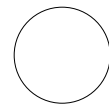
The CSS dimension properties allow you to control the height and width of an element.

The height and width properties are used to set the height and width of an element.

The height and width can be set to auto (this is default. Means that the browser calculates the height and width), or be specified in length values, like px, cm, etc., or in percent (%) of the containing block.

Table 7: CSS Height and Width Dimensions

Property	Description
height	Sets the height of an element
max-height	Sets the maximum height of an element



max-width	Sets the maximum width of an element
min-height	Sets the minimum height of an element
min-width	Sets the minimum width of an element
Width	Sets the width of an element

**Example:**

```
div {  
  width: 500px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}
```

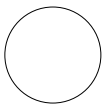
**Output:**

This div element has a height of 100px and a width of 500px.

## CSS Tables

You can set following properties of a table –

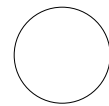
- ***The border-collapse*** specifies whether the browser should control the appearance of the adjacent borders that touch each other or whether each cell should maintain its style.
- ***The border-spacing*** specifies the width that should appear between table cells.
- ***The caption-side*** captions are presented in the <caption> element. By default, these are rendered above the table in the document. You use the caption-side property to control the placement of the table caption.
- ***The empty-cells*** specifies whether the border should be shown if a cell is empty.
- ***The table-layout*** allows browsers to speed up layout of a table by using the first width properties it comes across for the rest of a column rather than having to load the whole table before rendering it.



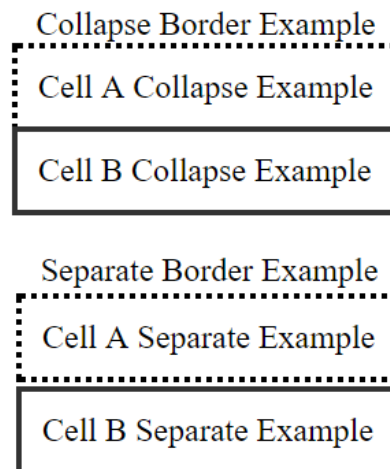
### Example:

```
<html>
  <head>
    <style type="text/css">
      table.one {border-collapse:collapse;}
      table.two {border-collapse:separate;}
      td.a {
        border-style:dotted;
        border-width:3px;
        border-color:#000000;
        padding: 10px;
      }
      td.b {
        border-style:solid;
        border-width:3px;
        border-color:#333333;
        padding:10px;
      }
    </style>
  </head>
  <body>
    <table class="one">
      <caption>Collapse Border Example</caption>
      <tr><td class="a"> Cell A Collapse Example</td></tr>
      <tr><td class="b"> Cell B Collapse Example</td></tr>
    </table>
    <br />
    <table class="two">
      <caption>Separate Border Example</caption>
      <tr><td class="a"> Cell A Separate Example</td></tr>
      <tr><td class="b"> Cell B Separate Example</td></tr>
    </table>
  </body>
</html>
```





## Output:



## CSS Layout - The position Property

The position property specifies the type of positioning method used for an element (static, relative, fixed or absolute).

### ***The position Property***

The position property specifies the type of positioning method used for an element.

There are four different position values:

- static
- relative
- fixed
- absolute

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the position property is set first. They also work differently depending on the position value.

### ***Position: static***

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page:

### ***Position: relative***

An element with position: relative; is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

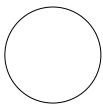
### ***Position: fixed***

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

### ***Position: absolute***

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).



However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: A "positioned" element is one whose position is anything except static.

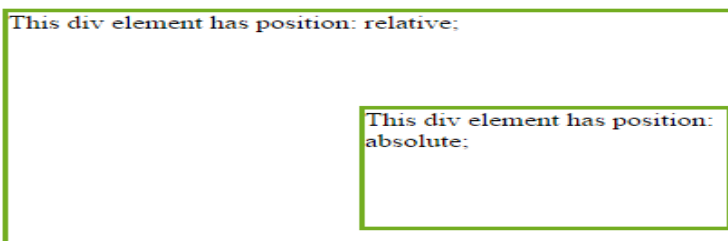
**Example:**

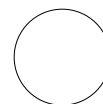
```
<!DOCTYPE html>
<html>
<head>
<style>
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}
div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
</style>
</head>
<body>
<h2>position: absolute;</h2>
<p>An element with position: absolute; is positioned relative to the nearest positioned
ancestor (instead of positioned relative to the viewport, like fixed):</p>
<div class="relative">This div element has position: relative;
  <div class="absolute">This div element has position: absolute;</div>
</div>
</body>
</html>
```

**Output:**

**position: absolute;**

An element with position: absolute; is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed):





## Exercises:

1. Make the following table with same format. Use the external style sheet.  
(width of table- 600 px)

Row 1	data 1	data 2	data 3
Row 1	data 1	data 2	data 3
Row 1	data 1	data 2	data 3
Row 1	data 1	data 2	data 3
Row 1	data 1	data 2	data 3

Movie Title	Genre	Year	Gross
Star Wars	Adventure, Sci-fi	1977	\$460,935,665
Howard The Duck	"Comedy"	1986	\$16,295,774

2. Prepare following form with using external style sheet.

☒ Registration

Account Details

Email \*

Repeat email \*

Password\*

Repeat Password\*

\* obligatory fields

Personal Details

Name \*

Phone \*

Street

City \*

Country \*

Website

http://

Further Information

Gender \*

Birthdate \*

Nationality \*

Children \*

Male Female 01 January e.g 1976

☐

HELPFUL INFORMATION

Here comes some explaining text, sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

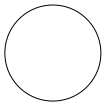
Terms and Mailing

☐ \* I accept the Terms and Conditions

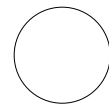
☐ I want to receive personalized offers by your site

☐ Allow partners to send me personalized offers and related services

Register »

**Review Questions:**

1. Explain the three main ways to apply CSS styles to a Web page:
2. What is an ID selector and how is it used?
3. What are the different CSS properties used to change dimensions and what values can they accept?



## Javascript

**Practical No:**\_\_\_\_\_

**Date:**\_\_\_\_\_

**Aim:** To study about Basic Javascript (Client side scripting with JavaScript, variables, functions, conditions, loops and repetition, Pop up boxes, Javascript and objects, JavaScript own objects, forms and validations).

### Theory

#### *What is 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.

JavaScript was first known as LiveScript, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name LiveScript. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The ECMA-262 Specification defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform

#### *Client-side JavaScript*

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

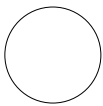
JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

#### *JavaScript - Syntax*

JavaScript can be implemented using JavaScript statements that are placed within the <script>...</script> HTML tags in a web page.

You can place the <script> tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the <head> tags.

The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.



```
<script ...>  
  JavaScript code  
</script>
```

The script tag takes two important attributes –

1. Language – This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
2. Type – This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript segment will look like –

```
<script ...>  
  JavaScript code  
</script>
```

### **Example:**

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "`!-->`". Here "`/*`" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function `document.write` which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>  
  <body>  
    <script language="javascript" type="text/javascript">  
      <!--  
        document.write("Hello World!")  
      //-->  
    </script>  
  </body>  
</html>
```

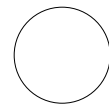
This code will produce the following result –  
Hello World!

### ***Whitespace and Line Breaks***

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

### ***Semicolons are Optional***

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your



statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10
    var2 = 20
  //-->
</script>
```

But when formatted in a single line as follows, you must use semicolons –

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10; var2 = 20;
  //-->
</script>
```

Note – It is a good programming practice to use semicolons.

### ***Case Sensitivity***

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

NOTE – Care should be taken while writing variable and function names in JavaScript.

### ***Comments in JavaScript***

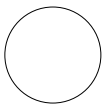
JavaScript supports both C-style and C++-style comments, Thus –

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

### **Example:**

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">
  <!--
    // This is a comment. It is similar to comments in C++
  /*
    * This is a multiline comment in JavaScript
    * It is very similar to comments in C Programming
  */
  //-->
</script>
```



## **JavaScript - Placement in HTML File**

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows –

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

### ***JavaScript in <head>...</head> section***

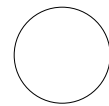
If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows –

```
<html>
  <head>
    <script type="text/javascript">
      <!--
        function sayHello() {
          alert("Hello World")
        }
      //-->
    </script>
  </head>
  <body>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </body>
</html>
```

### ***JavaScript in <body>...</body> section***

```
<html>
  <head>
  </head>
  <body>
    <script type="text/javascript">
      <!--
        document.write("Hello World")
      //-->
    </script>
    <p>This is web page body </p>
  </body>
</html>
```





### ***JavaScript in External File***

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The script tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using script tag and its src attribute.

```
<html>
  <head>
    <script type="text/javascript" src="filename.js" ></script>
  </head>
  <body>
    .....
  </body>
</html>
```

### **JavaScript Datatypes**

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

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.

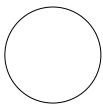
### **JavaScript Variables**

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the var keyword as follows.

#### **Example:**

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

**Example:**

For instance, you might create a variable named money and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

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

### JavaScript – 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

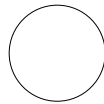
Lets have a look on all operators one by one.

```
<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);
    -->
  </script>
</body>
</html>
```



```
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>
```

### **JavaScript Conditional statements**

#### ***JavaScript If...Else Statements***

##### **Syntax:**

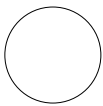
```
if (condition) {
    block of code to be executed if the condition is true
}
```

#### ***The else if Statement***

Use the else if statement to specify a new condition if the first condition is false.

##### **Syntax:**

```
if (condition1) {
    block of code to be executed if condition1 is true
} else if (condition2) {
    block of code to be executed if the condition1 is false and condition2 is true
} else {
    block of code to be executed if the condition1 is false and condition2 is false
}
```



**Example:** If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

The **output** of greeting will be:

Good day

### JavaScript loop statements

#### ***Different Kinds of Loops***

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

#### ***The For Loop***

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following **Syntax**:

```
for (statement 1; statement 2; statement 3) {  
    code block to be executed  
}
```

Statement 1 is executed before the loop (the code block) starts.

Statement 2 defines the condition for running the loop (the code block).

Statement 3 is executed each time after the loop (the code block) has been executed.

#### **Example:**

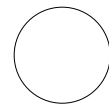
```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

#### ***The While Loop***

The while loop loops through a block of code as long as a specified condition is true.

**Syntax:**

```
while (condition) {  
    code block to be executed  
}
```

**Example:**

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

```
Example
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

**JavaScript - Page Redirection*****What is Page Redirection ?***

You might have encountered a situation where you clicked a URL to reach a page X but internally you were directed to another page Y. It happens due to page redirection. This concept is different from JavaScript Page Refresh.

***How Page Re-direction Works ?***

The implementations of Page-Redirection are as follows.

**Example 1:**

It is quite simple to do a page redirect using JavaScript at client side. To redirect your site visitors to a new page, you just need to add a line in your head section as follows.

```
<html>
<head>

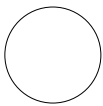
    <script type="text/javascript">
        <!--
            function Redirect() {
                window.location="http://www.tutorialspoint.com";
            }
        //-->
    </script>

</head>

<body>
    <p>Click the following button, you will be redirected to home page.</p>

    <form>
        <input type="button" value="Redirect Me" onclick="Redirect();" />
    </form>

</body>
</html>
```



## **JavaScript - Dialog Boxes**

### ***Alert Dialog Box***

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

### **Example:**

```
<html>
  <head>
    <script type="text/javascript">
      <!--
        function Warn() {
          alert ("This is a warning message!");
          document.write ("This is a warning message!");
        }
      //-->
    </script>
  </head>
  <body>
    <p>Click the following button to see the result: </p>
    <form>
      <input type="button" value="Click Me" onclick="Warn();" />
    </form>
  </body>
</html>
```

## **JavaScript - Objects Overview**

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.

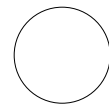
### ***Object Properties***

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;
```

### **Object Methods**

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the this keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

For example – Following is a simple example to show how to use the write() method of document object to write any content on the document.

```
document.write("This is test");
```

### **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");
```

### **The Object() Constructor**

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called Object() to build the object. The return value of the Object() constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the var keyword.

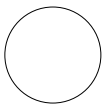
#### **Example 1:**

Try the following example; it demonstrates how to create an Object.

```
<html> <head>  
  <title>User-defined objects</title>  
  <script type="text/javascript">  
    var book = new Object(); // Create the object  
    book.subject = "Perl"; // Assign properties to the object  
    book.author = "Mohtashim";  
  </script>  
</head>  
<body>  
  <script type="text/javascript">  
    document.write("Book name is : " + book.subject + "<br>");  
    document.write("Book author is : " + book.author + "<br>");  
  </script>  
</body></html>
```

#### **Output:**

Book name is : Perl



Book author is : Mohtashim

### ***Defining Methods for an Object***

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

#### **Example:**

Try the following example; it shows how to add a function along with an object.

```
<html>
<head>
<title>User-defined objects</title>
  <script type="text/javascript">
    // Define a function which will work as a method
    function addPrice(amount){
      this.price = amount;
    }
    function book(title, author){
      this.title = title;
      this.author = author;
      this.addPrice = addPrice; // Assign that method as property.
    }
  </script>
</head>
<body>
  <script type="text/javascript">
    var myBook = new book("Perl", "Mohtashim");
    myBook.addPrice(100);
    document.write("Book title is : " + myBook.title + "<br>");
    document.write("Book author is : " + myBook.author + "<br>");
    document.write("Book price is : " + myBook.price + "<br>");
  </script>
</body>
</html>
```

#### **Output:**

Book title is : Perl

Book author is : Mohtashim

Book price is : 100

### **JavaScript HTML DOM**

With the HTML DOM, JavaScript can access and change all the elements of an HTML document.

#### ***The HTML DOM (Document Object Model)***

When a web page is loaded, the browser creates a Document Object Model of the page.



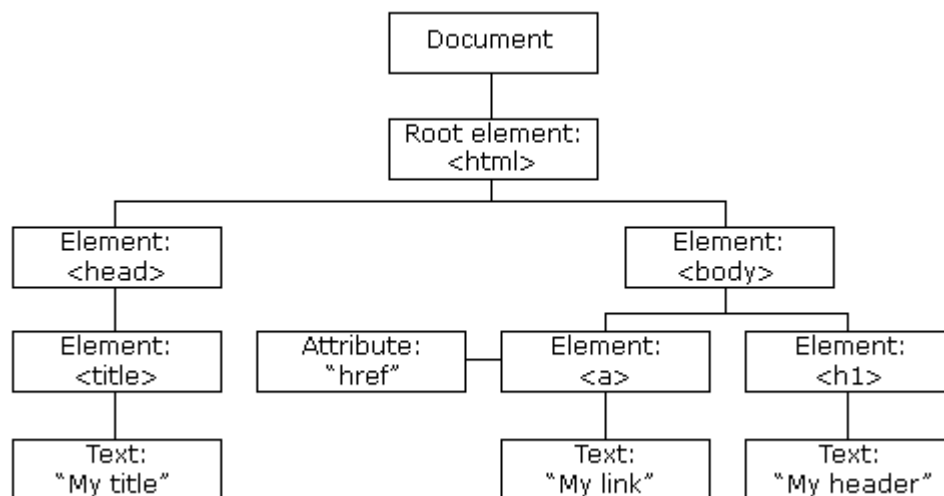
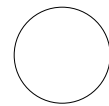


Fig. 1: The HTML DOM model tree of Objects:

### ***What is the DOM?***

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

The W3C DOM standard is separated into 3 different parts:

- Core DOM - standard model for all document types
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

### ***JavaScript - HTML DOM Methods***

HTML DOM methods are actions you can perform (on HTML Elements). HTML DOM properties are values (of HTML Elements) that you can set or change.

### **The DOM Programming Interface**

The HTML DOM can be accessed with JavaScript (and with other programming languages). In the DOM, all HTML elements are defined as objects. The programming interface is the properties and methods of each object. A property is a value that you can get or set (like changing the content of an HTML element).

A method is an action you can do (like add or deleting an HTML element).

### **Example:**

The following example changes the content (the innerHTML) of the <p> element with id="demo":

```
<html>
<body>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
</body>
</html>
```

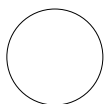


Table 1: Finding HTML Elements

Method	Description
<code>document.getElementById(id)</code>	Find an element by element id
<code>document.getElementsByTagName(name)</code>	Find elements by tag name
<code>document.getElementsByClassName(name)</code>	Find elements by class name

Table 2: Changing HTML Elements

Method	Description
<code>element.innerHTML = new html content</code>	Change the inner HTML of an element
<code>element.attribute = new value</code>	Change the attribute value of an HTML element
<code>element.setAttribute(attribute, value)</code>	Change the attribute value of an HTML element
<code>element.style.property = new style</code>	Change the style of an HTML element

Table 3: Adding and Deleting Elements

Method	Description
<code>document.createElement(element)</code>	Create an HTML element
<code>document.removeChild(element)</code>	Remove an HTML element
<code>document.appendChild(element)</code>	Add an HTML element
<code>document.replaceChild(element)</code>	Replace an HTML element
<code>document.write(text)</code>	Write into the HTML output stream

Table 4: Adding Events Handlers

Method	Description
<code>document.getElementById(id).onclick = function(){code}</code>	Adding event handler code to an onclick event

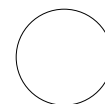
### Finding HTML Objects

The first HTML DOM Level 1 (1998), defined 11 HTML objects, object collections, and properties. These are still valid in HTML5.

Later, in HTML DOM Level 3, more objects, collections, and properties were added.

Table 5: HTML Objects

Property	Description	DOM
<code>document.anchors</code>	Returns all <a> elements that have a name attribute	1
<code>document.applets</code>	Returns all <applet> elements (Deprecated in HTML5)	1
<code>document.baseURI</code>	Returns the absolute base URI of the document	3
<code>document.body</code>	Returns the <body> element	1
<code>document.cookie</code>	Returns the document's cookie	1
<code>document.doctype</code>	Returns the document's doctype	3
<code>document.documentElement</code>	Returns the <html> element	3
<code>document.documentMode</code>	Returns the mode used by the browser	3
<code>document.documentURI</code>	Returns the URI of the document	3
<code>document.domain</code>	Returns the domain name of the document server	1
<code>document.domConfig</code>	Obsolete. Returns the DOM configuration	3
<code>document.embeds</code>	Returns all <embed> elements	3
<code>document.forms</code>	Returns all <form> elements	1



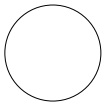
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

### Exercises

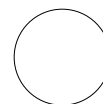
1. Write a JavaScript to find first 10 prime numbers.
2. Write a JavaScript for following scenario.
  - a) Take one button.
  - b) When we click on button it prompt dialog box.
  - c) In the dialog box there is one text box for entering your enrollment no.
  - d) Also this dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.
3. Write a JavaScript for numeric validation on Textbox

### Review Questions:

1. How to read elements of an array in JavaScript?
2. What is the difference between undefined and not defined in JavaScript?
3. What is callback?



**[This page is intentionally left blank]**



## XML

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** To study about Extensible Markup Language (XML), Document Type Definition (DTD) and Extensible Stylesheet Language Transformations (XSLT).

### Theory

#### XML

XML, the Extensible Markup Language, is a semi structured data model that has been proposed as the standard for data exchange on the Web. It is a simplified version of SGML (ISO 8879). XML meets the requirements of a flexible, generic, and platform-independent language, as presented earlier. Any XML document can be serialized with a normalized encoding, for storage or transmission on the Internet.

It is well-established to use the term “XML document” to denote a hierarchically structured content represented with XML conventions. Although we adopt this standard terminology, please keep in mind that by “document” we mean both the content and its structure, but not their specific representation which may take many forms. Also note that “document” is reminiscent of the SGML application area, which mostly focuses on representing technical documentation. An XML document is not restricted to textual, human-readable data, and can actually represent any kind of information, including images, of references to other data sources. XML is a standard for representing data but it is also a family of standards (some in progress) for the management of information at a world scale: XLink, XPointer, XML Schema, DOM, SAX, XPath, XSL, XQuery, SOAP, WSDL, and so forth.

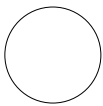
#### *XML documents*

An XML document is a labeled, unranked, ordered tree:

- **Labeled** means that some annotation, the label, is attached to each node.
- **Unranked** means that there is no a priori bound on the number of children of a node.
- **Ordered** means that there is an order between the children of each node.

The document can be serialized as follows:

```
<entry><name><fn>Jean</fn><ln>Doe</ln></name>INRIA<adress><city>
Cachan</city><zip>94235</zip></adress><email>j@inria.fr</email>
</job><purpose>like to teach</purpose></entry>
or with some beautification as
<entry>
<name>
<fn>Jean</fn>
<ln>Doe</ln> </name>
<work>
INRIA
<adress>
<city>Cachan</city>
<zip>94235</zip> </adress>
<email>j@inria.fr</email> </work>
<purpose>like to teach</purpose> </entry>
```



In this serialization, the data corresponding to the subtree with root labeled (e.g., work), is represented by a subword delimited by an opening tag `<work>` and a closing tag `</work>`. One should never forget that this is just a serialization. The conceptual (and mathematical) view of an XML document is that it is a labeled, unranked, ordered tree. XML specifies a “syntax” and no a priori semantics. So, it specifies the content of a document but not its behavior or how it should be processed. The labels have no predefined meaning unlike in HTML, where, for example, the label `href` indicates a reference and `img` an image. Clearly, the labels will be assigned meaning by applications.

In HTML, one uses a predefined (finite) set of labels that are meant primarily for document presentation. For instance, consider the following HTML document:

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
Abiteboul, Hull, Vianu
<br/> Addison Wesley, 1995 </p>
<p> <i> Data on the Web </i>
Abiteboul, Buneman, Suciu
<br/> Morgan Kaufmann. 1999 </p>
```

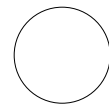
where `<h1>` indicates a title, `<p>` a paragraph, `<i>` italics and `<br/>` a line break (`<br/>` is both an opening and a closing tag, gathered in a concise syntax equivalent to `<br></br>`).

Observe that this is in fact an XML document; more precisely this text is in a particular XML dialect, called XHTML. (HTML is more tolerant and would, for instance, allow omitting the `</p>` closing tags.)

The layout of the document depends closely on the interpretation of these labels by the browser. One would like to use different layouts depending on the usage (e.g., for a mobile phone or for blind people). A solution for this is to separate the content of the document and its layout so that one can generate different layouts based on the actual software that is used to present the document. Indeed, early on, Tim Berners-Lee (the creator of HTML) advocated the need for a language that would go beyond HTML and distinguish between content and presentation. The same bibliographical information is found, for instance, in the following XML document:

```
<bibliography>
<book>
<title> Foundations of Databases </title>
<author> Abiteboul </author> <author> Hull </author>
<author> Vianu </author>
<publisher> Addison Wesley </publisher>
<year> 1995 </year> </book>
<book>...</book>
</bibliography>
```

Observe that it does not include any indication of presentation. There is a need for a stylesheet (providing transformation rules) to obtain a decent presentation such as that of the HTML document. On the other hand, with different stylesheets, one can obtain documents for several media (e.g., also for PDF). Also, the tags produce some semantic information that can be used by applications, (e.g., Addison Wesley is the publisher of the book). Such tag information turns out to be useful for instance to support more precise search than that provided by Web browser or to integrate information from various sources.



### DTD

XML documents need not typed. They may be. The first kind of typing mechanism originally introduced for XML is DTDs, for Document Type Definitions. DTDs are still quite often used. We will study in XML schema, which is more powerful and is becoming more standard, notably because it is used in Web services.

An XML document including a type definition is as follows:

```
<?xml version="1.0" standalone="yes" ?>
<!-- This is a comment - Example of a DTD -->
<!DOCTYPE email [
<!ELEMENT email ( header, body )>
<!ELEMENT header ( from, to, cc? )>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT cc (#PCDATA)>
<!ELEMENT body (paragraph*) >
<!ELEMENT paragraph (#PCDATA)>
<email>
<header>
<from> af@abc.com </from>
<to> zd@ugh.com </to>
</header>
<body>
</body>
</email>
```

The DOCTYPE clause declares the type for this document. Such a type declaration is not compulsory. Ignoring the details of this weird syntax, this is stating, for instance, that a header is composed of a from element, a to one, and possibly a cc one, that a body consists of a list of paragraphs, and finally that a paragraph is a string.

In general, the list of children for a given element name is described using a regular expression in BNF specified for that element.

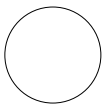
A most important notion is also that of namespace. Consider a label such as job. It denotes different notions for a hiring agency or for a (computer) application service provider.

Applications should not confuse the two notions. The notion of namespace is used to distinguish them. More precisely, consider the following XML piece of data:

```
<doc xmlns:hire='http://a.hire.com/schema'
xmlns:asp='http://b.asp.com/schema' >
...
<hire:job> ... </hire:job> ...
<asp:job> ... </asp:job> ...
</doc>
```

The hire namespace is linked to a schema, and the asp to another one. One can now mix the two vocabularies inside the same document in spite of their overlap.

XML also provides some referencing mechanisms that we will ignore for now.



When a type declaration is present, the document must conform to the type. This may, for instance, be verified by an application receiving a document before actually processing it. If a well-formed document conforms to the type declaration (if present), we say that it is valid (for this type).

### XSLT

The role of XSLT is to extract information from an input XML document and to transform it into an output document, often in XML, which is also something that XQuery can do. Therefore, both languages seem to compete with one another, and their respective advantages and downsides with respect to a specific application context may not be obvious at first glance.

Essentially:

- XSLT is good at transforming documents, and is for instance very well adapted to map the content of an XML document to an XHTML format in a Web application.

Example:

***XML File:***

```
<?xml version="1.0"?>
<?xml-stylesheet href="style1.xsl" type="text/xsl"?>
<items>
  <item partNum="123-AB">
    <productName>Porsche</productName>
    <quantity>1</quantity>
  </item>
  <item>
    <productName>Ferrari</productName>
    <quantity>2</quantity>
  </item>
</items>
```

***XSL Template***

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
  <html>
    <head></head> <body> <ol>
      <li>Root</li> <xsl:apply-templates/>
    </ol> </body>
  </html>
</xsl:template>
<xsl:template match="items">
  <li>Items</li>
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="item">
  <li>Item: <xsl:value-of select="productName"/></li>
</xsl:template></xsl:stylesheet>
```



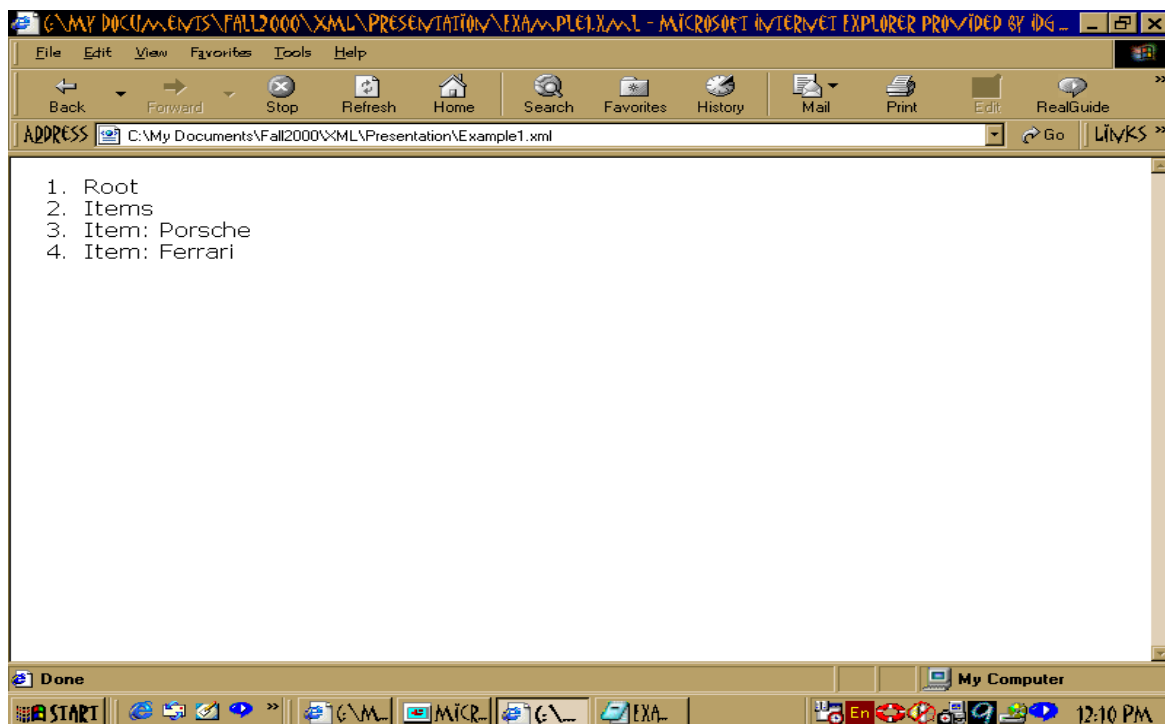
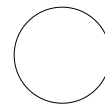


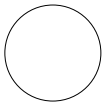
Fig. 1: Output

**Exercises:**

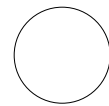
1. Write a program to make XML file for Railway Reservation System.
2. Write a program for any system to verify data typing of XML data with DTD.

**Review Questions:**

1. What is difference between XML Schema and DTD?
2. Describe the differences between XML and HTML.
3. Where is XML such an important development?



**[This page is intentionally left blank]**



# PHP

**Practical No:**\_\_\_\_\_

**Date:**\_\_\_\_\_

**Aim:** Introduction and basic syntax of PHP.

## Theory

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases. PHP is basically used for developing web based software applications.

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994. [1]

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

## *What is a PHP File?*

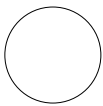
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

## *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.

## *How PHP Works?*

The PHP software works with the web server, which is the software that delivers web pages to the world. When you type a URL into your web browser's address bar, you're sending a message to the web server at that URL, asking it to send you an HTML file. The web server responds by sending the requested file. Your browser reads the HTML file and displays the web page. You also request a file from the web server when you click a link in a web page. In addition, the web server processes a file when you click a web page button that submits a form. This process is essentially



the same when PHP is installed. You request a file, the web server happens to be running PHP, and it sends HTML back to the browser.

### ***How the web server processes PHP files***

When a browser is pointed to a regular HTML file with an .html or .htm extension, the web server sends the file, as is, to the browser. The browser processes the file and displays the web page described by the HTML tags in the file. When a browser is pointed to a PHP file (with a .php extension), the web server looks for PHP sections in the file and processes them or, more exactly, hands them to the PHP processor, instead of just sending them as is to the browser. The web server/PHP processor processes the PHP file as follows:

- The web server starts scanning the file in HTML mode. It assumes the statements are HTML and sends them to the browser without any processing.
- The web server continues in HTML mode until it encounters a PHP opening tag (<?php).
- When it encounters a PHP opening tag, the web server hands the processing over to the PHP module. This is sometimes called escaping from HTML. The web server then assumes that all statements are PHP statements and uses the PHP module to execute the PHP statements. If there is output from PHP, the server sends the output to the browser.
- The web server continues in PHP mode until it encounters a PHP closing tag (?>).
- When the web server encounters a PHP closing tag, it returns to HTML mode. It resumes scanning, and the cycle continues from Step 1.

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** – PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here – <http://httpd.apache.org/download.cgi>
- **Database** – PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here – <http://www.mysql.com/downloads/>
- **PHP Parser** – In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser. This tutorial will guide you how to install PHP parser on your computer.

### **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
?>
```

The default file extension for PHP files is ".php".

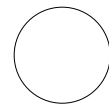
A PHP file normally contains HTML tags, and some PHP scripting code.

Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

**Example:**

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

Note: 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.

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

### ***PHP Case Sensitivity***

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

However; all variable names are case-sensitive.

### **PHP Variables**

#### ***Creating (Declaring) PHP Variables***

In PHP, a variable starts with the \$ sign, followed by the name of the variable:

#### **Example:**

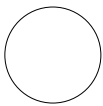
```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

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 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:

```
echo "Hello";
```

**Output:** Hello

```
echo 123;
```

**Output:** 123

```
echo "Hello", "World!";
```

**Output:** HelloWorld!

```
echo Hello World!;
```

**Output:** Not valid; results in an error message

### Example:

```
<?php  
$txt = "Hello";  
echo "Hi $txt!";  
?>
```

### Example:

```
<?php  
$txt = "W3Schools.com";  
echo "I love " . $txt . "!";  
?>
```

PHP has three different variable scopes:

1. local
2. global
3. static

## PHP 5 Data Types

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

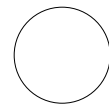
PHP supports the following data types:

1. Integer: A whole number
2. Floating-point number (float): A numeric value with decimal digits
3. String: A series of characters
4. Boolean: A value that can be either true or false
5. NULL: A value that represents no value
6. Array: A group of values in one variable
7. Object: A structure created with a class
8. Resource: A reference that identifies a connection

### *PHP determines the data type automatically.*

When writing PHP scripts, you don't need to specify which data type you're storing. The following two statements store different data types:

```
$var1 = 123;
```



```
$var2 = "123";
```

The value for \$var1 is stored as an integer. The value for \$var2 is stored as a string because it's enclosed in quotes.

***PHP converts data types automatically when it needs to.***

For instance, if you add two variables, one containing an integer and one containing a float, PHP converts the integer to a float so that it can add the two.

***You can determine the data type.***

Occasionally, you might want to store a value as a data type different than the data type PHP automatically stores. You can set the data type for a variable with a cast, as follows:

```
$var3 = "222";
```

```
$var4 = (int) $var3;
```

***Creating arrays***

The simplest way to create an array is to assign a value to a variable with square brackets ([ ]) at the end of its name. For instance, assuming that you haven't referenced \$cities at any earlier point in the script, the following statement creates an array called \$cities:

```
$cities[1] = "Phoenix";
```

```
$cities[2] = "Tucson";
```

```
$cities[3] = "Flagstaff";
```

***PHP String Functions***

1. The PHP strlen() function returns the length of a string.
2. The PHP str\_word\_count() function counts the number of words in a string.
3. The PHP strrev() function reverses a string.
4. 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.
5. The PHP str\_replace() function replaces some characters with some other characters in a string.

**Example:**

1. echo strlen("Hello world!"); // outputs 12
2. echo str\_word\_count("Hello world!"); // outputs 2
3. echo strrev("Hello world!"); // outputs !dlrow olleH
4. echo strpos("Hello world!", "world"); // outputs 6
5. echo str\_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!

### **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***

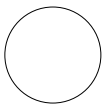
To create a constant, use the define() function.

**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:

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

### **Formatting a date**

The function that you will use most often is `date`, which converts a date or time from the timestamp format into a format that you specify. The general format is

```
$mydate = date("format",$timestamp);
```

`$timestamp` is a variable with a timestamp stored in it. This code assumes that you previously stored the timestamp in the variable, which you might do using a PHP function (as we'll describe later in this section). If `$timestamp` isn't included, the current time is obtained from the operating system and used. Thus, you can get today's date with the following:

```
$today = date("Y/m/d");
```

If today is August 10, 2013, this statement returns 2013/08/10

The format is a string that specifies the date format that you want stored in the variable. For instance, the format "y-m-d" returns 13-08-10, and "M.d.Y" returns Aug.10.2013. Table 1-6 lists some of the symbols that you can use in the format string. (For a complete list of symbols, see the documentation at [www.php.net/manual/en/function.date.php](http://www.php.net/manual/en/function.date.php).) The parts of the date can be separated by a hyphen (-), a dot (.), a forward slash (/), or a space.

### ***Storing a timestamp in a variable***

You can assign a timestamp with the current date and time to a variable with the following statement:

```
$today = time();
```

Another way to store a current timestamp is with the statement

```
$today = strtotime("today");
```

You can store specific timestamps by using `strtotime` with various keywords and abbreviations that are similar to English. For instance, you can create a timestamp for January 15, 2014, as follows:

```
$importantDate = strtotime("January 15 2014");
```

## **PHP Operators**

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

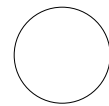
- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

## **PART 2**

- **Conditional statements:** Statements that execute only when certain conditions are met. The PHP conditional statements are `if` and `switch` statements.

- **Loops:** Statements that repeatedly execute a block of statements. Four types of loops are `foreach`, `for`, `while`, and `do..while` loops.





- **Functions:** Statements that can be reused many times. Many tasks are performed in more than one part of the application. PHP allows you to reuse statement blocks by creating a function.

### 1. Conditional statements

In PHP we have the following conditional statements:

- if statement - executes some code only if a specified condition is true
- if...else statement - executes some code if a condition is true and another code if the condition is false
- if...elseif....else statement - specifies a new condition to test, if the first condition is false
- switch statement - selects one of many blocks of code to be executed

#### PHP - The if Statement

The if statement is used to execute some code only if a specified 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:

```
<?php  
$t = date("H");  
  
if ($t < "20") {  
    echo "Have a good day!";  
}  
?>
```

#### PHP - The if...else Statement

Use the if...else statement to execute some code if a condition is true and another code if the condition is false.

##### Syntax

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
if (condition) {  
    code to be executed if condition is true;} else {  
    code to be executed if condition is false;}
```

##### Example:

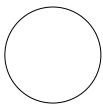
```
<?php  
$t = date("H");  
if ($t < "20") { echo "Have a good day!";} else  
{  
    echo "Have a good night!";} ?>
```

#### PHP - The if...elseif....else Statement

Use the if....elseif...else statement to specify a new condition to test, if the first condition is false.

##### Syntax

```
if (condition) {  
    code to be executed if condition is true;
```



```
} elseif (condition) {  
    code to be executed if condition is true;  
} else {  
    code to be executed if condition is false;}
```

### **The PHP switch Statement**

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; }
```

## **2. Loops**

In PHP, we have the following looping statements:

- while - loops through a block of code as long as the specified condition is true
- 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 condition is 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:**

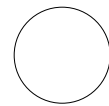
```
<?php  
$x = 1;  
while($x <= 5) {  
    echo "The number is: $x <br>";  
    $x++;  
} ?>
```

### **The PHP for Loop**

The for loop is used when you know in advance how many times the script should run.

#### **Syntax:**

```
for (init counter; test counter; increment counter) {  
    code to be executed;  
}
```

**Example:**

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

**PHP 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;
}
```

Note: A function name can start with a letter or underscore (not a number).

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:**

```
<?php
function writeMsg() {
    echo "Hello world!";
}
writeMsg(); // call the function?>
```

**PHP Function Arguments**

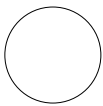
Information can be passed to functions through arguments. An argument is just like a variable.

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:**

```
<?php
function familyName($fname) {echo "$fname Refsnes.<br>";}
familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```



### 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:

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

### PHP Functions - Returning values

To let a function return a value, use the return statement:

#### Example:

```
<?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);
?>
```

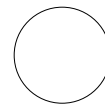
You use an include statement to bring the content of an external text file into your script. The format for an include statement is

```
include("filename");
```

The file can have any name. We, your humble book authors, like to use the extension .inc so that we know the file is an include file as soon as we see the name. It helps with the organization and clarity of a website.

PHP provides four types of include statements:

- include: Includes and evaluates the specified file. It displays a warning if it can't find the specified file.
- require: Performs the same way as the include statement, except that it produces, in addition to a warning, a fatal error when it can't find the specified file, stopping the script at that point.
- include\_once: Performs the same as the include statement, except it includes the file only once. If the file has already been included, it won't be included again. In some scripts, a file might be included more than once, causing function redefinitions, variable reassignments, and other possible problems.
- require\_once: Performs the same as the require statement, except it includes the file only once. If the file has already been included, it won't be included again. This statement prevents problems that might occur when a file is included more than once.

**Example:**

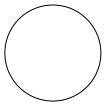
```
<html>
  <body>
    <?php include("menu.php"); ?>
    <p>This is an example to show how to include PHP file!</p>
  </body>
</html>
```

**Exercises:**

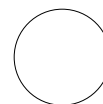
1. Write a PHP Code to illustrate the following concepts.
  - a) Print "Hello World!"
  - b) Embedding PHP within HTML
  - c) PHP single line comments (Contain program definition)
  - d) Creating variables in PHP (define variable name, enrollment\_no, college name, branch name, gender)
2. Write a PHP code to implement following concepts.  
Print the date/time in following format.
  - a) 11/12/2015
  - b) 11-12-2015
  - c) December 11, 2015 07:26:09 AM
  - d) 07:26 am
3. Write a PHP code for given number is odd or even. (Use If..Else Condition)
4. Write a PHP code that print the odd and even number from 1 to 100 (Use For loop)
5. Write a PHP code that for basic calculator operations (Use switch case)

**Review Questions:**

1. What are the different types of PHP variables?
2. What is the purpose of `_CLASS_` constant?
3. What is the purpose `$_PHP_SELF` variable?



**[This page is intentionally left blank]**



## PHP Forms

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** To study about PHP Form Processing, Files, Cookies and Sessions.

### Theory

#### PHP - Files & I/O

Following functions related to files –

- Opening a file, Reading a file
- Writing a file, Closing a file

#### **1. Opening and Closing Files**

The PHP fopen() function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Table 1: Files modes

Mode	Purpose
<b>r</b>	Opens the file for reading only. Places the file pointer at the beginning of the file.
<b>r+</b>	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
<b>w</b>	Opens the file for writing only. Places the file pointer at the beginning of the file. And truncates the file to zero length. If files does not exist then it attempts to create a file.
<b>w+</b>	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. And truncates the file to zero length. If files does not exist then it attempts to create a file.
<b>a</b>	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
<b>a+</b>	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

#### **2. Reading a file**

Once a file is opened using fopen() function it can be read with a function called fread(). This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the filesize() function which takes the file name as its argument and returns the size of the file expressed in bytes.

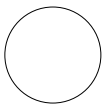
So here are the steps required to read a file with PHP.

1. Open a file using fopen() function.
2. Get the file's length using filesize() function.
3. Read the file's content using fread() function.
4. Close the file with fclose() function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

**Example:**

```
<html> <head>
  <title>Reading a file using PHP</title>
</head> <body>
  <?php
    $filename = "tmp.txt";
    $file = fopen( $filename, "r" );
    if( $file == false )
```



```
{
    echo ( "Error in opening file" );
    exit();
}

$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );
fclose( $file );
    echo ( "File size : $filesize bytes" );
echo ( "<pre>$filetext</pre>" );
?>
</body>
</html>
```

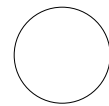
### 3. *Writing a file*

A new file can be written or text can be appended to an existing file using the PHP `fwrite()` function. This function requires two arguments specifying a file pointer and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using `file_exists()` function which takes file name as an argument

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
    if( $file == false ) {    echo ( "Error in opening new file" );
    exit(); }
fwrite( $file, "This is a simple test\n" );
fclose( $file ); ?>
<html>  <head>
    <title>Writing a file using PHP</title>
</head> <body>
    <?php
        $filename = "newfile.txt";
        $file = fopen( $filename, "r" );
            if( $file == false )
        {    echo ( "Error in opening file" );
        exit();    }
        $filesize = filesize( $filename );
        $filetext = fread( $file, $filesize );
        fclose( $file );
        echo ( "File size : $filesize bytes" );
        echo ( "$filetext" );
        echo("file name: $filename");
    ?></body></html>
```





### **PHP Form Handling**

The PHP superglobals \$\_GET and \$\_POST are used to collect form-data.

**NOTE: Please read GET and POST in Experiment no- 5**

#### ***PHP - A Simple HTML Form***

The example below displays a simple HTML form with two input fields and a submit button:

**Example:**

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

#### **Output:**

Welcome John

Your email address is john.doe@example.com

#### ***GET vs. POST***

Both GET and POST create an array (e.g. array( key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

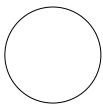
Both GET and POST are treated as \$\_GET and \$\_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

\$\_GET is an array of variables passed to the current script via the URL parameters.

\$\_POST is an array of variables passed to the current script via the HTTP POST method.

#### ***When to use GET?***

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.



GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

### ***When to use POST?***

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

### **PHP – Cookies**

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

1. Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
2. Browser stores this information on local machine for future use.
3. When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

### ***The Anatomy of a Cookie***

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this –

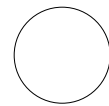
```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
           path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this –

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables `$_COOKIE` or `$HTTP_COOKIE_VARS[]` which holds all cookie names and values. Above cookie can be accessed using `$HTTP_COOKIE_VARS["name"]`.



### ***Setting Cookies with PHP***

PHP provided setcookie() function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

setcookie(name, value, expire, path, domain, security);

Here is the detail of all the arguments –

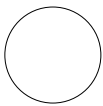
- Name – This sets the name of the cookie and is stored in an environment variable called HTTP\_COOKIE\_VARS. This variable is used while accessing cookies.
- Value – This sets the value of the named variable and is the content that you actually want to store.
- Expiry – This specifies a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- Path – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- Domain – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- Security – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies name and age these cookies will be expired after one hour.

```
<?php
    setcookie("name", "John Watkin", time()+3600, "/", "", 0);
    setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>
    <head>
        <title>Setting Cookies with PHP</title>
    </head>
    <body>
        <?php echo "Set Cookies"?>
    </body>
</html>
```

### ***Accessing Cookies with PHP***

PHP provides many ways to access cookies. Simplest way is to use either \$\_COOKIE or HTTP\_COOKIE\_VARS variables. Following example will access all the cookies set in above example.



```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      echo $_COOKIE["name"]. "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"]. "<br />";
      echo $_COOKIE["age"] . "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["age"] . "<br />";
    ?>
  </body>
</html>
```

### ***Deleting Cookie with PHP***

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```
<?php
  setcookie( "name", "", time()- 60, "/", "", 0);
  setcookie( "age", "", time()- 60, "/", "", 0);
?>
<html>
  <head>
    <title>Deleting Cookies with PHP</title>
  </head>
  <body>
    <?php echo "Deleted Cookies" ?>
  </body>
</html>
```

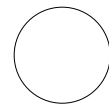
## **PHP Sessions**

### ***What is a PHP Session?***

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So, Session variables hold information about one single user, and are available to all pages in one application.



### ***Start a PHP Session***

A session is started with the session\_start() function.

Session variables are set with the PHP global variable: \$\_SESSION.

Now, let's create a new page called "demo\_session1.php". In this page, we start a new PHP session and set some session variables:

#### **Example:**

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favourite"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>
```

Note: The session\_start() function must be the very first thing in your document. Before any HTML tags.

### ***Get PHP Session Variable Values***

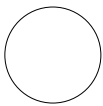
Next, we create another page called "demo\_session2.php". From this page, we will access the session information we set on the first page ("demo\_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session\_start()).

Also notice that all session variable values are stored in the global \$\_SESSION variable:

#### **Example:**

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favourite"] . ".";
?>
</body>
</html>
```



### ***Destroy a PHP Session***

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

#### **Example:**

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
?>
</body></html>
```

#### **Exercises:**

1. Write a PHP code for open the file and write your basic details (students).
2. Prepare following form in HTML-PHP and print the all the filled details in the bottom of the page when we click on SUBMIT button.

**Class Registration Form**

Name:

E-mail:

Specific Time:

Class details:

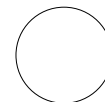
Gender: ☐ Female ☐ Male

**Your Given details are as :**

3. Write a HTML PHP code for Setting a cookies and accessing a cookies.
4. Write a Program for PHP Session.

#### **Review Questions:**

1. What is the purpose of `$_FILES` variable in PHP?
2. Opening and Closing Files in php
3. What is the use of `rand()` in php?



## PHP and MySQL

**Practical No:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Aim:** To study about PHP and MySQL Basic commands and data base operation (Update and Delete)

### Theory

#### *What is MySQL*

MySQL is one of the most popular relational database systems being used on the Web today. It is freely available and easy to install, however if you have installed Wampserver it already there on your machine. MySQL database server offers several advantages:

- MySQL is easy to use, yet extremely powerful, secure, and scalable.
- MySQL runs on a wide range of operating systems, including UNIX, Microsoft Windows, Apple Mac OS X, and others.
- MySQL supports standard SQL (Structured Query Language).
- MySQL is ideal database solution for both small and large applications.
- MySQL is developed, and distributed by Oracle Corporation.
- MySQL is very fast and secure. It includes data security layers that protect sensitive data from intruders.
- MySQL database stores data into tables like other relational database. A table is a Collection of related data, and it is divided into rows and columns.

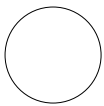
#### *PHP MySQL contain following contents*

- Connecting to MySQL database – Learn how to use PHP to open and close a MySQL database connection.
- Create MySQL Database Using PHP – This part explains how to create MySQL database and tables using PHP.
- Delete MySQL Database Using PHP – This part explains how to delete MySQL database and tables using PHP.
- Insert Data to MySQL Database – Once you have created your database and tables then you would like to insert your data into created tables. This session will take you through real example on data insert.
- Retrieve Data from MySQL Database – Learn how to fetch records from MySQL database using PHP.
- Using Paging through PHP – this one explains how to show your query result into multiple pages and how to create the navigation link.
- Updating Data into MySQL Database – this part explains how to update existing records into MySQL database using PHP.
- Deleting Data from MySQL Database – this part explains how to delete or purge existing records from MySQL database using PHP.
- Using PHP to Backup MySQL Database – Learn different ways to take backup of your MySQL database for safety purpose.

#### **PHP Connect to MySQL Server**

##### *Open a Connection to MySQL Database Server*

In order to access the data inside a MySQL database, you first need to open a connection to the MySQL database server. In PHP you can easily do this using the `mysqli_connect()` function. All



communication between PHP and the MySQL database server takes place through this connection. The basic syntax of the `mysqli_connect()` function is given with:

`mysqli_connect(host, username, password, dbname);`

Table 1: MySQL Parameters

Parameter	Description
<b>Optional</b>	All the parameters are optional
Host	Either a host name or an IP address
Username	The MySQL user name
Password	The MySQL password to get access
Dbname	The name of the MySQL database to use

**Example:**

```
<?php
/*
Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password)
*/
$link = mysqli_connect("localhost", "root", "");

// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error());
}
?>
```

If you want to close it earlier you can do this by simply calling the PHP `mysqli_close()` function.

`mysqli_close($link);`

**PHP MySQL Create Database and Tables**

***Creating the MySQL Database***

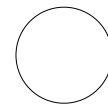
Since all tables are stored in a database, so first we have to create a database before creating tables. The `CREATE DATABASE` statement is used to create a database in MySQL.

Let's make a SQL query using the `CREATE DATABASE` statement, after that we will execute this SQL query through passing it to the `mysqli_query()` function to finally create our database. The following example creates a database named "demo".

**Example:**

```
<?php
/* Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password) */
$link = mysqli_connect("localhost", "root", "");
// Attempt create database query execution
$sql = "CREATE DATABASE demo";
//execute the query
mysqli_query($link, $sql)
}
// Close connection
mysqli_close($link);?>
```





### ***Adding Tables to MySQL Database***

Since our database is created now it's time to add some tables to it. The CREATE TABLE statement is used to create a table in MySQL database.

So let's make a SQL query using the CREATE TABLE statement, after that we will execute this SQL query through passing it to the mysqli\_query() function to finally create our table.

```
<?php
/*
Attempt MySQL server connection. Assuming you are running MySQL
server with default setting (user 'root' with no password)
*/
$link = mysqli_connect("localhost", "root", "", "demo");
// Attempt create table query execution
$sql = "CREATE TABLE persons(person_id INT(4) NOT NULL PRIMARY KEY
AUTO_INCREMENT, first_name CHAR(30) NOT NULL, last_name CHAR(30) NOT
NULL, email_address VARCHAR(50))";
mysqli_query($link, $sql)
// Close connection
mysqli_close($link);
?>
```

### **PHP MySQL Retrieve the data**

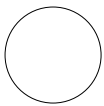
#### ***mysql\_fetch\_array()***

This function returns row as an associative array, a numeric array, or both. This function returns FALSE if there are no more rows.

#### ***mysql\_fetch\_assoc()***

PHP provides another function called mysql\_fetch\_assoc() which also returns the row as an associative array.

```
<?php
$dbhost = 'localhost:3036';
$dbuser = 'root';
$dbpass = 'rootpassword';
$conn = mysql_connect($dbhost, $dbuser, $dbpass);
$sql = 'SELECT emp_id, emp_name, emp_salary FROM employee';
mysql_select_db('test_db');
$retval = mysql_query( $sql, $conn );
while($row = mysql_fetch_array($retval, MYSQL_ASSOC))
{
    echo "EMP ID :{$row['emp_id']} <br> ".
        "EMP NAME : {$row['emp_name']} <br> ".
        "EMP SALARY : {$row['emp_salary']} <br> ".
        "-----<br>";
}
echo "Fetched data successfully\n";
mysql_close($conn);
?>
```



### Insert Data into MySQL Database

Now that you've understood how to create database and tables in MySQL. In this tutorial you will learn how to execute SQL query to insert records in a table.

Let's make a SQL query using the INSERT INTO statement with appropriate values, after that we will execute this SQL query through passing it to the mysqli\_query() function to insert data in table. Here's an example, which adds a record to the persons table by specifying values for the 'person\_id', 'first\_name', 'last\_name' and 'email\_address' fields:

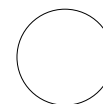
```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Attempt insert query execution
$sql = "INSERT INTO persons (person_id, first_name, last_name, email_address)
VALUES (1, 'Peter', 'Parker', 'peterparker@mail.com)";
mysqli_query($link, $sql)
// Close connection
mysqli_close($link);
?>
```

### **Insert Data into a Database from an HTML Form**

Let's create an HTML form that can be used to insert new records to persons table.

Here's a simple HTML form that has three text <input> fields and a submit button.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Add Records Form</title>
</head>
<body>
<form action="#" method="post">
    <p>
        <label for="firstName">First Name:</label>
        <input type="text" name="firstname" id="firstName">
    </p>
    <p>
        <label for="lastName">Last Name:</label>
        <input type="text" name="lastname" id="lastName">
    </p>
    <p>
        <label for="emailAddress">Email Address:</label>
        <input type="text" name="email" id="emailAddress">
    </p>
    <input type="submit" value="Add Records">
</form></body></html>
```



## Retrieving and Inserting the Form Data

When a user clicks the submit button of the add records HTML form, in the example above, the form data is sent to 'insert.php' file. The 'insert.php' file connects to the MySQL database server, retrieves forms fields using the PHP \$\_POST variables and finally execute the insert query to add the records. Here is the complete code of our 'insert.php' file:

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Check connection
if($link === false){
    die("ERROR: Could not connect. " . mysqli_connect_error()); }
// Escape user inputs for security
$first_name = mysqli_real_escape_string($link, $_POST['firstname']);
$last_name = mysqli_real_escape_string($link, $_POST['lastname']);
$email_address = mysqli_real_escape_string($link, $_POST['email']);
// attempt insert query execution
$sql = "INSERT INTO persons (first_name, last_name, email_address) VALUES
('$first_name', '$last_name', '$email_address')";
if(mysqli_query($link, $sql))
{
    echo "Records added successfully.";
}
Else
{
    echo "ERROR: Could not able to execute $sql. " . mysqli_error($link); }
// close connection
mysqli_close($link);
?>
```

The UPDATE statement is used to change or modify the existing records in a database table. It is typically used in conjugation with the WHERE clause to apply the changes to only those records that matches specific criteria.

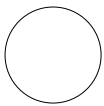
The basic syntax of the UPDATE statement can be given with:

UPDATE      table\_name      SET      column1=value,      column2=value2,...      WHERE  
column\_name=some\_value

### Example:

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Attempt update query execution
$sql = "UPDATE persons SET email_address='peterparker_new@mail.com' WHERE
first_name='Peter' AND last_name='Parker'";

mysqli_query($link, $sql);
// Close connection
mysqli_close($link);
?>
```



### **PHP MySQL DELETE Query**

Just as you insert records into tables, you can delete records from table using the DELETE statement. It is typically used in conjunction with the WHERE clause to delete only those records that matches specific criteria. The basic syntax of the DELETE statement can be given with:

DELETE FROM table\_name WHERE column\_name=some\_value

#### **Example:**

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo");
// Attempt update query execution
$sql = "DELETE FROM persons WHERE first_name='John'";
mysqli_query($link, $sql);
// Close connection
mysqli_close($link);
```

#### **Exercises:**

1. To study about PHP and MySQL Basic commands and data base operation (Update and Delete)
  1. Insert the data in database of the form (that was created in experiment 6 Que 3).
  2. Retrieve the data in tabular for of inserted above data.
  3. Make the one webpage with following functionalities.

Make 3 pages

- a) Form.php
- b) Update.php
- c) Delete.php

#### **Scenario for UPDATE:**

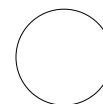
When I click on update button new page (update.php) will be open.

In this opened page you have to retrieve the data from database and shown in text box.

Also put one button and update this existing record.

#### **Scenario for DELETE:**

When I click on delete button then data will be deleted and show the message that “Data removed from database”



Page 1 Form.php  
Students Details- INSERT

**Insert New Student Details**

Enrolment No.

Name

Contact No

Email ID.

**INSERT**

**Inserted Student Details**

Enrolment No	Name	Contact No	Email ID	Action
121232322343	ABC	1234345678	<a href="mailto:abc@gmail.com">abc@gmail.com</a>	<a href="#">Update</a>   <a href="#">Delete</a>
123212333333	PQR	3244554333	<a href="mailto:pqr@gmail.com">pqr@gmail.com</a>	<a href="#">Update</a>   <a href="#">Delete</a>
876543234567	XYZ	1232349898	<a href="mailto:xyz@gmail.com">xyz@gmail.com</a>	<a href="#">Update</a>   <a href="#">Delete</a>
123432345667	LMN	3234567890	<a href="mailto:lmn@gmail.com">lmn@gmail.com</a>	<a href="#">Update</a>   <a href="#">Delete</a>

Page 2 Update.php  
Students Details- UPDATE

**Update Student Details**

Enrolment No.

Name

Contact No

Email ID.

**UPDATE**

**Review Questions:**

1. What is the difference between MySQL and SQL?
2. Summarizes different data types MySQL supports.